Алексей Федорчук
Алексей Торн

FreeBSD

Установка, настройка, использование

- Особенности инсталляции
- Теория и практика конфигурирования
- Ядро, процессы, файловые системы
- Сетевые и серверные возможности

МАСТЕР_СИСТЕМ_

Алексей Федорчук
Алексей Торн

FreeBSD:
установка, настройка, использование

Санкт-Петербург
«БХВ-Петербург»
2003

УДК 681.3.06
ББК 32.973.26-018.2
Ф33

Федорчук А. В., Торн А. В.

Ф33 FreeBSD: установка, настройка, использование. — СПб.: БХВ-Петербург, 2003. — 640 с.: ил.

ISBN 5-94157-200-X

В книге приводится описание операционной системы FreeBSD, которая сегодня является наиболее зрелой и технологически совершенной среди семейства открытых и свободных UNIX-подобных систем. FreeBSD снискала широкую популярность в сфере интернет- и интранет-технологий, сетевых и коммуникационных служб. Однако в неменьшей степени она пригодна для использования в качестве универсальной настольной системы общего назначения, что обусловлено ее устойчивостью и защищенностью, широким выбором приложений различного характера. Именно эту, менее освещенную особенность системы FreeBSD, авторы и постарались донести до читателя. Для более полного информационного обеспечения рассматриваемых тем во многих главах содержатся ссылки на дополнительные источники.

Для начинающих администраторов или пользователей FreeBSD, имеющих достаточно большой опыт в использовании ОС Windows 9x

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Анна Кузьмина*
Редакторы:	*Петр Науменко, Юрий Рожко*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Оформление серии	*Via Design*
Зав. производством	*Николай Тверских*

**Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.08.03.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 51,6.
Доп. тираж 3000 экз. Заказ № 1069
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.**

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

**Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.**

ISBN 5-94157-200-X

© Федорчук А. В., Торн А. В., 2003
© Оформление, издательство "БХВ-Петербург", 2003

Содержание

Предисловие **1**
О предмете 1
О содержании 3
Об аудитории 7
О технологии 8
О терминологии 8
Об условных обозначениях 10
Благодарности 11

ЧАСТЬ I. УСТАНОВКА И НАСТРОЙКА **13**

Глава 1. Вместо введения **15**
Преамбула личного характера 15
Общие сведения о FreeBSD 18
Из прошлого FreeBSD 22
Дистрибуция 30
Лицензии 32
Круг пользователей 35
Дополнительные источники 37

Глава 2. Конспект основных понятий **39**
Представление о системной консоли 40
Представление о командных оболочках 42
Работа с документацией 46
Кое-что о текстовых редакторах 50
Представление о файловых системах 53
Интермедия о дисках 54
О правах доступа 59
О графическом режиме работы 60
Дополнительные источники 61

Глава 3. Подготовка к установке **62**
Машина для FreeBSD 62
Получение дистрибутива 64
Дальнейшая подготовка 67
Дополнительные источники 70

Глава 4. Установка 72

Алгоритм знакомства 72
Программа sysinstall — общая характеристика 74
Алгоритм идеальной установки 89
Начальные настройки 97
Дополнительные источники 104

Глава 5. Установка программ 106

Доустановка базовых компонентов 106
Установка из пакетов 109
Коллекция портов 115
Установка непортированных программ 121
Установка оконной системы X 123
Дополнительные источники 132

Глава 6. Ядро системы 134

Ядро и его функции 134
Три и одна причина для регенерации ядра 135
О процессе 136
Структура конфигурационного файла 139
Вводная секция 140
Секция SMP Options 141
Секция CPU Options 141
Секция Compability Options 142
Секция Debugging Options 143
Секция Networking Options 143
Секция Filesystems Options 143
Секция POSIX P1003.1B 144
Секция SCSI Devices 145
Секция Miscellaneous Devices and Options 145
Структура секции Hardware device Configuration 145
Общая поддержка устройств 146
Клавиатура и мышь 146
Видеосистема 147
Системная консоль 148
Накопители 149
Сетевые карты 150
Коммуникационные порты 150
Устройства USB 151
Прочие устройства 152
Теория и практика конфигурирования 153
Проблемы и решения 157
Дополнительные источники 159

Глава 7. Загрузка и останов системы 161

Стадии начальной загрузки 161
Настройка загрузчика 164
Стартовые файлы 165
Останов системы 167
Дополнительные источники 169

Глава 8. Процессы 170

Вводные замечания 170
Понятие процесса 171
Атрибуты процесса 174
Жизнь процесса 179
Управление процессами 181
Дополнительные источники 185

Глава 9. Файлы и их атрибуты 186

Файлы и их имена 186
Классификация файлов 189
Атрибуты файлов 192

Глава 10. Файловые системы 200

Логическая организация файловой системы 200
Управление файловыми системами 202
Автоматическое монтирование файловых систем 208
Настройка файловых систем 211
Дополнительные источники 215

Глава 11. Пользователи и группы 217

Пользователи 217
Группы 221
Управление пользователями и группами 223
Дополнительные источники 228

Глава 12. Системная консоль FreeBSD 229

Возможности консольного режима 229
Видеорежимы консоли 234
Украшение консоли 239
Настройка клавиатуры 245
О консольной мыши 252
Дополнительные источники 253

Часть II. Использование 255

Глава 13. Принципы командного интерфейса 257

Команды, опции, аргументы 258
Регулярные выражения 262
Командные конструкции 264
Понятие о пользовательских сценариях 270
Дополнительные источники 274

Глава 14. Обзор пользовательских команд 275

Команды для работы с файлами 276
Создание файлов 276
Навигация по файловой системе 279
Получение информации о файлах 281
Манипулирование файлами 285
Архивация и компрессия 289
Команда *find* — универсальное средство работы с файлами 295
Команды для обработки файлов 300
Просмотр файлов 300
Сравнение, объединение и деление файлов 304
Поиск в файлах 310
Некоторые общесистемные команды 313
Некоторые полезные команды разного назначения 315
Дополнительные источники 322

Глава 15. Командные оболочки 323

Что такое Shell 323
Управляющие последовательности 326
Встроенные команды 329
Буфер истории команд 332
Управление заданиями 335
Переменные оболочки и среды 336
Псевдонимы 338
Средства профильной настройки 339
Выбор пользовательской оболочки 342
Оболочки-визуализаторы 344
Дополнительные источники 347

Глава 16. Командная оболочка zsh 349

Обзор возможностей 349
Встроенные команды 352
Командные конструкции 353
Управляющие последовательности 355
Навигация 356
Редактирование 357

Автодополнение ... 358
Управление процессами ... 358
История команд ... 358
Дисплей и помощь ... 359
Разное ... 359
Настройка zsh ... 359
Дополнительные источники ... 364

Глава 17. Текстовые редакторы ... 366

Вводные замечания ... 366
Редактор lpe ... 368
vi и Vim ... 371
Редактор jed ... 379
Редактор le ... 380
Дополнительные источники ... 382

Глава 18. Текстовый редактор joe ... 384

Обзор возможностей ... 384
Система помощи ... 386
Характерные особенности ... 388
Макрокоманды ... 392
Настройка joe ... 394
Заключительные соображения ... 398
Штатные команды joe и соответствующие им комбинации клавиш ... 400
Дополнительные источники ... 404

Глава 19. Прочие консольные приложения ... 405

Графика в консоли ... 405
Консольный Интернет ... 408
Мультимедийная консоль ... 409
Средства работы с кириллицей ... 415
Дополнительные источники ... 421

Глава 20. Оконная система X и менеджеры окон ... 422

Оконная система X ... 422
Об оконных менеджерах ... 428
Об интегрированных средах ... 435
Оконный менеджер Blackbox ... 438
Дополнительные источники ... 445

Глава 21. Текстовые редакторы графического режима ... 446

NEdit — общая характеристика ... 447
File ... 449
Edit ... 450
Search ... 452
Preferences ... 454

Shell 455
Macro 457
Windows 458
Help 459
Настройка NEdit 461
Language Modes 462
Auto-indent, *Wrap* и *Tabs* 464
Text Fonts 466
Customize Menus 468
Searching 472
Syntax Highlighting, *Statistics Line*, *Incremental Search Line* и *Show Line Numbers* 473
Make Backup Copy и *Sort Open Prev. Menu* 473
Warning 473
Initial Window Size 474
Конфигурирование .Xdefaults 474
Файлы .nedit и .neditdb 478
Макросы и Shell 479
Пример применения: html-редактор своими руками 483
Дополнительные источники 488

Глава 22. Система X Window: обзор приложений 489

Текстовые процессоры 489
Текстовый процессор Lyx 493
Электронные таблицы 501
Графика и мультимедиа 503
Дополнительные источники 508

Глава 23. FreeBSD и другие ОС 509

Распределение дискового пространства 510
Совместный запуск FreeBSD и других систем 513
Совместный доступ к данным 517
Запуск Linux-приложений 519
О приложениях DOS и Windows в FreeBSD 523
Дополнительные источники 524

Глава 24. Что день грядущий нам готовит 525

Дополнительные источники 528

ЧАСТЬ III. СЕТЕВЫЕ И СЕРВЕРНЫЕ ВОЗМОЖНОСТИ 529

Глава 25. FreeBSD в сетевом окружении 531

Основы компьютерных сетей 531
Принципы TCP/IP. Модель OSI 532
Протоколы стека TCP/IP 534

Информация в заголовках протоколов ... 535
Адресация в Интернете. Сетевая маска. Классы сетей ... 537
Три задачи поддержки сетевого взаимодействия ... 538
Что такое интерфейс? ... 538
Настройка шлюза ... 543
Распределение данных ... 545
Программы-получатели ... 545
Супердемон inetd ... 550
Разрешение имен ... 554
Сетевые настройки стартовых скриптов ... 556
Утилиты проверки работоспособности сети ... 562

Глава 26. Маршрутизация и защита сети ... 568

Маршрутизация ... 568
Статическая и динамическая маршрутизация ... 568
Трансляция адресов ... 570
Соглашение о приватных адресах ... 570
Трансляция адресов демоном natd ... 571
IPNAT — альтернативный механизм трансляции адресов ... 574
Брандмауэры ... 575
Защита терминальной службы ... 581

Глава 27. FreeBSD и основные сетевые сервисы ... 583

Соединение "точка-точка" (PPP) ... 583
Конфигурирование демона pppd ... 585
Настройка пользовательской программы ppp ... 586
Службы e-mail ... 589
Общие сведения о SMTP ... 589
Почтовые агенты ... 590
Настройка программы sendmail ... 591
Web-сервер ... 592
Как работает HTTP-сервер? ... 592
Расположение файлов сервера Apache ... 595
Утилиты пакета Apache ... 597
Директивы времени исполнения ... 597
FTP-сервер ... 598
Назначение и размещение файлов FTP-сервера ... 599
Управление FTP-доступом ... 601
Сервер DNS ... 601
Структура системы доменных имен ... 601
Файлы и программы пакета BIND ... 603
Конфигурационный файл named.conf ... 606
Сетевая файловая система ... 608
Конфигурирование сервера NFS ... 608
Демоны NFS ... 610

Конфигурирование клиента NFS 610
Демон автоматического монтирования 611
Безопасность и NFS 612
Сервер Samba 612
Особенности протокола SMB 612
Установка и конфигурирование сервера Samba 613
Безопасность сервера Samba 617
Поддержка DHCP 618
Основы протокола DHCP 618
Настройка клиента DHCP 619
Сервер DHCP 620

Послесловие 625

Предметный указатель 627

Моим детям, Оле и Вите, посвящается:
их опыт освоения компьютера послужил
для меня незаменимым источником впечатлений.

Предисловие

Рост популярности Linux в последние годы вызывает интерес и к другим представителям семейства открытых и бесплатных UNIX-подобных операционных систем (ОС), из которых самой известной является FreeBSD. Именно эта ОС и представляет предмет настоящей книги.

О предмете

Об операционной системе FreeBSD слышали многие — есть даже мнение, что это, наряду с Web-сервером Apache, один из самых удачных (вернее, удавшихся) проектов на базе движения открытых исходных текстов. Во всяком случае обычный эпитет, применяемый к FreeBSD, — это "круто". Однако мало кто из пользователей PC эту систему видел, что называется, вживе. И еще меньше — тех, кто ее использует в обыденной жизни. Впрочем, надеюсь, что по прочтении этой книги число таковых увеличится...

В отличие от Linux, FreeBSD не получила в настоящее время всенародной популярности и не поминается всеми и каждым в контексте противостояния монополии Microsoft всему остальному миру. Тем не менее она широко известна в узких кругах и насчитывает немалое число преданных поклонников. Чем же определяется (или может определяться) интерес пользователей к системе FreeBSD? Многими факторами, из которых я выделил бы пять главных.

Во-первых, FreeBSD — наиболее зрелая из всех открытых и свободных UNIX-подобных систем (да и отличных от UNIX — тоже). Хотя формально эта ОС возникла в 1993 г., на два года позже Linux, заложенные в ней концепции развиваются с 1974 г. Более того, многие из них, ныне составляющие неотъемлемую часть коммерческих UNIX, зародились именно в рамках BSD-систем — прямых предшественников FreeBSD.

Во-вторых (и это — следствие первого фактора), FreeBSD — технологически наиболее совершенна среди всего этого семейства, что выражается в ее устойчивости и защищенности, поддержке отраслевых стандартов и перспективных новшеств.

В-третьих, для FreeBSD, помимо базовых компонентов системы, доступен абсолютно весь круг программного обеспечения с открытым исходным ко-

дом, созданного в ходе развития ОС Linux, в рамках движения Open Sources, GNU, Free Software и родственных им проектов. Более того, она располагает средствами исполнения большинства бинарных приложений Linux, в том числе и коммерческих, для которых исходный код недоступен. В-четвертых, условия лицензирования FreeBSD (т. н. BSD-лицензия), предполагая открытое и свободное ее распространение, не запрещают, в отличие от Linux и программ проекта GNU (GNU is Not UNIX — типичный для сообщества Open Sources акроним, который можно перевести как GNU — это вам не UNIX), распространяемых в рамках лицензии GPL (General Public License — Стандартная Общественая Лицензия, в наиболее адекватном переводе Елены Тяпкиной), создание на ее основе закрытых программных комплексов и (или) их компонентов. Это открывает перспективы для разработки разного рода специализированных приложений, в том числе и коммерческих. В качестве примера можно привести недавно созданную Mac OS X, основанную на ядре BSD.

Наконец, последним по счету, но не по значению, особенно для отечественного пользователя, является ее концептуальная интернационализация. Конечно, список поддерживаемых ею языков не столь широк, как у исходно интернациональных дистрибутивов Linux, например, Linux Mandrake или Altlinux. В этом списке мы не увидим языков маори, эскимосского или даже казахского. Но зато качество поддержки тех языков, для которых она декларируется, более чем на высоте. В частности, базовая русификация FreeBSD выполняется уже в ходе установки. А в дальнейшем есть средства для работы с любыми кодировками русского языка.

Как следствие, FreeBSD пользуется широкой и заслуженной популярностью в сфере интернет- и интранет-технологий и коммуникаций. Существуют различные количественные оценки масштабов ее применения здесь, однако качественно все они сходятся в том, что число Web-, FTP-, почтовых и прочих серверов под ее управлением сопоставимо с таковым для всех дистрибутивов Linux или всех вариантов проприетарных UNIX-систем. И уж заведомо превосходит количество инсталляций ОС линии Windows NT/2000 в этой сфере. Это — в мировом масштабе, в российских же условиях крен в сторону FreeBSD выражен еще более резко.

Несмотря на все сказанное, литература по FreeBSD на русском языке очень бедна: до настоящего времени не было издано ни одной оригинальной или переводной книги, посвященной ей, список журнальных публикаций не насчитывает и дюжины. Основными источниками информации об этой системе служат переводы официальной документации проекта FreeBSD и очень качественные, но неполные (посвященные в основном администрированию системы) и иногда устаревшие описания Ивана Паскаля, Дмитрия Карпова и Андрея Федорова, доступные исключительно в виде online-версий на нескольких серверах (список — см. в конце *главы 1*). А главное, нет описания

возможностей FreeBSD для настольного применения, хотя по ряду причин (о которых будет говориться во вводной части) ее потенциал в этой сфере выше, чем у Linux.

Следует добавить, что и англоязычная литература по FreeBSD в объеме существенно уступает количеству книг по Linux: "в любом книжном магазине можно найти пару дюжин книг о Linux, и одну, а скорее всего ни одной книги о FreeBSD" (**http://www.osp.ru/cw/1999/08/18.htm**). И дело отнюдь не в отсутствии спроса — "появление слова FreeBSD в заголовке гарантирует интерес покупателей к продукции книжных издательств" (там же).

Это представляется несправедливым, не соответствующим роли системы в мире информационных технологий и требующим объяснений. Ряд таких объяснений приводится в статье Грэга Лиея "Что такое BSD" (**http://www.FreeBSD.org.ua/articles/explaining-bsd**). Со своей стороны рискну высказать такое объяснение: подавляющее большинство знатоков FreeBSD — это опытные (и высокооплачиваемые) администраторы серьезных компьютерных систем, которые не имеют ни времени, ни, видимо, желания заниматься литературным трудом. Написание же книги о FreeBSD без достаточного опыта практической работы с ней — затея исходно обреченная на провал.

Вследствие всего изложенного выше автор этих строк и берет на себя смелость предложить вниманию читателей книгу, посвященную установке и настройке, устройству и использованию FreeBSD. При этом он отнюдь не является профессиональным системным администратором и не претендует на лавры UNIX-гуру. Однако автор имеет собственный (и, как представляется, — успешный) опыт применения FreeBSD в качестве личной настольной системы, предназначенной для решения практических задач, возникающих в ходе научных исследований в области геологии. Кроме того, эта же система используется им и для информационного обеспечения работы научного коллектива, включающего, помимо сотрудников некоего московского академического института, также иногородних и зарубежных участников.

О содержании

Книга включает, помимо настоящего предисловия, три части, порядок изложения материала в которых определяется последовательностью знакомства пользователя с системой и этапами ее освоения.

Первая часть посвящена установке и настройке системы. Описание основных этапов дается в том порядке, в каком эти проблемы возникают перед пользователем и в котором их целесообразно (основываясь на опыте автора) решать. Примеры действий здесь даются в виде эмпирических рецептов,

основанных на вводном конспекте, без углубленного рассмотрения их внутренней сущности.

Однако начинается первая часть вводной главой общего плана. Для старта — преамбула личного характера, в которой автор попытался если не обосновать, то по крайней мере объяснить причины своей приверженности системе FreeBSD. В основном же она посвящена общей характеристике BSD-систем вообще и ОС FreeBSD в частности и их истории. Особое внимание к последней теме вызвано тем, что не последнюю роль в популярности Linux сыграла романтичная история ее создания, широко растиражированная СМИ (в том числе и не компьютерными). Однако история создания FreeBSD не менее интересна и намного более драматична, и потому заслуживает подробного изложения.

Далее в *главе 1* очерчивается сфера нынешнего использования FreeBSD и определяется ее потенциал как настольной системы общего назначения, а также круг потенциальных пользователей. Затем идет описание форм распространения FreeBSD и условий ее лицензирования.

Глава 2 также носит вводный характер. В ней содержится конспект основных понятий и терминов, предназначенный для читателей без опыта работы с UNIX-системами и необходимый для понимания всего дальнейшего материала. Кроме того, здесь содержатся относительно развернутые сведения о дисковых накопителях и их разделах в FreeBSD — без них трудно приступить к ее установке.

Подготовке к установке FreeBSD посвящена *глава 3*. Основное ее содержание — аппаратные требования системы, вопросы получения дистрибутива, а также такие подготовительные мероприятия, как изготовление загрузочных дискет.

Глава 4 посвящена собственно установке и начальной настройке системы. В ней дается подробное описание предназначенной для этого программы sysinstall, а также предлагается метод оптимальной установки системы.

Содержание *главы 5* — установка дополнительного программного обеспечения. В ней главным образом рассматривается коллекция портов — универсальная система установки и обновления программ. Затронута также установки откомпилированных пакетов и установка программ из произвольных исходных текстов. В заключительном разделе — вопросы установки системы X Window.

Глава 6 — о ядре системы FreeBSD. Вкратце рассмотрены назначение ядра и его устройство, основное же внимание — практическому вопросу: процессу регенерации ядра, включая его конфигурирование, сборку и установку.

В *главе 7* речь пойдет о загрузке и останове системы. В ней рассмотрены начальный загрузчик системы, стартовые файлы, методы безопасного прекращения работы.

Далее в первой части затронуты (в объеме, необходимом пользователю) базовые понятия UNIX, как то: процессы *(глава 8)*, файлы и их атрибуты *(глава 9)*, файловые системы *(глава 10)*, пользователи и права доступа *(глава 11)*; здесь же — максимально кратко рассматриваются вопросы администрирования системы. Серьезное внимание уделено такому важному для использования FreeBSD вопросу, как текстовая консоль *(глава 12)*.

Во второй части рассматривается практическое использование FreeBSD, в первую очередь как настольной системы. В основе чего лежит главный и универсальный интерфейс UNIX-систем — интерфейс командной строки (CLI), описываемый в *главе 13*. Далее, в *главе 14*, рассматриваются основные пользовательские команды FreeBSD, предназначенные для управления файлами и работы с текстами.

Глава 15 посвящена командным оболочкам — интегрированным рабочим средам консольного режима. А в *главе 16* подробно описывается одна из наиболее развитых командных оболочек — zsh.

Тема *главы 17* — консольные текстовые редакторы, один из основных инструментов при работе в UNIX-системах. Один из таких редакторов, Joe, объединяющий в себе простоту использования и функциональность — предмет подробного рассмотрения в *главе 18*. Ну, а в *главе 19* дан обзор прочих консольных приложений, предназначенных для работы с графикой, языковой (в первую очередь кириллической) поддержки, а также мультимедийных приложений.

В *главе 20* рассматривается оконная система X (X Window System), которая хотя и не является неотъемлемой частью FreeBSD, играет в ней, как и во всех других UNIX-системах, важную роль, являясь универсальным средством обеспечения работы в графическом режиме. Однако система X — эффективная среда для работы с текстами. И потому *глава 21* содержит описание текстовых редакторов для нее, на примере наиболее мощного представителя своего класса — редактора Nedit. В *главе* же *22* дан краткий обзор прочих приложений графического режима — текстовых процессоров, электронных таблиц, программ для работы с графикой и мультимедийной информацией.

Тема *главы 23* — совместное использование FreeBSD с другими операционными системами (Windows и Linux), средства запуска Linux-приложений, эмуляции DOS и Windows. Наконец, *глава 24* содержит краткое описание особенностей грядущих версий FreeBSD, которые, вероятно, будут доступны пользователям вскоре после выхода этой книги.

На этом завершаются принадлежащие мне две первые части книги, и я предоставляю слово моему соавтору, написавшему заключительную, третью часть.

* * *

Книга, которую вы держите в руках, представляет ОС FreeBSD с "пользовательской" стороны, как интересную, стабильную и удобную настольную систему. Но то, что сегодня изучается из любопытства, завтра — как знать? — может стать основным занятием или серьезным хобби, которое, например, свяжет с сетью Интернет всех жильцов вашего многоквартирного дома. И в этом случае FreeBSD — лучший выбор.

Действительно, несправедливо, описывая FreeBSD, "оставить в стороне" наиболее сильные стороны этой ОС, благодаря которым она завоевала признание и получила столь широкое распространение: маршрутизатор, брандмауэр и сервер сетевых служб. Развитие обеих самых популярных свободно распространяемых UNIX-систем — FreeBSD и Linux — привело их к примерно одинаковому набору предоставляемых потребителю возможностей, но качество сервисов определяется реализацией основного набора системных функций, которые закладываются при проектировании системы. И здесь-то и проявляется разница: FreeBSD — сервер сетевых служб, Linux же — настольная пользовательская система, конкурирующая с широко известными ОС фирмы Microsoft.

Некоторые серверные черты FreeBSD встроены в ядро системы (например, IP-фильтры), другие — устанавливаются из пакетов, разработанных независимыми производителями (например, Web-сервер Apache). Но все, что функционирует на FreeBSD, базируется на надежном фундаменте системных библиотек, совершенствующихся уже несколько десятилетий.

Третья часть книги *(главы 25—27)* содержит краткий обзор серверных возможностей FreeBSD на фоне описания общих задач и понятий компьютерных сетей, данного в форме, доступной даже для самого неискушенного в этом вопросе читателя.

А. Торн

Книга представляет собой последовательное изложение вопроса, подчиненное единому сюжету. Однако она не претендует на всестороннее освещение предмета — это задача, непосильная для одного человека. В частности, не будучи программистом, автор не останавливается на возможностях, предоставляемых FreeBSD для разработки программного обеспечения. Основная цель книги — продемонстрировать, что FreeBSD вполне в состоянии выполнять роль настольной ОС общего назначения для достаточно широкого круга пользователей.

Книга не содержит большого количества иллюстративного материала, за исключением минимума, абсолютно необходимого для понимания предмета. Всякого рода скриншоты в данном случае носили бы преимущественно декоративный характер. Ведь FreeBSD, как и все UNIX-системы, изначально

ориентирована на текстовое представление материала. И автор попытался показать, что текстовая форма может быть не менее выразительной, чем графическая. Поэтому все изложение сопровождается примерами применяемых на практике команд и командных конструкций, сценариев, конфигурационных файлов и т. д. Ведь писать о UNIX без команд — это все равно, что писать о живописи без репродукций, а о музыке — без нот.

Большинство из приводимых примеров в соответствующих ситуациях могут применяться "как есть" или с необходимыми модификациями. Однако автор, естественно, не может гарантировать их пригодности на все случаи жизни и даже просто работоспособности в произвольных конфигурациях. Единственное, за что он ручается — что все описанное в книге проделано им собственными руками и работало на его машинах (впрочем, достаточно разнообразных, от Celeron/400 до Pentium-4/1900).

Об аудитории

Книга адресована широкому кругу квалифицированных пользователей любых компьютерных платформ. Предварительные знания основных понятий и терминов UNIX желательны, но не обязательны. Необходимое условие — лишь желание читателя ознакомиться с новой операционной системой и возможностями ее применения.

В ходе изложения автор часто прибегает к сопоставлению с Linux. И поэтому книга может быть интересна пользователям этой операционной системы. При этом я отнюдь не призываю к отказу от Linux и поголовному переходу на FreeBSD. Однако эти системы имеют много общего и представление об одной из них может помочь пониманию другой — опять-таки сужу по собственному опыту.

В ходе работы над книгой автор имел ввиду в первую очередь задачи и проблемы своих коллег — научных работников любого профиля. И потому их можно считать целевой аудиторией книги. Однако аналогичные проблемы могут возникнуть и перед специалистами иного профиля — инженерами, например, или экономистами.

FreeBSD — идеальная система для самообразования в области информационных технологий. И потому книга может оказаться полезной для студентов, предполагающих работать в этой сфере. Хотя она и не может заменить собой фундаментальные работы в области Computer Sciences.

Наконец, я рискнул бы предложить эту книгу вниманию тех, кто только начинает знакомство с компьютером — компьютерным энтузиастам любого возраста, наделенным любопытством выше среднего уровня. Вопреки распространенному мнению, личный опыт (вернее, личный опыт моих весьма юных детей) убедил меня, что подрастающее поколение осваивается в среде команд и командных конструкций текстового режима ничуть не тяжелее

(если не легче), нежели в визуальных графических средах. И потому с первых же нажатий клавиш не вредно узнать, что мир информационных технологий не исчерпывается Windows любого рода. Соответственно, книга эта может быть не лишней и для родителей юных любителей компьютеров.

О технологии

Немаловажной стороной концепции книги является процесс ее подготовки, в ходе которой я пытался, в частности, показать, что FreeBSD располагает достаточными средствами и для задач этого рода.

Разумеется, вся книга от первого до последнего слова написана под управлением этой ОС — в текстовом редакторе Joe (предмете подробного рассмотрения одной из глав второй части). Текст набирался непосредственно в формате pure HTML, благо Joe позволяет автоматизировать ввод тэгов, не отвлекая от течения творческой мысли (в книгах не принято ставить смайлики, однако, надеюсь, читатель поймет, где они подразумеваются по смыслу).

Таблицы подготовлены в табличном процессоре sc (описанном в главе о научных приложениях второй части). Для подготовки иллюстраций использовались scr2png (скриншоты текстовой консоли) и GIMP (скриншоты приложений графического режима и окончательная доводка всех иллюстраций). Впечатление о виде книги в печатном варианте было получено путем предварительной верстки в программе Lyx.

Книга была написана под аккомпанемент песен Александра Вертинского, Юрия Визбора, Владимира Ланцберга, Тимура Шаова, воспроизводившихся с помощью mpeg-проигрывателя mp3blaster и предварительно оцифрованных посредством mpeg-кодера bladeenc. Все использовавшиеся мной программы описаны в соответствующих главах второй части.

О терминологии

В отечественной компьютерной литературе (как традиционной, "бумажной", так и сетевой) существуют две терминологические тенденции, назовем их условно пуристической и сленговой. Первая характерна для большинства издательств и "толстых" журналов, позиционирующих себя в качестве профессиональных. Суть ее — в подборе для исходно английских терминов максимально "русских" эквивалентов. Сленговая же тенденция, преобладающая в публикациях online, выражена в простой транслитерации (часто с подчеркнутым нарушением ее правил) соответствующих английских слов и аббревиатур.

Пуристическая терминология всегда ведет к утяжелению текста. Например, вместо короткого английского слова software или даже просто soft (последним не гнушаются даже издания весьма академического толка) упот-

ребляется неуклюжее описательное определение "программное обеспечение". И это еще не самый тяжеловесный оборот — читатель легко может умножить количество примеров.

Кроме того, борьба за чистоту русского компьютерного языка иногда имеет просто комический эффект — вспомним "КД-ПЗУ", внедрявшееся некогда одним уважаемым мной изданием (догадались, что это значит? правильно, просто-напросто CD-ROM). Или, скажем, замену чуждого иностранного слова "менеджер" исконно русским — "диспетчер".

Кстати, последний пример наглядно показывает основной недостаток пуристической терминологии — ее неоднозначность, вплоть до полного искажения смысла. Перевод английского термина Window Manager как "диспетчер окон" способен только ввести в заблуждение человека, недостаточно знакомого с предметом.

И потому, да простят меня ревнители чистоты Великого и Могучего, я категорически отказываюсь от пуристического подхода. Он вызывает у меня в памяти известного адмирала Шишкова, боровшегося за это еще в XVIII веке. Хотя, подозреваю, в отличие от его современных последователей, для адмирала это было просто формой эпатажа и проявлением своеобразного юмора...

Сленговая терминология тоже не свободна от недостатков: она не вполне изящна, часто выглядит чужеродной и непонятной непосвященным, подобно уголовной "фене" (хотя даже чисто русский эквивалент не более ясен при незнании сути дела). Однако у нее есть одно неоспоримое достоинство — однозначность, унаследованная от английского прототипа.

Конечно, использование околокомпьютерного (как и любого другого) сленга требует взвешенного подхода. Нелепо переносить английские кальки в области, где существует развитая, устоявшаяся и более чем адекватная русская терминология, например, в область полиграфии и шрифтовых технологий (где, к слову, русская и вообще континентальная европейская традиция очень сильно отличается от англо-американской). Однако во многих случаях именно сленговая терминология позволяет адекватно выразить суть явления. Более того, она настолько вросла в околокомпьютерный быт, что полный отказ от нее возможен только ценой потери ясности изложения.

Примером может служить поиск адекватной русской передачи термина X Window System. Обычно в качестве таковой предлагается достаточно неуклюжее сочетание "система X Window", хотя дословный перевод его — "Оконная система X" (так же не верх литературного изящества), где X — и в английском, и в русском языке выполняет роль идеограммы, а не языковой конструкции. И потому сленговое выражение "иксы" наиболее точно передает суть дела. В подтверждение чего можно сослаться на перевод книги Патрика Фолькердинга (создателя дистрибутива Linux Slackware).

Примерно такого взвешенного подхода я и старался придерживаться в своей книге. С одной стороны, я широко использую общепринятые (и общепонятные в компьютерных кругах) сленговые выражения — как ради адекватности, так и для большей выразительности. То есть — придания колорита: ведь автор, пишущий, скажем, о быте жителей Беломорья или Северо-Востока, не станет избегать местных идиоматических выражений. Так почему же при описании компьютерных будней следует отказываться от выражений, бытующих в этой среде? С другой стороны, надеюсь, я не скатываюсь до уровня жаргона времен моей юности, типа "вайтовых трузеров на голдовом зиппере", да еще "с фирмовой лабелой". В случаях полной невозможности подобрать русский эквивалент я предпочитаю дать оригинальное (английское) выражение.

И последнее. Русская терминология, относящаяся к UNIX-системам, складывалась исторически, на протяжении длительного времени и усилиями очень различных коллективов. Кроме того, в дальнейшем на нее наложилась терминология (также весьма разнородная) локализованных версий иных операционных систем. И потому она довольно далека от унификации. Заниматься таковой — не нахожу в себе ни моральных, ни физических сил (да и смысла, признаться, тоже). Поэтому при каждом первом упоминании термина я привожу все известные мне его синонимы (в необходимых случаях — и разночтения). И в дальнейшем употребляю эти синонимы, руководствуясь исключительно собственными представлениями об изяществе русской речи.

Об условных обозначениях

В книге фигурирует большое количество названий программ, команд, их опций и аргументов, управляющих комбинаций клавиш и тому подобных материй. Для единообразия и удобства восприятия автор старался придерживаться следующей легенды.

Названия операционных систем, рабочих сред, программ даны в оригинальном написании (естественно, латиницей), как правило, с заглавной буквы и без шрифтового выделения, например, FreeBSD, Nedit, WindowMaker. Примеры команд, сценариев, системных сообщений, конфигурационных файлов даются отдельными блоками в моноширинном начертании. При этом собственно команда предваряется символом приглашения командной строки:

```
$ cp file1 file2
```

отсутствующем во фрагментах сценариев, конфигурационных файлов и в системных сообщениях:

```
usage: mkdir [-pv] [-m mode] directory...
```

Моноширинным шрифтом в теле абзацев выделяются, кроме команд, их опции и аргументы, переменные, за исключением имен файлов, путей к ним.

Для выделения названий интерфейсных элементов, таких как пункты меню (**File**, **Open**), экранные кнопки (**OK**) и т. д., используется полужирное начертание основной гарнитуры текста. Им же передаются адреса сайтов (**http://freebsd.by.ru**).

Названия клавиш клавиатуры заключаются в угловые скобки (<Ctrl>). Для клавишных комбинаций (управляющих последовательностей) принят знак + (плюс) в случае одновременного их нажатия, и символ - (дефис) — при нажатии последовательном. Таким образом, комбинация <Ctrl>+<C> соответствует нажатию клавиши <C> при нажатой клавише <Ctrl>, а комбинация <Escape>-<B> отвечает последовательному нажатию сначала клавиши <Esc>, ее отпусканию и нажатию клавиши <B>. То же правило действует и в сложных комбинациях, например, типа <Ctrl>+<K>-<D>.

Если клавиши в управляющих последовательностях нечувствительны к регистру символов, последний дается в той форме, как приведен в документации. Все случаи зависимости значения от регистра символов оговариваются в явном виде (и, разумеется, комбинации клавиш даются в соответствии с требуемым регистром).

В некоторых случаях (например, в разделе о редакторе vi *главы 17*) одни и те же символы выступают, в зависимости от режима его работы, и как управляющие клавиши, и как команды. В этом случае следует обращать внимание на выделение. То есть символ <w> будет означать управляющую ("горячую") клавишу нормального режима vi, а символ `w` — команду его командного режима.

Благодарности

Первая благодарность, разумеется, создателям BSD и разработчикам FreeBSD: очевидно, без их усилий не было бы и предмета для этой книги. И, конечно же, апологетам концепции открытого и свободного программного обеспечения — на отечественных просторах их олицетворяет Алексей Новодворский. Ведь концепция эта далеко переросла границы отрасли информационных технологий, приобретя (не побоюсь громкого слова) общечеловеческое значение.

Отдаленным прототипом книги послужили циклы заметок на моих сайтах — **http://onix.nm.ru** и **http://freebsd.by.ru**. И потому хотелось бы выразить благодарность администрации серверов Новая Почта (**http://www.newmail.ru**) и By.Ru (**http://www.by.ru**) за возможность размещения на них своих страниц.

С тематикой книги тесно переплетаются мои статьи, публиковавшиеся на протяжении 2001—2002 гг. в журнале "Компьютерра" (**http://www.computerra.ru**) и

на сайте Софтерра On-Line (**http://www.softerra.ru**). Я не могу отказать себе в удовольствии сказать слова признательности в адрес коллектива редакции журнала. И в первую очередь — лично Сергея Scout'а Кащавцева и Федора Сорекса, предоставивших мне трибуну для изложения своих взглядов и мыслей.

Благодаря публикациям в "Компьютерре" я познакомился (правда, заочно) с рядом ее авторов. Из них мне особо хотелось бы поблагодарить Владимира Попова и Алексея Крюкова: их статьи (а главное, последующая переписка) немало помогли мне в ряде случаев.

Я получал и получаю большое количество писем от читателей моих "бумажных" и электронных публикаций с советами, указаниями на ошибки, уточнениями, дополнениями и прочей конструктивной критикой. И потому должен выразить признательность всем моим корреспондентам, хотя и вынужден воздержаться от приведения поименного списка. Тем не менее, особо хотелось бы отметить Павла Кайгородова и Стаса Гобунова, чьи советы очень помогли мне в изучении FreeBSD.

Я благодарен своим коллегам по лаборатории тектоники приокеанических зон Геологического института РАН. Все они относятся с пониманием к моему труду на ниве свободного софта и по возможности стараются не обременять меня выполнением моих обязанностей по должностной инструкции. Особенно это относится к моему прямому и непосредственному начальнику — зав. лабораторией Сергею Соколову, от которого я всегда получаю поддержку в своих начинаниях. Хотя подчас это и входит в противоречие с его должностными обязанностями. Этой книгой, надеюсь, хоть частично вернуть свой долг чести...

Не могу не поблагодарить нашего системного администратора (aka зав. ОНТИ) — Дмитрия Кудрявцева, с которым не только постоянно обсуждаю вопросы открытого и свободного программного обеспечения, но и получаю поддержку во всех моих (иногда нездоровых) экспериментах. Не говоря уже об оперативности, с коей он устраняет все "насморки" нашей сети, возникающие по объективным причинам...

Отдельная благодарность — моему многолетнему соратнику Кириллу Крылову, покинувшему отчизну в поисках счастья в личной жизни. Что, впрочем, не прервало нашего общения относительно открытого софта.

Хотелось бы поблагодарить и команду издательства "БХВ-Петербург": вот уже вторая книга подготовлена нами в теплой, дружеской обстановке и взаимном (надеюсь) консенсусе.

Наконец, главная благодарность — моей жене Лене, не слишком ревновавшей меня к леди Free во время работы над книгой.

ЧАСТЬ I

Установка и настройка

Глава 1

Вместо введения

В этой главе будет дано общее представление об ОС FreeBSD, ее истории, условиях распространения и сфере применения. Однако сначала я попытаюсь рассказать, почему я люблю эту систему — возможно, мои мотивы покажутся читателю убедительными.

Преамбула личного характера

Чудо женщина жила,
Не добра и не мила,
Но к себе она влекла
Джентльменов без числа
От Дувра до Хольспая...
Африкой она была,
Южной Африкой была,
Нашей Африкой была,
Африкой без края...

Редьярд Киплинг

Прежде чем начинать рассказ о системе, не худо бы убедить читателя в том, что система эта лично ему нужна и полезна. Я же даром убеждения не обладаю — иначе подался бы в религиозные пророки или комсомольские секретари. И потому, вместо обоснования несравненных достоинств FreeBSD, просто попробую рассказать, почему и за что я ее люблю. Хотя, когда речь идет о любви, понятия "почему" и "за что" кажутся неуместными. Или, по крайней мере, рациональному истолкованию не поддающимися. Так что для начала — преамбула личного характера. Читатели, не имевшие опыта работы с UNIX, обнаружат в ней несколько незнакомых слов. Не пугайтесь, это не ругательства, что я постараюсь показать во второй главе.

Мое шапочное знакомство с FreeBSD началось лет пять назад — на этапе первичного приобщения к миру свободного софта. Рыская по книжным магазинам в поисках дистрибутивов Linux, только что появившихся в широкой продаже, я то и дело натыкался на коробки с изображением чертика с вилами и надписью — FreeBSD. Чем-то они неосознанно привлекали ме-

ня — возможно, загадочностью: об этой системе в русскоязычной литературе имелись только отрывочные упоминания. Тем не менее опробовать эту систему я не рискнул — это тогда казалось мне, старому пользователю DOS, избалованному годами работы в Windows, непосильным.

Непосредственное же общение с FreeBSD пришлось на последние дни ушедшего тысячелетия. К тому времени я, безжалостно истребив на своей машине известное произведение Самой Великой Софтверной Компании, уже более или менее освоился с Linux, и даже научился выполнять в нем все жизненно важные для меня задачи. Скажу больше — Linux как настольная система меня вполне устраивал.

Казалось бы, чего еще желать? Однако неискоренимое любопытство требовало знакомства с чем-то новым. И напрашивающееся решение для удовлетворения любопытства — установить FreeBSD. Что я и не замедлил проделать...

Первым чувством, охватившим меня еще в процессе установки, было чувство растерянности. Которое только усилилось после запуска новой системы. При полном, казалось бы, внешнем сходстве с Linux, все во FreeBSD было иным: другая номенклатура накопителей, другое представление о дисковых разделах, другая консоль, другие командные оболочки...

Помнится, по первости я немало затратил времени, чтобы отыскать в консоли FreeBSD средство для пролистывания экранов — там в этом качестве выступали клавиши <PageUp>/<PageDown> при включенном режиме <ScrollLock>, а отнюдь не комбинация клавиш <Shift>+<PageUp>/<PageDown>, как в Linux-консоли.

Относительно же командных сред достаточно сказать, что по умолчанию таковой во FreeBSD выступал некий /bin/sh — как потом выяснилось, точный римейк традиционно-архаичного командного интерпретатора Shell Борна, средства интерактивной работы в котором не идут ни в какое сравнение с таковыми оболочек bash или tcsh. И с непривычки производит он впечатление если не убожества, то уж предельного аскетизма — точно.

Совершенно иной подход к процессу установки... Идеологически иной подход к локализации... Даже X, который как бы и в Африке — X, также выглядел и настраивался несколько иначе. Короче говоря, не знаешь, за что хвататься в первую очередь — за настройку ли среды, консоли, установку русских буковок или что другое...

И еще: особенностью FreeBSD казалось почти полное отсутствие средств автоматизации настройки — хотя бы типа аналогов утилиты linuxconf. Далеко не сразу понял я, что программа sysinstall, используемая при установке — вместе с тем и почти универсальный инструмент конфигурирования. Поначалу все требовало ручной правки конфигурационных файлов в текстовом редакторе. В качестве коего в базовом наборе предлагались либо vi в его

классической ипостаси, либо ее — редактор хоть и простой (местами — даже очень простой, как реклама незабвенной фирмы "Сэлден" — или «Сэдом», уж простите, запамятовал), но непривычный.

Благо, как ни странно, FreeBSD оказалась очень хорошо документирована, в том числе — и на русском языке. Количественно, конечно, русскоязычные источники информации по этой системе очень уступали таковым для того же Linux. Но зато качественно — то, что имелось, было на высоте. И, главное, можно было быть уверенным, что все они относились к одной и той же системе. Тогда как для Linux в ряде случаев было трудно понять, к какому конкретному дистрибутиву относится данный FAQ или еще какой HOW TO. Правда, и тут не обошлось без ложки дегтя — изрядная часть этой документации относилась к версиям FreeBSD весьма преклонного возраста.

Главное же — в этой документации очень много говорилось о системном администрировании, и почти ничего — об использовании в мирных (т. е. настольных) целях. Что подкреплялось высказыванием одного из моих корреспондентов, что держать FreeBSD на десктопе — сродни извращению...

Вопрос казался ясным — стирать FreeBSD и возвращаться к уже привычному Linux. Однако что-то иррациональное влекло меня к этой системе, не позволяя запустить fdisk. Тогда я списывал это на упрямство и уязвленное самолюбие — за десять лет околокомпьютерной жизни не бывало, чтобы я в конце концов не понял, как что-то устроено и работает.

В итоге через некоторое время я начал понимать логику установки и конфигурирования FreeBSD, строгую красоту ее настроек. Конечно, работа в ней требует существенно большей дисциплины мысли и действий, чем работа в т. н. user-ориентированных дистрибутивах Linux (не говоря уже о Windows). Однако чисто эмпирические алгоритмы действий в стандартных (и даже не очень стандартных) ситуациях нарабатывались достаточно быстро. И я прочно, казалось бы, записал FreeBSD в свой рабочий арсенал...

Однако затем колесо фортуны свершило очередной свой оборот. И, не корысти ради, но только хлеба насущного снискания для, на изрядный промежуток времени мне пришлось вернуться к Linux — на FreeBSD не оставалось ни времени, ни места на диске. Но воспоминания о ней, как об оставленной в силу обстоятельств возлюбленной, продолжали греть душу...

И потому я с вожделением ожидал очередного периода всенародного рождественского запоя в ночь с 24 декабря на 14 января. Это — время, когда на мою службу запрещается доступ, я остаюсь в своей заснеженной деревне без связи с внешним миром и волен заниматься чем угодно. В этот раз перспектива на ближайшие три недели была ясна — я возвращаюсь к моей леди Free.

Должен заметить, что к тому времени мне стало очень не нравиться направление, в котором развиваются наиболее распространенные дистрибутивы Linux. Пресловутая ориентированность на конечного пользователя привела

к тому, что система утрачивала управляемость — средства автоматического конфигурирования стремились все сделать за тебя, подчас даже не спрашивая согласия. Прямо так, как это имеет место в Windows. Но ведь не для того стиралась Windows с диска, чтобы получить ее новую реинкарнацию — да еще подчас и реализованную в этом плане не лучшим образом.

Так что жребий был брошен, Рубикон перейден, CD-диск вставлен в привод, Reset нажат... Начинался процесс очередной инсталляции. И тут я наконец понял причину своей, чисто инстинктивной, симпатии к этой системе. FreeBSD — это женщина, более того — леди. Да, леди весьма суровая и не склонная к сантиментам. Она, подобно киплинговской Африке, не добра и не мила для случайных пришельцев. Но своим преданным поклонникам — всегда ответит взаимностью. По крайней мере, я на такую взаимность надеялся твердо...

Примечание

Маленькое отступление: эпиграф из Киплинга дан в переводе Гуткина, цитата по памяти (хотя за точность почти ручаюсь). Нынче его переводы мало известны (я с ними знаком по самодельному машинописному сборнику начала 60-х годов XX века, утерянному в ходе кочевой жизни). Если кто знает о их переиздании или нахождении в Сети — буду признателен за информацию.

Что же до демонической природы системы, подчеркиваемой эмблемой черта с вилами... Истина где-то близко. Она, подобно демоническим женщинам древних мифов, готова подвергнуть своего избранника тяжким испытаниям и требовать от него соответствия героическому идеалу. Но, при таком соответствии, в конце концов дарует победу.

Не берусь утверждать, что соответствую идеалу леди Free в полной мере. Однако записал себя в число ее преданных поклонников. И пока не имел повода раскаиваться в своем решении...

Общие сведения о FreeBSD

Правда, очень правда то,
Что она такая.
Африкой ее зовут,
Южной Африкой зовут,
Нашей Африкой зовут,
Африкой без края...

Редьярд Киплинг

Покончив с лирическим прологом, пора познакомиться с FreeBSD поближе. Разумеется, при условии, что мне удалось если не убедить читателя в необходимости этого, то хотя бы заинтересовать. Итак, что же такое FreeBSD?

Как ни странно, система эта, широко известная в узких кругах (и весьма в них популярная), мало знакома околокомпьютерным народным массам — намного меньше, чем ее собрат по Open Sources, Linux, приобретший в последние годы изрядную популярность. И потому начнем с самого начала.

Разнообразные формальные определения понятия "система FreeBSD" можно найти на сайте проекта FreeBSD (**http://www.freebsd.org**) и в сетевой документации по ней. Суть их сводится к тому, что это (цитирую один из переводов FreeBSD Handbook — **http://www.FreeBSD.org.ua**) "великолепная операционная система, основанная на 4.4BSD-Lite2", которая "работает на компьютерах с архитектурой Intel (x86) и DEC Alpha" и предоставляет такие функции, как вытесняющая многозадачность, многопользовательский доступ, защищенный режим работы процессора, сетевые возможности по протоколу TCP/IP и прочее, и прочее, и прочее. К чему обычно добавляется, что система эта 32-разрядная (64-разрядная — для архитектуры Alpha) и является UNIX-подобной.

Чтобы более не возвращаться к этому вопросу, замечу, что о версии для DEC Alpha (ныне, как известно, уже давно не DEC, а Compaq) в этой книге не будет говориться ни слова — за незнакомством, неактуальностью и ввиду близости (к сожалению) безвременной кончины этой архитектуры. Так что все, написанное ниже, имеет силу только в отношении архитектуры PC.

Все сказанное, безусловно, верно. Однако мало проясняет существо дела для пользователя, знакомого по преимуществу (если не исключительно) только с ОС семейства Windows. И потому требует несколько более развернутого комментария.

Начну в обратном порядке, с последнего положения — о UNIX-подобии. Оно означает, что FreeBSD представляет собой функциональный аналог коммерческих (или, как сейчас стало модно говорить, проприетарных) UNIX-систем (к коим относятся, например, Sun Solaris или IBM AIX), но, однако, ОС UNIX не является, так как не содержит кода UNIX. А главное — не получила права на эту торговую марку от ее владельцев.

Утверждение это (хотя, вероятно, правильное юридически) несколько спорно с фактической точки зрения. Что можно проиллюстрировать сравнением с другой ОС, которую также принято относить к UNIX-подобным — с широко известным Linux. Каковой, действительно, является воспроизведением функций UNIX с "чистого листа". Система же FreeBSD имеет с проприетарными UNIX-системами общих предков и долгое время развивалась совместно с ними. Более того, многие функции, без которых немыслимы современные UNIX-системы, появились впервые именно в прототипах BSD-систем.

Впрочем, в некоторых источниках FreeBSD без лишнего лицемерия называют одной из разновидностей UNIX. Но о UNIX-подобии вообще, об истории FreeBSD и связанном с ней юридическом крючкотворстве мы еще поговорим. Пока же достаточно осознать, что FreeBSD имеет те же особен-

ности и предоставляет пользователю в общем и целом практически те же возможности, что и любой из представителей клана UNIX.

Одна из таких особенностей (последняя в порядке перечисления) — 32-разрядность, интуитивно понятна. FreeBSD функционирует на истинно 32-разрядных процессорах Intel (и совместимых с ними, типа AMD или Cyrix), используя их адресацию памяти, защищенный режим и прочие архитектурные особенности. Естественно, она категорически откажется работать на машинах класса XT или AT (буде у кого такое еще сохранилось).

Естественно, что версия FreeBSD для 64-разрядных процессоров Alpha является 64-разрядной же. Однако, как уже сказано, эту тему мы замнем для ясности.

Обратимся теперь к функциональности FreeBSD. Здесь наибольший интерес для пользователя представляет, пожалуй, многозадачность. Именуемая обычно истинной, реальной, приоритетной или вытесняющей. Это означает, что, в отличие от Windows, также декларирующей свою многозадачность, FreeBSD распределяет ресурсы машины между задачами, руководствуясь системой динамически изменяемых приоритетов. И, следовательно, ни одно из приложений не в состоянии парализовать выполнение иных процессов в системе (чему в Windows мы тьму примеров сыщем).

Во избежание недоразумений сразу подчеркну, что здесь и далее под термином Windows я подразумеваю исключительно системы линии Windows 3.xx/95/98/ME. ОС линии NT/2000/XP, как известно, не имеют с ними почти ничего общего, кроме имени. И потому в случае необходимости я их и буду именовать для краткости — NT.

С реальной многозадачностью FreeBSD тесно связана ее защищенность: любая запущенная в этой системе задача функционирует в своем адресном пространстве, изолированном от воздействия иных задач. Следствие для пользователя — устойчивость системы: FreeBSD крайне сложно "уронить" сколь угодно скверно написанной программой. Забегая вперед, замечу, что мне известен чуть ли не единственный способ это сделать — запуск программы xf86cfg для конфигурирования системы X Window в графическом режиме.

Из многозадачности и защищенности следует и многопользовательский доступ к FreeBSD как системе и к ресурсам машины (машин), ею управляемым. Кроме того, это позволяет обеспечить защиту пользовательских данных как от случайных ошибок, так и от враждебных на них посягательств.

Конечно, все перечисленные особенности не являются уникальными для FreeBSD — те же самые слова может сказать о своей системе пользователь и Linux, и OpenBSD, и любой иной UNIX-системы. И к тому же все эти несравненные достоинства не способны доставить много радости пользователям, если не подкреплены приложениями, использующими их на полную катушку. Что же касается приложений — то "их есть у меня".

Начнем с того, что основная прикладная база FreeBSD (командные оболочки, средства работы с текстами, от редакторов до систем верстки, система X Window, графические редакторы, коммуникационные средства и программы для работы с Интернетом и многое, многое другое) — общее достояние всех открытых UNIX-подобных систем. Представляющее собой, как правило, воспроизведение или усовершенствование классических UNIX-утилит и приложений. А поскольку в подавляющем большинстве они распространяются свободно и в исходных текстах (на условиях лицензий GPL, BSD и родственных), то легко могут быть скомпилированы для работы в любой из таких систем — непосредственно или после минимальной адаптации.

Однако это — далеко не всегда необходимо: создатели FreeBSD уже позаботились о своих пользователях, и система эта распространяется с большим количеством скомпилированных бинарных пакетов (т. н. packages), работоспособных после несложной процедуры установки.

Более того, все эти приложения и утилиты не просто доступны пользователю в исходных текстах. Благодаря т. н. системе портов они могут быть пересобраны им самостоятельно, с собственными настройками, и — в максимально современном виде: система портов берет на себя заботу об отслеживании изменений в многочисленных проектах, связанных с основным проектом FreeBSD, поиске наиболее свежих версий, контроле зависимостей между взаимосвязанными компонентами, наложении необходимых заплат (patch) и тому подобные функции. Возможна также полная пересборка системы — вплоть до ядра, общесистемных библиотек и компиляторов, например, с оптимизацией под конкретную систему...

Следует заметить, правда, что ныне некоторые UNIX-ориентированные программы разрабатываются только в версиях для Linux. Особенно это касается квазикоммерческих приложений, которые, к тому же, часто недоступны в исходных текстах. И потому круг приложений для FreeBSD несколько уже, чем для Linux. Однако и это далеко не смертельно: FreeBSD располагает средствами для запуска бинарных Linux-приложений. Большинство из которых не только вполне работоспособно, но даже, в отличие от эмулируемых Windows-программ, функционирует практически без падения производительности. Высказывается даже мнение, что Linux-приложения функционируют под FreeBSD с большими быстродействием и устойчивостью.

На последнем, впрочем, не настаиваю — собственных количественных данных по сему вопросу у меня нет. Однако, думаю, сказанного достаточно, чтобы если не присоединиться к первой фразе из процитированного выше определения FreeBSD, то по крайней мере признать ее обоснованность. При этом я отнюдь не хочу сказать, что система Linux, например, плоха: обе они имеют свои особенности, и выбор между ними, в известной мере, вопрос вкуса и соответствия конкретным задачам (а то и просто случая).

Впрочем, к проблеме выбора я еще вернусь в послесловии к этой книге. А пока интересно было бы посмотреть, как FreeBSD дошла до жизни такой. Для чего потребуется обратиться к ее истокам...

Из прошлого FreeBSD

Древнее золото редко блестит,
Древний клинок — ярый.
Выйдет на битву король-следопыт:
Зрелый — не значит старый.

Дж. Р. Р. Толкиен "Властелин Колец" (перевод А. Кистяковского)

Думаю, не ошибусь, если скажу, что популярность Linux в изрядной мере обусловлена романтической историей его возникновения, напоминающей рождественскую сказку (для взрослых) о бедном студенте. Историей, к тому же, широко распространенной не только компьютерной, но и, так сказать, общенародной прессой. История создания FreeBSD такой известности не получила. Однако она не менее интересна и, как мне кажется, гораздо более драматична.

Формально возникновение FreeBSD относится к 1993 г. — к этому времени Linux существовал уже около двух лет. Однако на самом деле история FreeBSD уходит в седую, по компьютерным меркам времени, древность.

Мне не довелось быть ее свидетелем — в те времена, почти былинные, высшим достижением компьютерной мысли для меня был даже не калькулятор, а "железный феликс" на пару с логарифмической линейкой. И потому рассказать ее могу, только основываясь на литературных источниках — как я их понимаю. Исчерпывающий же их список (для русскоязычных, разумеется, данных) можно найти в разделе дополнительных источников, завершающем эту главу.

Начнем по порядку — не так давно, 1 января 2000 г., все прогрессивное человечество широко отметило (в узком кругу) 30-летний юбилей системы UNIX. Не то, что бы она была создана именно в этот день — просто UNIX-машины во всем мире отсчитывают свое системное время с 1970.01.01 (в американском формате дат). А поскольку корни FreeBSD восходят к первым версиям UNIX, эту же дату можно считать началом и ее праистории.

Что касается UNIX — эта операционная система создавалась в конце 60-х — начале 70-х гг. прошлого века группой товарищей из Bell Labs (подразделения могучей корпорации AT&T) на аппаратуре, подлежащей списанию в утиль. И, насколько я могу судить по источникам, — на голом энтузиазме, без особой поддержки дорогих руководителей.

Прошу прощения, но в предыстории FreeBSD я по возможности воздержусь от перечисления имен. Конечно, мир информационных технологий должен знать своих героев. Но он их и так знает. Кроме того, с одной стороны, UNIX и все ее производные — плод труда такого количества людей, что для перечисления их имен не хватит и объема "Капитала". С другой же — многие из этих имен оказались столь известными, что перечислять их здесь кажется неуместным — это будет выглядеть так, будто я, подобно Фурманову, к чужой славе примазываюсь...

На дальнейшую судьбу UNIX огромное влияние оказали юридические коллизии текущего момента. Незадолго до создания этой системы корпорация AT&T подверглась антимонопольному преследованию (подобно Microsoft ныне), в результате чего потерпела поражение в правах — на деятельность ее был наложен ряд ограничений. В частности, насколько я понимаю (я не силен в отечественной юриспруденции, а уж в американской — тем более), она не имела права торговать программными продуктами, в число коих попадала и новорожденная UNIX.

Однако не пропадать же добру, созданному как бы и само собой, но — на проприетарном оборудовании. И юридический владелец UNIX (тогда еще никто и не предполагал, что слово это станет торговой маркой, за право использования которой придется платить деньги), начиная с 1974 г., стал передавать исходные ее тексты в университеты — "в образовательных целях", как задумчиво пишут в источниках.

Это не было свободным распространением в смысле проекта GNU или движения Open Sources (движения за Открытый код), хотя система (вернее, в то время — не более, чем ее прототип) и передавалась в исходных текстах с правом их изучения, модификации, доработки и прочего потрошения. Однако, во-первых, это требовало обладания лицензией на исходный код UNIX, которая передавалась AT&T вместе с таковым, но — за деньги. Хотя, как пишут, и символические — уж не знаю, сколько это составляло по американским масштабам середины 70-х.

А во-вторых, и это — главное, условия этой самой лицензии не допускали дальнейшего свободного распространения ни системы целиком, ни отдельных ее компонентов, содержащих исходный код UNIX. Именно этим и была заложена основа дальнейшего сюжета.

Однако до юридических коллизий было еще далеко. А пока университеты с радостью приобщались к новой операционной системе, в которой были реализованы все передовые идеи того времени. И, к тому же, в принципе способной функционировать практически на всем спектре тогдашнего оборудования. Напомню, что речь идет о середине 70-х годов прошлого века — Стив Джобс еще не помышлял о создании компьютера и использовал родительский гараж по прямому назначению, а Билл Гейтс не освободил мир своим MS-DOS от засилья CP/M.

Итак, предысторию FreeBSD можно исчислять примерно с 1974 г., когда исходные тексты UNIX попали, в частности, и в Университет Беркли (штат Калифорния) — учреждение, известное всем, интересовавшимся историей т. н. точных наук, давшее миру не одно поколение достойнейших их представителей. Именно сюда UNIX "вписался с точностью патрона, досланного в патронник" (Олег Куваев). И здесь с лихвой выполнил свои "научно-образовательные цели"...

В Беркли, в условиях открытого общения профессиональных специалистов в области Computer Sciences, система UNIX медленно, но верно превращалась именно в то, чем она стала ныне. И в значительной мере — именно усилиями трудящихся из университета, объединенных в Computer System Research Group (CSRG), финансировавшуюся в сугубо мирных целях, как не трудно догадаться, Министерством обороны США. Достаточно сказать, что такие неотъемлемые составляющие любой UNIX-системы, как протокол TCP/IP, система виртуальной памяти, концепция сокетов и многое другое, ведут свою родословную непосредственно из Беркли.

Группа CSRG, начиная с 1976 г., распространяла свои достижения на магнитных лентах под названием Berkeley Software Distribution, что и дало в дальнейшем имя BSD-системам. Это первичное понимание аббревиатуры BSD, которое не следует путать с Berkeley Software Development (или Berkeley Software Design, мне попадались обе расшифровки этой аббревиатуры) — фирмой, выпускавшей до недавнего времени BSDi, коммерческий клон BSD-систем.

Конечно, система UNIX развивалась не только в Беркли. Во второй половине 70-х годов появилось немало ее клонов, оказавших то или иное влияние на систему. Однако они отмирали, сливались, перекрещивались, и имена их ныне помнят только очевидцы событий и особо интересующиеся вопросом. А в результате к году 1978-му реальностью стало существование двух линий развития UNIX: System III/V и BSD UNIX.

Первая линия, немало позаимствовав из "берклианских" достижений (если не в плане кода — здесь судить не берусь, то в идейном смысле — точно), носила сугубо проприетарный характер. Именно она легла в основу большинства современных (т. е. ныне здравствующих и развивающихся) коммерческих UNIX-систем, таких, как HP-UX, AIX, Solaris.

Линия же BSD UNIX дописывалась, доделывалась и переделывалась по преимуществу все на той же "берклианской" почве на протяжении более чем десятилетия. В 80-х гг. последовательно появлялись ветви 1BSD, 2BSD, 3BSD, дававшие, в свою очередь, боковые побеги, в том числе и коммерческие (SunOS, например, или A/UX — уже встарь были попытки приобщения Macintosh к миру UNIX, вылившиеся ныне в Mac OS X). Однако наиболее славная судьба была уготована ветви 4BSD, особенно последней ее версии, 4.4BSD — уже прямому прародителю FreeBSD.

В итоге к рубежу 90-х годов выяснилось, что исходного (проприетарного) UNIX-кода в составе берклианских UNIX осталось не так уж и много. И родилась идея создания полностью открытой операционной системы, распространяемой свободно и в исходных текстах. К этому же времени прекратилось (не иначе как вследствие распада мировой системы социализма) финансирование группы CSRG, и она столь же благополучно распалась — все в жизни имеет свою оборотную сторону, даже крах коммунистической идеологии. Но дело ее не пропало. Его подхватили, расширили, укрепили и закалили в боях многие из бывших членов CSRG.

Поскольку здесь начинается уже собственно история FreeBSD, пора вспомнить и о персоналиях. Именно созданием общедоступной UNIX-системы, причем — на общедоступной же платформе, сиречь Ix86, озаботились Вильям и Линна Джолитц. Базируясь на одном из побегов ветви 3BSD — BSD Net/2, они дописали недостающие компоненты и создали 386BSD — первую BSD-систему, претендовавшую на звание открытой в собственном смысле слова. И, насколько я понимаю, первый берклианский побег, портированный на PC (IBM-совместимые компьютеры, как их тогда еще задумчиво называли).

По словам очевидцев, система эта не была еще вполне готовой к употреблению. Однако она активно исправлялась и улучшалась весьма широким кругом разработчиков, благодаря чему возник институт patchkit — корректирующего набора, позволяющего превратить 386BSD в работоспособную операционку.

Поддержание такого комплекта заплат, судя по всему, оказалось задачей хлопотной и не очень благодарной. Вероятно, именно поэтому основоположник 386BSD, Билл Джолитц, к началу 1993 г. "находился в состоянии полного пренебрежения к ней" (здесь и далее — свидетельствует очевидец и активный участник событий, Джордан Хаббард, перевод проекта Русской документации FreeBSD — **http://www.freebsd.org.ua**).

Однако и в дальнейшем история развивалась, да простят меня читающие это дамы, почти по товарищу Ленину и "Трем поколениям". Впрочем, можно уподобить ее и истории возвращения в мир Кольца Всевластья — в ней столько же неслучайных совпадений, приведших, после многих бед и мытарств, к благоприятному исходу (хотя до того, последнего и настоящего, исхода истории FreeBSD, надеюсь, еще очень и очень далеко).

В том же 1993 г. три (опять же) последних координатора "Заплаточного проекта" — упомянутый выше Джордан Хаббард, Нейт Вильямс и Род Граймс, — решили "привести промежуточный снэпшот 386BSD в порядок, исправив множество проблем, которые механизм patchkit не мог решить". Это предполагалось сделать путем предоставления промежуточных "очистных" снэпшотов. Однако планы тройки по борьбе с заплатами "были

невежливо оборваны, когда Билл [Джолитц — *А. Ф.*] внезапно решил забрать его [вероятно, свои — *А. Ф.*] санкции у проекта без любых ясных комментариев, что должно быть сделано вместо этого".

Однако и это не очень повредило делу. К проекту присоединились Джулиан Элишер и Дэвид Гринмен. Именно последнему принадлежит заслуга изобретения имени нового проекта — FreeBSD и приобретения на него права собственности.

Вахта же Хаббарда выразилась в том, что он "связался с Walnut Creek CDROM с мыслью о путях последующего улучшения каналов распространения FreeBSD для множества невезучих без доступа к Internet. Walnut Creek CDROM не только поддержал идею распространения FreeBSD на CD, но также пошел далеко вперед и предоставил проекту компьютер для работы и быстрый доступ к Internet. Без почти беспрецедентной веры Walnut Creek CDROM, в то время полностью неизвестный проект [видимо, проект FreeBSD — *А. Ф.*], вряд ли FreeBSD зашел далеко и так быстро, как сегодня."

Не могу удержаться от комментария: странные все же бывают компании в этом бездушном и бездуховном мире чистогана, где все покупается и все продается... Вспомним, что примерно в те же годы Walnut Creek приложила руку и к распространению одного из первых дистрибутивов Linux — Slackware Патрика Фолькердинга. Не знаю, сколько уж чистогану они на этом нахапали — в додефолтные времена дистрибутивыs Slackware и FreeBSD от Walnut Creek продавались у нас (в Москве — компаниями "Медаихауз" и "Электротех Мультимедиа") по цене, посильной даже научному сотруднику с получки. Как память о тех временах, у меня по сию пору хранится Slackware 3.2 в фирменном исполнении Walnut Creek — бокс о шести дисках, даже с печатным руководством. И товарный чек на него с суммой, равной (#_версии)×100 руб.

Однако вернемся к нашей истории. В декабре 1993 г. совместные усилия проекта FreeBSD и Walnut Creek обрели зримое воплощение в виде FreeBSD 1.0, распространявшейся как с FTP-серверов (вспомним — тогда это был почти единственный способ получения софта, за исключением коммерческого "коробочного"), так и на CD. Успех был закреплен выходом релиза 1.1 в мае 1994.

Ветка FreeBSD 1.x базировалась все на той же Net/2, что и произведение Джолитца, из которого она включила многочисленные дополнения. Кроме того, существенным компонентом ее стали утилиты и приложения проекта GNU. Все это были открытые и свободные разработки. Однако исходная лента Net/2 изначально содержала некоторое количество проприетарного UNIX-кода. По словам знатоков, его было немного, но это были именно критически важные фрагменты, превращавшие берклианские разработки в цельную работоспособную систему.

Следует заметить, что на основе 4BSD в это же время развивались еще два проекта BSD/OS и NetBSD. Первый имел как бы коммерческий статус и ныне воплотился в BSDi — UNIX-подобную систему, распространяемую за деньги, но с исходными текстами (последние — за отдельную мзду). Система эта за последние два года была перепродана дважды — сначала все той же Walnut Creek, затем — Wind River, и судьба ее остается неясной. Впрочем, для нас она вряд ли представляет практический интерес...

Проект NetBSD зародился даже несколько раньше, чем FreeBSD (первый ее выпуск, 0.8, датируется апрелем 1993 г.), и также имел целью реализовать полностью открытый и свободный вариант UNIX. В отличие от FreeBSD — с упором на максимально возможную мультиплатформенность: ныне трудно поверить, что в начале 90-х это дело казалось очень актуальным, и ему предрекали успех.

Правда, именно тогда в PC Magazin появилась статья (к сожалению, не помню автора) под зловещим названием: "Через 10 лет все платформы, кроме IBM PC, уйдут в небытие". Десяти лет еще не прошло, но к тому дело идет семимильными шагами — если говорить о платформах настольных. Соответственно, и популярность NetBSD оказалась весьма скромной. Правда, она дала побег — OpenBSD, в которой мультиплатформенность надстроена ориентацией на максимальную безопасность. Кто знает, может, в свете недавно минувших событий это будет востребовано...

Тут-то господа правообладатели, чуя наживу, и напомнили разработчикам из Беркли о своих правах. А права на исходный код UNIX и торговую марку, нареченную этим именем, приобрела у AT&T фирма Novell. В тот момент одержимая, подобно сиятельному Камильбеку из "Повести о Ходже Насреддине", хватательным рвением: среди ее приобретений того времени были и лучший текстовый процессор всех времен и народов (без иронии!) — WordPerfect, и DR-DOS с ее базировавшимся на GEM графическим интерфейсом. Да-да, Windows — далеко не первая графическая оболочка над DOS, в те годы с ней успешно тягались, помимо GEM, и DeskView, и Geoworks; иных уж нет, а те далече...

Начался "вялотекущий судебный процесс о легальности версии Net/2 из Беркли". В результате юридического сутяжничества из системы, лежащей в основе FreeBSD, были изъяты все следы частнособственнического кода — а, повторяю, речь шла именно о критически важных фрагментах. Гильотинированная версия получила имя 4.4BSD-Lite, и всем претендентам на BSD-наследие было рекомендовано в добровольно-принудительном порядке перейти на ее использование. Правда из врожденного американского гуманизма в рамках проекта FreeBSD было разрешено выпустить еще одну Net/2-based версию — ею стала 1.1.5.1, последняя ласточка в ветке 1.x...

Катастрофа проекта казалась неизбежной — легким движением руки цельная и работоспособная система превратилась в симпатичнейшего уродца,

которого можно было только "приласкать, обогреть". Но — "приключения никогда не кончаются". И снова — слово Джордану:

"Тогда FreeBSD приступил к сложной задаче — буквально полному изобретению себя из абсолютно новой и довольно неполной системы 4.4BSD-Lite. «Lite» был в прямом смысле light потому, что из него удалили большие куски кода, необходимого для создания реально загружающейся системы... и фактически порт 4.4BSD для платформы Intel был очень неполным".

Реинкарнация недостающих фрагментов заняла около года. И в итоге первая версия FreeBSD — 2.0, несмотря "на множество недотесанных углов", снискавшая значительный успех, а главное — к лицензионной чистоте которой не смог бы придраться ни один сутяга, вышла в декабре 1994 г. Именно она положила начало традиции, не прерывающейся и поныне. Традиции, сразу же (в июне 1994 г.) продолженной в более устойчивой и простой для установки FreeBSD версии 2.0.5.

А дальше — дальше была плавная эволюция. Примерно два-три раза в год выпускается новая версия системы (2.1.x, затем — 2.2.x), она обрастает приложениями и утилитами (значительная часть которых происходит из проекта GNU и Фонда свободного программного обеспечения), совершенствуется ядро, улучшается (как это ни странно для чисто американской по происхождению системы) интернациональная поддержка.

В ноябре 1996 г. происходит событие, определившее структуру разработки FreeBSD до сегодняшнего дня: ветка 2.x.x была выведена из активной разработки, получив имя STABLE. Отныне, вплоть до последнего релиза (2.2.8 в ноябре 1998 г.), в ней лишь исправляются ошибки и вносятся мелкие безопасные изменения. А все долговременные и принципиально новые разработки концентрируются в версии 3.0-CURRENT. Каковая претворяется в STABLE в октябре 1998 г.

С этого времени и поныне единовременно развиваются две ветки FreeBSD — STABLE, предназначенная для широкого применения, и CURRENT, ориентированная главным образом на разработчиков и энтузиастов. Так, в январе 1999 г. обособляется ветка 4.0-CURRENT, обретшая статус стабильной в марте 2000 г. и сохраняющая его по сей день. Одновременно происходит очередное ветвление — появляется 5.0-CURRENT, воплощающая перспективы FreeBSD.

Однако и прежние, по отношению к текущей стабильной, ветки не теряют поддержки разработчиков — на уровне исправления ошибок. На FTP-сервере проекта (**ftp://ftp.freebsd.org**) можно обнаружить обновления для версий, начиная по крайней мере с 2.2.

А не так давно на дереве FreeBSD появился еще один интересный побег — PicoBSD (**http://people.freebsd.org/~picobsd/picobsd.html**). То есть — FreeBSD на одной (без преуменьшения, объем дистрибутива — ровно 1 474 560 байт)

дискете. Тем не менее, по уверению разработчиков, она вполне функциональна: имеются варианты сетевой и DialUp-рабочей станции, сетевого роутера и даже сервера своего рода.

Однако завершим историю. На вторую половину 90-х гг. выпадает взрывообразное развитие Интернета. И FreeBSD, изначально на него ориентированная, начинает победное шествие по Web-серверам всего мира. В том числе — и России. Существуют количественно различные оценки относительного веса различных ОС в интернет-службах. Однако качественно они сходятся в одном — преобладании в этой сфере FreeBSD-серверов. Преобладании не только над NT и проприетарными UNIX-системами, но и над столь же открытым и свободным Linux.

А вот о настольных применениях этого сказать нельзя. Если Linux малопомалу пробивает дорогу на пользовательские десктопы, то FreeBSD, похоже, к этому и не стремится. Статистика заходов на сайты, тематически связанные с UNIX и Open Sources, показывает, что доля FreeBSD среди клиентских машин составляет менее полупроцента.

Очень надеюсь, что мой труд по крайней мере поставит вопрос — а не пора ли изменить сложившееся положение? Впрочем, к этой теме я вернусь в следующих разделах. А пока — посмотрим, какие уроки мы можем извлечь из истории, дабы не уподобиться якобинцам из старой песенки студентов-историков Удмуртского государственного университета.

Первый вывод — технологического плана. Хотя FreeBSD по анкетным, так сказать, данным и моложе Linux, за спиной у нее — долгая история совместного с UNIX развития. То есть она — леди с прошлым. Что имеет и свои минусы, и свои плюсы. Не могу отказать себе в удовольствии процитировать одного из разработчиков — Мэтта Диллона (**http://www.FreeBSD.org.ua/articles/vm-design**): "Хотя некоторые считают BSD «старой» операционной системой, те из нас, кто работает над ней, видят ее скорее системой со «зрелым» кодом". Что замечательно согласуется со строками эпиграфа.

Примечание

К слову сказать — не говорите мне, что транскрипция фамилии автора эпиграфа неправильна — она дана по цитируемому изданию (М: Радуга, 1988, с. 31), первому полному в отечественной истории: таким образом переводчики подчеркнули немецкие корни рода Толкиенов (Tolkien).

Однако снова предоставлю слово Мэтту: "Самой большой ошибкой, которую может допустить программист, является игнорирование истории, и это именно та ошибка, которую сделали многие другие современные операционные системы [вернее, разумеется, их разработчики — *А. Ф.*]... Я плохо переношу тех, кто не учит истории".

Готов подписаться под каждым словом обеими руками. Добавив от себя, что относится это не только к программистам и программированию.

Итак, первый вывод из истории FreeBSD — что это система сложившаяся, устоявшаяся, где-то даже консервативная. Но при этом — постоянно развивающаяся и совершенствующаяся. Действительно, взять хотя бы программу ее установки — я не слышал, чтобы она хоть сколько-нибудь радикально изменилась со второй версии (что сплошь и рядом бывает с инсталляционными программами дистрибутивов Linux — иногда до полной неузнаваемости). И в то же время в ней постоянно появляются полезные усовершенствования, облегчающие жизнь. Даже я, при ограниченном сроке общения, заметил их своими глазами.

Кроме того, FreeBSD (и ее предтечи) возникла и развивалась в университетской среде, не просто высококлассными программистами, но людьми с неслабой теоретической подготовкой. Следствием чего явилась исходная продуманность ее архитектуры. И опять помяну Диллана: "В мире программирования алгоритмы становятся более важными, чем код, и именно из-за академических корней в BSD изначально большое внимание уделялось проработке алгоритмов". И это — второй вывод из рассмотренной истории, который также можно отнести к технологии.

Третий же вывод, так же следующий из академического происхождения FreeBSD, носит, условно говоря, гносеологический характер. Ученые (по крайней мере, те, кто заслуживает неругательного значения этого слова) — люди, основным стимулом деятельности которых является удовлетворение собственного любопытства. И FreeBSD, как творение академических исследователей, — это система, идеально для такого удовлетворения подходящая. Как сама по себе (поверьте, изучение FreeBSD — занятие по увлекательности мало с чем сравнимое), так и как инструмент исследования в иных научных областях, в том числе — и далеких от Computer Science...

Последний же вывод из истории имеет отношение к тому, что называют затертым словосочетанием "общечеловеческие ценности". Не знаю, кто как, а я так просто преклоняюсь перед мужеством разработчиков, дважды оказывавшихся у разбитого корыта накануне успеха. И дважды начинавших все если и не сначала, то — с самых, что называется, корней. Каждый, кому приходилось хоть однажды начинать жизнь "с чистого листа", знает, как это сложно. А уж решиться на это второй раз — на такое найдет в себе силы не каждый (за себя, например, не ручаюсь). Так что четвертый исторический вывод: такие ребята просто не могли сделать плохой системы...

Дистрибуция

Как я уже сказал в предыдущем разделе, ОС FreeBSD ныне достигла стадии зрелости. Что выражается, в частности, в ее легкой доступности всему прогрессивному человечеству (а с некоторых пор — и обитателям России). И распространяется она в двух формах — на "твердых" носителях и с FTP-сервера проекта и его официальных зеркал по всему миру.

В качестве "твердых" носителей ныне выступают диски CD-ROM (хотя в скором времени можно ожидать и версии на DVD — по крайней мере, программа установки их поддерживает). Комплекты CD-дисков распространяются, во-первых, непосредственно проектом FreeBSD, во-вторых — упомянутой выше компанией Walnut Creek CD-ROM (**http://www.cdrom.com**). Есть и иные распространители, но их перечисление вряд ли актуально для обитателей России и Ближнего Зарубежья. Где, впрочем, также имеется ряд компаний-распространителей — уж не знаю, таков ли их официальный статус или нет.

Насколько мне известно, текущие CD-версии FreeBSD (следует заметить, что в продаже подчас можно встретить и версии весьма древние) продаются в двух вариантах — однодисковом боксе и боксе на пять дисков. Первый — это собственно инсталяционный (Install) диск, содержащий базовые компоненты системы, некоторый (меняющийся от версии к версии) набор пакетов (packages) и систему X Window.

В полный комплект, кроме этого, входит mini-диск (та же комплектация, что и Install, но без пакетов и X Window), спасательный диск для аварийных ситуаций (Fix-It) и два диска с дополнительными пакетами. Кроме этого, оригинальные боксы, изданные Walnut Creek CD ROM, содержат небольшое печатное руководство (на английском языке).

Цена на дистрибутивы колеблется в зависимости от аппетитов продавца и исполнения (оригинальное от Walnut Creek или тиражирование FTP-версий), но в среднем не существенно превышает базарную стоимость ворованных Windows-наборов (в пересчете на 1 CD).

Однако приобретение дисковых наборов FreeBSD отнюдь не обязательно — система может быть абсолютно бесплатно скачана с FTP-сервера проекта (плата за трафик, которая может многократно превысить стоимость дисков вкупе с их с доставкой — не в счет). Официальный сервер проекта — **ftp://ftp.freebsd.org**, площадка которого расположена "где-то в Америке". И качать с него что-либо объемное из России — занятие не из самых радостных.

Однако он имеет множество официальных зеркал по всему миру, полный список которых приведен на сайте проекта — **http://www.freebsd.org**. Среди них — и четыре российские зеркала, скорость соединения с которыми вполне удовлетворительна. Все официальные зеркала FTP-сервера FreeBSD абсолютно идентичны по структуре и наполнению — насколько я понимаю, это одно из условий получения официального статуса.

Существуют и неофициальные зеркала FTP-сервера проекта FreeBSD, обычно поддерживаемые крупными интернет-провайдерами (только в Москве мне таких известно штук пять). Их отличие — неполное соответствие оригиналу, выраженное в отсутствии старых версий системы и (или) запоздалом появлении последних обновлений.

Формы распространения системы через FTP-сервер проекта FreeBSD существенно разнообразней. Во-первых, там имеются iso-образы CD-дисков, идентичные распространяемым в розничной торговле.

Во-вторых, дистрибутивы могут скачиваться и в виде отдельных архивных файлов — их компоновка практически не отличается от той, что присутствует на CD-диске, купленном или записанном с iso-образа.

В-третьих, на FTP-серверах можно обнаружить коллекции пакетов (packages) — предварительно откомпилированных программ, не входящих в базовый набор системы (но присутствующих в полных CD-версиях).

В-четвертых, FTP-серверы проекта — один из источников исходных текстов для коллекции портов. Подробнее об этом будет сказано в соответствующем разделе. Здесь же замечу, что в коллекции портов обычно дается ссылка на исходный FTP- или HTTP-сервер разработчика конкретной программы и на его зеркала. Однако исходники всех их продублированы и на FTP-сервере проекта FreeBSD.

И, наконец, по FTP доступно и полное дерево исходных текстов всего проекта FreeBSD — именно в нем в первую очередь появляются все обновления системы, осуществляющиеся несколько раз в сутки. Правда начинающим пользователям системы обращаться к нему не рекомендуется...

Регистрация на всех официальных FTP-серверах проекта абсолютно свободна и осуществляется по общим правилам анонимного FTP-доступа, обычно — с указанием FTP в качестве имени пользователя (т. н. логина) и адреса электронной почты вместо пароля. На практике, как говорят, отдельные серверы имеют некоторые ограничения, не позволяющие зайти на них с машины без уже установленной системы FreeBSD, однако мне с такими сталкиваться не приходилось.

Лицензии

Итак, пользователь, желающий приобщиться к системе FreeBSD, имеет множество возможностей для ее получения — на CD за весьма небольшую мзду, или абсолютно бесплатно (т. е. ни копейки не платя разработчикам) через Интернет. В любом случае им оплачивается только физическая стоимость носителя и его доставки (не без учета накладных расходов и прочих бухгалтерских заморочек, вероятно) или сетевой трафик. И речь ни в коем случае не идет об оплате интеллектуальной собственности авторов системы или отдельных ее компонентов.

Возникает резонный вопрос — а не государственная ли это богадельня, подобно 2-му дому Собеса города Старгорода. Нет, не богадельня. И сама система FreeBSD, и все входящие в нее компоненты не принадлежат к классу

т. н. общественной (не путать с общенародной — согласно Владимиру Асмолову, "все для народа, что плохо лежит") собственности — public domain.

Начать с того, что и система FreeBSD как таковая, и все входящее в нее дополнительное программное обеспечение защищены копирайтом их создателей и подпадают под действие законов об авторском праве. О чем имеются ясные и недвусмысленные указания в соответствующих файлах, обычно так и называемых — COPYRIGHT. Другое дело, что список обладателей авторских прав на некоторые программы может занимать не один экран монитора...

Далее, распространение FreeBSD осуществляется на некоторых условиях, не только дающих пользователю права, и немалые, но и налагающего на него определенные обязанности. Для базовых компонентов системы (т. е. FreeBSD par excellence) и то, и другое регламентируется лицензией, которая так и именуется — лицензия BSD.

Предоставляемые ею права включают в себя право на доступ к исходным тестам программы, полученной в рамках этой лицензии, его изучение и модификацию, а также распространение этой программы в оригинальном или модифицированном виде. Основных же обязанностей — две: распространитель обязан сохранять указание на авторство оригинала и не ограничивать права пользователей, получивших этот оригинал (или его модификацию) от него (например, путем злонамеренного сокрытия исходных текстов).

Лучше всего суть BSD-лицензии выражена в сборнике часто задаваемых вопросов (FAQ) проекта FreeBSD: "слово «free» используется здесь [т. е. в названии системы — *А. Ф.*] в двух смыслах, один означает «бесплатно», а другой «вы можете делать всё, что хотите»". За исключением пары вещей, которые вы не можете делать с FreeBSD, например, претендовать на то, что являетесь её разработчиком, на самом деле можно делать с ней всё, что вам заблагорассудится".

То есть (продолжу цитирование) "Она может использоваться безо всяческих выплат, даже для извлечения выгоды." И самое главное — "Все исходные тексты операционной системы свободно доступны, на её использование в других разработках (как коммерческих, так и некоммерческих) и дальнейшее распространение наложены минимальные ограничения".

К слову сказать — главное из этих минимальных ограничений, т. н. "оговорка о рекламе", было не так давно официально снято соответствующим департаментом Университета Беркли, что избавляет меня от необходимости его приводить. Тем более, что я никогда не мог запомнить его не то что дословно, но даже близко к тексту...

Кроме базовых компонентов системы, дистрибутив FreeBSD включает и дополнительное программное обеспечение, которое разрабатывалось разными авторами в рамках проекта GNU, движения Open Sources, под эгидой Free Software Foundation (FSF) и иных соплеменных им по духу. В принципе

участники всех этих проектов, движений и фондов категорически не желают смешивать себя друг с другом, однако это — совершенно другая история (хотя и очень интересная), о которой я говорить здесь не буду. Тем более, что для пользователей их продукции главное, что она, продукция эта, распространяется на условиях т. н. General Public License (все известные мне русские переводы этого интуитивно понятного выражения достаточно корявы или тавтологичны, поэтому далее я буду называть ее просто — GPL), и еще нескольких, именуемых совместимыми.

GPL в целом близка по букве и духу BSD-лицензии, за одним очень важным исключением. Мало того, что весь лицензированный под GPL софт обязан сопровождаться исходными текстами при любой форме распространения, как бесплатной, так и коммерческой. Это требование распространяется и на все программы, разработанные на основе GPL-программ и содержащие их код, — они автоматически подпадают под действие лицензии GPL. Что послужило основанием для появления выражения "GPL-вирус".

Условия же лицензирования BSD не предписывают обязательной открытости исходных текстов для программ, основанных на ней и включающих код какого-либо софта из базового комплекта FreeBSD. То есть разработчик вправе создать на основе FreeBSD собственную систему и распространять свой личный в нее вклад любым образом, в том числе и на коммерческой основе без исходных текстов.

Впрочем, заимствованные из FreeBSD компоненты он закрыть все равно не в праве. Примером чему — неоднократно поминавшаяся мной ранее Mac OS X: ядро ее, т. н. Darwin, основанное на ядре FreeBSD, может скачать любой желающий. Хотя собственно разработки Apple — графической оболочки Aqua в свободном доступе я что-то не видел...

Это различие лицензий BSD и GPL является предметом дискуссий в мире открытого программного обеспечения. Среди приверженцев лицензий GPL и BSD, особенно сохранивших пережитки социализма в сознании (вероятно, по причине того, что ни тем, ни другим при социализме жить не довелось — к вящему их счастью, полагаю), широко обсуждается вопрос, какая из лицензий более соответствует идеям свободы и справедливости.

Апологеты GPL утверждают, что все, основанное на открытом коде, должно оставаться открытым ныне и присно, и во веки веков. Дабы никто не смог воспользоваться плодами чужого труда в целях наживы. Адепты лицензии BSD, напротив, полагают, что такое требование есть нарушение свободы личности, имеющей право сделать результаты своего собственного интеллектуального труда коммерческой (или какой-либо иной) тайной.

Дискуссия своей беспредметностью напоминает споры социал-демократов и анархистов. Закончившуюся, как известно, тем, что пришли большевики, засунувшие и тех, и других на одни нары. И потому данного вопроса я

больше касаться не буду. Тем более, что даже самые строгие ревнители FSF признают совместимость лицензий BSD и GPL. То есть распространяемые на их условиях программы вполне могут быть включены в единый проект. Интересующихся деталями вопроса отсылаю к материалам семинара, проводимого фирмой Altlinux (**http://www.altlinux.ru**), где они обсуждаются профессиональными юристами, не чуждыми, к тому же, миру программ с открытыми исходниками.

К слову сказать, один из компонентов дистрибутива FreeBSD, XFree86 (открытая и свободная ипостась системы X Window), защищается собственной лицензией X-консорциума, в общем и целом сходной с лицензиями BSD и GPL, но отличающейся некоторыми деталями. Что и ей, однако, не мешает быть совместимой с канонической GPL.

Для пользователя же FreeBSD главное то, что любая из лицензий на входящий в ее дистрибутив софт дает ему достаточно прав и свобод. По крайней мере, несколько больше, нежели вложенная в коробку с ОС Microsoft (R) Windows (TM) иже с ними. А уж какая из этих лицензий больше соответствует высшим идеалам свободы — вопрос скорее философский.

Круг пользователей

> Так почему же вас не было на тех тракторных санях и не ваше лицо обжигал морозный февральский ветер, читатель? Где были, чем занимались вы все эти годы? Довольны ли вы собой?
>
> *Олег Куваев "Территория"*

Очень надеюсь, что из предыдущих разделов вводной части читатель понял, что FreeBSD — это система, замечательная технологически, имеющая длинную и интересную историю, и к тому же в высшей мере способствующая развитию свободы и демократии. Если добавить сюда ее практическую бесплатность — пора задаться вопросом: а почему эта система до сих пор не стоит на вашем десктопе?

Для ответа на него остается определить лишь одно: а нужна ли эта система народу? И в первую очередь — лично вам, о многотерпеливый мой читатель? Для чего следует очертить сферу ее возможного применения и наметить круг ее пользователей, существующий и потенциальный.

Сначала — о сфере применения. Об использовании FreeBSD в качестве интернет- и интранет-серверов и серверов локальных сетей сказано уже достаточно, и возвращаться к этой теме я не буду. Обратимся к применениям настольным. Каковые возможны в двух качествах — как специализированных

рабочих станций и как персональных компьютеров общего назначения (в том числе и домашних).

Немного определений. Фантастический рост производительности настольных персоналок привел к размыванию границы между ними и рабочими станциями. К тому же и те, и другие ныне подчас функционируют под управлением ОС одного и того же семейства (сами знаете, какого). Так что единственным отличием между этими классами машин остается почти исключительно их назначение.

Представляется, что к классу рабочих станций можно отнести компьютеры, предназначенные преимущественно для запуска одной программы специального назначения и комплекса обслуживающих ее утилит. Персоналка же по определению призвана решать широкий круг задач, часто с противоречивыми требованиями к аппаратуре.

Перспективы FreeBSD в классе рабочих станций можно оценить по аналогии с Linux. Для последней ОС, как известно, второй сферой приложения, после сетевого в самом широком смысле слова, оказались задачи рендеринга трехмерных изображений, анимации и создания спецэффектов. Что в известной мере обусловлено легкостью адаптации профессионального софта, создававшегося для этих целей исходно на платформе Silicon Graphics.

Однако и FreeBSD для задач такого рода подходит ничуть не меньше: основная программная база у них общая, цена собственно системы — идентична (т. е. стремится к нулю), требования к аппаратуре одинаковы. Что же касается адаптации ранее разработанных приложений — думается, что и здесь нет принципиальных различий. Более того, ввиду особенностей BSD-лицензии, такая адаптация для FreeBSD может показаться более привлекательной в коммерческом смысле.

Теперь — именно об использовании на настольных персоналках. Тут тоже не прослеживается какой-либо разницы с Linux. Все решаемые на ПК задачи можно разделить на три основные группы:

- задачи представления материала;
- задачи "креативного" — творческого характера;
- задачи аналитическо-исследовательского плана.

Типичный пример задач первого круга — т. н. офисные применения. Причем — вне зависимости от степени их сложности, будь то составление докладной записки на полстраницы, презентации бизнес-плана или квартального отчета при многомиллиардных оборотах. Общее между ними то, что они должны доносить до народа некие уже полученные результаты, причем в форме для народа доступной и более-менее общепринятой (по крайней мере, привычной). И потому при прочих равных условиях для решения задач этого круга привлекаются максимально стандартные средства. Какие — думаю, конкретизировать не нужно...

Вторая группа задач — это то, что объединяется понятием Content Creation, адекватного русского перевода которому пока никто не придумал. Здесь и дизайн всякого рода, и высокая полиграфия, и аудио- и видеоприменения. Это, с одной стороны, также представление некоего материала. Однако если в первом случае он преподносится в той или иной a priory заданной форме, то здесь именно форма и является предметом разработки. И оригинальность последней — один из критериев оценки результата. Для задач этого круга критичной становится часто максимальная абстрагированность от деталей технической реализации. И потому естественный выбор для их решения — платформа, изначально разрабатывавшаяся для того, чтобы от этих самых деталей пользователя изолировать — Macintosh, Amiga, BeOS.

Наконец, третий круг задач принято называть обработкой и анализом данных (подчеркну — любых, от финансовых до астрономических). Хотя реально часто это не столько анализ (осуществляемый дедуктивным методом), сколько — синтез, инструмент коего — индукция. Для задач этой категории FreeBSD может оказаться оптимальным выбором. Хотя бы потому, что FreeBSD изначально разрабатывалась исследователями, и в значительной мере — именно для исследовательских целей. А ведь любая "аналитическая" задача содержит в себе элемент исследования.

Таким образом, использование FreeBSD в качестве настольной системы оправдано при преобладании задач обработки данных. Однако она вполне в состоянии выполнять роль универсальной системы, располагая определенными средствами для выполнения как презентационных, так и "креативных" задач. Разумеется, если доля их в общем объеме работ не преобладающая — в противном случае, как я уже сказал, лучше остановиться на иных платформах.

Следовательно, единственным препятствием к широкому применению FreeBSD остается незнакомство с ней ее потенциальных пользователей. Именно для его преодоления и написана эта книга.

Дополнительные источники

Основным источником информации о FreeBSD является официальный сайт проекта — **http://www.freebsd.org**. Он имеет два не вполне официальных русских зеркала — **http://freebsd.org.ru** и **http://freebsd.org/ru/**. Наиболее полные переводы официальной документации проекта — на сайте **http://www.FreeBSD.org.ua/**. Кроме того, большое количество сведений о FreeBSD можно обнаружить на русскоязычных сайтах **http://www.opennet.ru/** и **http://alone.dubna.ru:8080/**.

Наиболее полные руководства по FreeBSD на русском языке следующие:

- Иван Паскаль. Заметки об администрировании FreeBSD — **http://www.tsu.ru/~pascal/**;

- Андрей Федоров. Операционная система FreeBSD — **http://www.anriintern.com/computer/freebsd/** и **http://www.citforum.ru/operating_systems/freebsdc/index.shtml**;
- Андрей Лаврентьев. Заметки по FreeBSD — **http://unix1.jinr.ru/~lavr/**;
- Дмитрий Карпов. Инсталляция FreeBSD в качестве Internet-сервера — **http://www.pi2.ru:8101/prof/techsupp/fbsdinst.htm**.

Сведения об истории FreeBSD, BSD-систем и UNIX вообще можно обнаружить в следующих работах:

- Джордан Хаббард. Краткая история FreeBSD. Руководство FreeBSD (FreeBSD Handbook). Рус. пер. — **http://www.FreeBSD.org.ua/handbook/**;
- Грэг Лиэй. Что такое BSD. Рус. пер. — **http://www.FreeBSD.org.ua/articles/explaining-bsd/**;
- Вадим Колонцов. ОС BSD жила, живет и будет жить 18.03.1997. Открытые системы, 03/1997 (версия online — **http://www.osp.ru/os/1997/03/37.htm**);
- С. Д. Кузнецов. Основание и история. В кн.: Операционная система UNIX. — **http://alone.dubna.ru:8080/history/glava_1.html**;
- История UNIX. Byte/Россия, 2000, #1 (17), 31—33 (автор не указан, вероятно — Алексей Выскубов).

Использованию открытых UNIX-подобных систем (в том числе и FreeBSD) конечными пользователями и их сравнительному анализу посвящен цикл статей Алексея Федорчука на Софтерре Online (**http://www.softerra.ru**) — Демоны, пингвины и пользователи, включающий:

- Какая ОС нужна народу? — **http://www.softerra.ru/review/oses/linux/10125/**;
- Начнем с пингвинов. — **http://www.softerra.ru/review/oses/linux/11118/**;
- Вызываем демонов. — **http://www.softerra.ru/review/oses/linux/11255/**;
- Пингвины против демонов? — **http://www.softerra.ru/review/oses/linux/11397/**.

Наконец, коллекцию изображений демонов BSD можно обнаружить на сайте одного из создателей BSD — Керка МакКасика (**http://www.mckusick.com/beastie/index.html**), который, к слову сказать, обладает авторскими правами на них.

Глава 2

Конспект основных понятий

Если книга эта дочитана до данной главы, значит моя аудитория еще не потеряла интереса к предмету изложения. То есть — готова к знакомству с системой. Однако, прежде чем взяться за это дело, необходимо освоить некоторые базовые понятия, общие для всех UNIX-систем, и, хотя бы на элементарном уровне, их специфические приемы работы. Чему и посвящена данная глава.

Изложение в ней носит конспективный характер, без детализации и объяснений. Все затронутые здесь темы будут более подробно рассмотрены позднее. Однако некоторое представление о них необходимо для установки и первичной настройки системы, без чего дальнейшее знакомство с ней, разумеется, невозможно.

Так что прошу рассматривать эту главу как своего рода краткий словарь. Читатели, имеющие опыт работы с любой UNIX-системой, могут смело пропустить ее почти всю — никаких откровений они здесь не обнаружат. Читателям, не имеющим такого опыта, изучение данной главы также не обязательно, хотя и очень желательно. Однако они могут (и скорее всего без этого не обойдется) обращаться к ней по мере необходимости, встречая в дальнейших главах непонятные термины или поначалу неясные описания приемов работы.

Исключение — раздел про дисковые накопители (который, кстати, не столь конспективен, как прочие): знакомство с ним необходимо уже на первых этапах установки системы. Не вреден он и пользователям других UNIX-систем: номенклатура дисковых накопителей и их разделов в FreeBSD очень своеобразна, сильно отличаясь от принятой, например, в Linux.

Чтобы не повторяться, все сказанное в этой главе относится к установкам свежеинсталлированной системы по умолчанию (если прямо не оговорено иное). Многие (если не большинство) из описанных параметров могут изменяться пользователем по своему усмотрению, что и будет продемонстрировано в свое время.

Основа любой UNIX-системы, и FreeBSD здесь не исключение, — интерфейс командной строки (Command Line Interface, или CLI), реализуемый в

текстовом режиме (именуемом также консольным или терминальным) посредством командных интерпретаторов, или оболочек (shell). Разумеется, возможна и работа в графическом режиме, осуществляемая с помощью системы X Window (оконная система X, именуемая в быту просто "иксами") и надстраивающих ее оконных менеджеров (Window Manager) и (или) интегрированных графических сред (Graphic Environment). Однако приемы работы в них сходны с таковыми в любой графической системе и вряд ли покажутся очень необычными для пользователей Windows. Так что ниже я сконцентрирую внимание именно на консольном режиме.

Представление о системной консоли

Консоль — это первое, что видит пользователь после запуска FreeBSD. Однако чтобы получить возможность выполнять в ней хоть какие-то действия, он должен предварительно зарегистрироваться (или, как еще говорят, авторизоваться) в системе. Для этого от него требуется в ответ на соответствующие приглашения ввести уникальное пользовательское имя (login) и собственный пароль (password), выбранные самостоятельно или данные администратором системы (каковым в случае персоналки выступает наша же скромная персона).

Администратор системы (суперпользователь, superuser, root-оператор или просто root) — это тоже своего рода пользователь, имеющий свой логин (root) и пароль, не подлежащий передаче посторонним лицам. В отличие от обычного пользователя, он обладает практически неограниченными правами на действия в системе. Действия же обычного пользователя подвержены ряду ограничений. Не распространяющихся, впрочем, на его собственные пользовательские данные и настройки — в отношении них его полномочия неограничены (т. е. тут он, подобно Цезарю в деревне, выступает root-оператором локального масштаба).

В UNIX-системах, в целях пущей секретности, при вводе пароля на экране не отображается ничего, даже звездочек. В ответ на ошибку набора логина или пароля выдается сообщение об ошибке авторизации (Login incorrect), без указания на то, где именно она допущена.

С авторизацией связано первое правило безопасности при работе в UNIX, написанное кровью на обломках разрушенных файловых систем: без необходимости не регистрироваться как root-оператор и не вести от его имени никакой повседневной работы, за исключением действий по настройке.

Кроме обычной авторизации, возможна и работа в однопользовательском режиме, когда пользователь получает права root-оператора без ввода его пароля. Режим этот используется исключительно для действий в аварийных ситуациях, когда обычная пользовательская регистрация почему-либо невозможна.

В принципе пользователь может отказаться от использования пароля. Однако делать это не рекомендуется даже на локальной домашней машине, дабы не вырабатывалась скверная привычка. Для машины же, подключенной к локальной или глобальной сети (даже через модем), использование пароля практически обязательно — и это второе правило техники безопасности, выбитое на скрижалях слезами легионов пользователей по случайно утраченным данным.

Авторизовавшись в системе, пользователь наконец получает доступ к системной консоли. Это понятие охватывает сочетание стандартных устройств ввода и вывода. Что, применительно к ПК, означает просто-напросто клавиатуру и экран монитора соответственно. Консоль обладает собственным буфером памяти (т. н. экранным буфером), через который возможен доступ к ранее введенной информации (командам, данным, текстам, системным сообщениям), вышедшей за границы экрана по вертикали. Просмотр экранного буфера осуществляется клавишами <PageUp> (назад) и <PageDown> (вперед) при включенном переключателе <ScrollLock>.

Мышь также некоторым образом входит в состав консоли. Однако она не выполняет здесь функцию указательно-позиционирующего устройства (т. е. не влияет на положение текстового курсора), а служит для копирования и вставки фрагментов экрана. Первое осуществляется выделением экранных блоков — обычным образом, ведением графического курсора мыши при нажатой левой кнопке. Щелчком правой кнопки достигается расширение или сокращение выделенного блока.

Подчеркну, что выделение возможно только в пределах видимой части экрана. Если включить режим пролистывания экранного буфера (или просто воспользоваться клавишами <PageUp> или <PageDown> в какой-либо программе типа текстового редактора), область выделения не расширяется, а просто смещается относительно первоначально выделенного текста.

Любой выделенный с помощью мыши экранный фрагмент (даже размером в один символ, что происходит при фиксации курсора мыши в любой точке экрана) автоматически помещается в собственный буфер памяти. Откуда может быть скопирован в позицию курсора, на этот раз — уже текстового. Делается это щелчком средней кнопки мыши. На двухкнопочных мышах эту функцию может выполнять одновременное нажатие на левую и правую кнопки. Колесо прокрутки скроллирующих мышей при нажатии также работает как средняя кнопка.

Кроме системной (физической) консоли, в FreeBSD существует понятие консоли виртуальной. Это — (почти) самостоятельные терминалы, как если бы они были физически подключены к одной и той же машине. На любом из них можно зарегистрироваться от лица различных пользователей или root-оператора или запустить несколько сеансов работы одного и того же пользователя. Программы (или их экземпляры), запущенные на разных кон-

солях, выполняются совершенно независимо друг от друга. Однако между ними возможен обмен данными — с помощью мыши, как было описано выше. По умолчанию в свежеустановленной системе (в версиях FreeBSD ветки 4.x) существует восемь виртуальных консолей. Переключение между ними осуществляется комбинацией клавиш <Alt> и соответствующей функциональной клавиши (<F1>, <F2> и т. д.). При этом каждая консоль маркируется в верхней части экрана следующим образом:

```
FreeBSD/i386 (free.my.domain) (ttyv#)
```

где # — ее номер, причем отсчет, как это обычно принято в UNIX-системах, начинается с нуля. То есть первая консоль, активизированная при загрузке, индицируется как `ttyv0`, при нажатии комбинации клавиш <Alt>+<F2> мы видим метку `ttyv1`, и т. д.

Представление о командных оболочках

В предыдущем разделе мы рассмотрели основные свойства консоли — связующего звена между аппаратурой машины и пользовательскими программами. Из которых первейшей (как по времени запуска, так и по значению) является командный интерпретатор, или командная оболочка (shell). Это — своего рода аналог COMMAND.COM из MS-DOS. Точнее — наоборот: командный интерпретатор MS-DOS — не более чем недоношенный отпрыск семейства командных интерпретаторов shell из UNIX. И потому последние, в отличие от COMMAND.COM, предоставляют множество дополнительных возможностей.

Кроме того, в UNIX вообще (и в дистрибутиве FreeBSD в частности) существует множество разновидностей командных оболочек, разделяемых на два основных семейства — sh- и csh-совместимые (по синтаксису используемого языка). Подробно об этом будет говориться в *главе 13*. Пока замечу только, что наиболее используемые современные представители обоих семейств — bash и tcsh, — практически идентичны по своим возможностям. И все сказанное ниже относится, за немногими оговоренными исключениями, именно к обеим этим оболочкам.

Правда, по умолчанию в FreeBSD для обычного пользователя устанавливается оболочка /bin/sh, обладающая весьма слабыми интерактивными возможностями. Однако, как будет показано в дальнейшем, изменить среду пользователя труда не составляет.

Основным элементом интерфейса командной оболочки является командная строка, начинающаяся с приглашения для ввода команд (далее обозначается $, хотя вид приглашения может быть настроен в широких пределах).

Как следует из названия, главное назначение командной оболочки — интерпретация и исполнение команд, встроенных в среду (т. е. не имеющих

соответствующего им исполнимого файла) или внешних (исполнимых бинарных файлов или сценариев оболочки). Команда состоит, как правило, из имени встроенной команды (или имени исполняемого файла для внешней команды), опций (параметров, или флагов), не во всех случаях являющихся обязательными, определяющих условия ее выполнения, и аргументов (одного или нескольких), указывающих на цель исполнения команды. Например, команда

```
$cp file1 file2
```

предписывает скопировать (`cp` — имя встроенной в shell команды) файл с именем `file1` (первый аргумент) в файл с именем `file2` (второй аргумент), т. е. создать новый файл с содержанием, идентичным исходному. А команда

```
$cp -R catalog1 catalog2
```

выполняет ту же процедуру для каталогов, причем — рекурсивно (опция `-R`), т. е. в отношении всех входящих в `catalog1` файлов и вложенных подкаталогов.

Команды, опции и аргументы обязательно разделяются пробелами. Опции в большинстве случаев предваряются (без пробела!) символом дефиса '-' (для опции в краткой односимвольной форме) или двойного дефиса '--' (при использовании полного имени опции). В первом случае символы опций могут следовать подряд или каждая — предваряться собственным дефисом (в зависимости от команды и смысла опции). При полной форме опций каждая из них предваряется собственным дефисом. Символ двойного дефиса с последующим пробелом указывает на конец цепочки опций, все последующие символы интерпретируются как аргументы команды.

Впрочем, это правило имеет и исключения. Например, команда `tar` (создание архивов) исторически использовала сплошной список опций в краткой форме, без предваряющих дефисов. Современные же версии `tar` позволяют как применение дефиса перед списком, так и возможность обойтись без него.

Большинство команд в любых UNIX-системах имеет простой мнемонический смысл (для хоть как-то понимающих английский язык): `cp` (от *copy*) — копирование, `mv` (от *movie*) — перемещение, `rm` (от *remove*) — удаление, `mkdir` (от *make directory*) — создание каталога, `cd` (от *change directory*) — смена текущего каталога, `ls` (от *list*) — вывод содержимого каталога.

Здесь необходимо важное предупреждение: большинство команд UNIX может выполняться рекурсивно, т. е. применяться ко всем вложенным подкаталогам и файлам, что требует аккуратного обращения. Так, командой

```
$rm -R /
```

выполненной от лица суперпользователя, можно уничтожить все файлы и каталоги на всех физических дисках системы, поскольку символ слэш '/' означает, как будет сказано чуть ниже, корневой каталог, в который вложе-

ны все прочие. И причем, вследствие особенностей файловой системы FreeBSD уничтожить безвозвратно — аналогов программ UNERASE и UNDELETE из MS-DOS здесь нет в принципе.

В именах файлов, используемых как аргументы команд, допустимы шаблоны (маски), образуемые обычно почти по тем же правилам, что и в MS-DOS: '*' — любая последовательность символов, '?' — любой одиночный символ и т. д. В отличие от MS-DOS, маска '*' может соответствовать именно любому символу (и точке в том числе), и отсутствию символа вообще.

Как уже говорилось, существуют команды, встроенные в оболочку, и команды внешние. Исполнимые файлы внешних команд должны находиться в определенных каталогах, полные пути к которым определены в соответствующих файлах конфигурации (аналогично путям поиска, определяемых в файле AUTOEXEC.BAT в MS-DOS). Для пользователя это обычно каталоги /bin, /usr/bin, /usr/local/bin и usr/X11R6/bin (для администратора добавляются каталоги /sbin, /usr/sbin). Если в ответ на ввод команды следует сообщение об ошибке вида

```
$имя_команды
имя_команды: command not found
```

это не обязательно означает ее отсутствие в системе: возможно, команда эта расположена в каталоге, не охваченном списком путей конфигурационного файла. В этом случае ее следует запустить с указанием полного пути. Следует отметить, что по умолчанию текущий каталог (обозначаемый символом точки) в список путей не входит, что часто служит причиной недоумения начинающих пользователей UNIX. Для запуска команды из текущего (не определенного в PATH) каталога это следует указать в явном виде:

```
$./имя_команды
```

Имена файлов, обычно используемых как аргументы, напротив, ищутся в текущем каталоге. Если же требуется использовать как аргумент файл, находящийся в каком-либо другом каталоге, путь к нему следует указать в явном виде.

Пути могут указываться в абсолютной (относительно корневого каталога) или относительной (относительно текущего каталога) форме. Пример первой формы:

```
$cd /home/username/data1
```

что предписывает перейти в подкаталог data1 в домашнем каталоге пользователя. Относительная форма пути

```
$cd ../data2
```

указывает, что следует перейти из каталога /home/username/data1 в каталог /home/username/data2. Приняты следующие сокращенные обозначения ка-

талогов: ~/, $home или (правильнее в общем случае) $HOME — домашний каталог пользователя, / — корневой каталог файловой системы, . — текущий каталог, .. — каталог, родительский по отношению к текущему (т. е. расположенный на один уровень выше в иерархии каталогов). Сам по себе символ слэш '/' при указании пути всегда соответствует корневому каталогу файловой системы.

Однако командные оболочки bash и tcsh избавляют от необходимости точного запоминания и команд, и путей к файлам благодаря свойству автодополнения. Чтобы воспользоваться им, достаточно набрать одну или несколько первых букв команды или пути к файлу и нажать клавишу <Tab>. Если введенных символов хватает для однозначного определения команды или файла, недостающие будут добавлены автоматически. Если нет — в bash на экран будет выведен список совпадающих с шаблоном вариантов, из которых можно выбрать требуемый. В tcsh того же результата можно добиться комбинацией клавиш <Ctrl>+<D>. А для безвариантного автодополнения, кроме клавиши <Tab>, можно использовать и комбинацию <Ctrl>+<I>.

В обеих оболочках нет и необходимости снова и снова вводить повторяющиеся команды со всеми их опциями и аргументами — можно прибегнуть к истории команд. Просмотр ее осуществляется разными способами:

- клавишами управления курсором <Up> (назад) или <Down> (вперед) вплоть до нахождения нужной, каковая и запускается на исполнение нажатием клавиши <Enter>;
- вызовом команды `history`, выводящей на экран нумерованный список ранее вводившихся команд; для повторного исполнения любой из них достаточно набрать символ '!' и ее номер (без пробела);
- комбинациями клавиш <Ctrl>+<P> (предыдущая команда), <Ctrl>+<N>.

Повторение ранее введенных команд возможно и средствами системной консоли. Для этого следует пролистать экранный буфер, выделить мышью нужную команду и щелчком средней клавиши поместить ее в командную строку.

Есть еще один способ избежать повторного набора команд — сценарии, или скрипты, однако это будет темой специального разговора.

Что еще следует знать о командных оболочках на данном этапе? Пожалуй, об обеспечиваемых ими средствах управления многозадачностью. И для этого также есть несколько возможностей:

- запуск команды в фоновом режиме с добавлением в ее конец, после опций и аргументов, символа '&' (через пробел) — после нажатия клавиши <Enter> сразу, не дожидаясь завершения работы команды, на экране появляется приглашение командной строки и можно вводить следующую команду;

- объединение нескольких команд в одну строку с разделением их символом точки с запятой (;) и пробелом; в этом случае команды исполняются одна за другой в порядке их перечисления; если строка при этом получается слишком длинной и не умещается на экран, ее можно оборвать символом обратного слэша '\' с последующим нажатием клавиши <Enter> — в этом случае исполнения уже введенной команды не происходит, и набор следующей можно продолжать после вторичного приглашения командной строки;
- механизм контроля заданий (jobs Control); текущая задача приостанавливается (т. е. переводится в фоновый режим) комбинацией клавиш <Ctrl>+<Z>, после чего возможен запуск в командной строке другой программы; возврат к приостановленной задаче — `fg #` процесса, который можно узнать, запустив команду `jobs`.

Наконец, последнее: командные оболочки FreeBSD предоставляют два очень мощных средства — возможность перенаправления их ввода/вывода и объединения команд в конвейеры с помощью соответствующих операторов — > и | (оба они выделяются пробелами с обеих сторон). Смысл этих понятий будет объясняться в соответствующей главе. Пока достаточно сказать, что результат выполнения команды может быть записан в виде файла (как говорят, перенаправлен в файл). Например, конструкция

```
$ls > list
```

не выведет на экран список файлов текущего каталога, а запишет его в файл с именем list. А конструкция

```
$ls | less
```

обеспечит постраничный просмотр списка (если размер его выходит за пределы экрана). Перенаправление и конвейеризация могут быть объединены в одной конструкции, например:

```
$ls > list | lpr
```

не только создаст файл, содержащий список текущего каталога, но и выведет его на печать.

Здесь не место для подробного описания команд или их опций. Тем более, что все команды FreeBSD очень хорошо документированы, и получить почти исчерпывающую информацию об их использовании можно с помощью все той же командной оболочки. Вопрос этот столь важен, что его целесообразно вынести в отдельный раздел.

Работа с документацией

Главнейшим средством для получения оперативной информации почти по любому вопросу в FreeBSD являются традиционные во всех UNIX-системах

страницы экранной документации — manual pages или просто man (ласково именуемые в народе "Манями"). Формат их вызова

```
$man имя_команды
```

Ответом на такой запрос будет загрузка соответствующего экранного руководства.

Каждая man-страница начинается с имени команды (или программы) и указания на номер группы страниц, к которым она отнесена (что такое группа страниц, будет сказано чуть ниже). Далее следуют название команды с ее синонимами, если таковые имеются, и краткая справка по использованию. Например, для команды `ls` начало man-страницы выглядит следующим образом:

```
LS(1)      FreeBSD General Commands Manual      LS(1)
NAME
  ls — list directory contents
SYNOPSIS
  ls [-ABCFGHLPRTWabcdfgiklnoqrstu1] [file ...]
```

и т. д. Затем следует более или менее подробное описание использования команды:

```
DESCRIPTION
   For each operand that names a file of a type other than directory, ls
   displays its name as well as any requested, associated information.
   For each operand that names a file of type directory, ls displays the
   names of files contained within that directory, as well as any re-
   quested, associated information.
   If no operands are given, the contents of the current directory are
   displayed.  If more than one operand is given, non-directory operands
   are displayed first; directory and non-directory operands are sorted
   separately and in lexicographical order.
```

и ее опций:

```
The following options are available:
   -A    List all entries except for . and ..  Always set for the super-
         user.
   -B    Force printing of non-graphic characters in file names as \xxx,
         where xxx is the numeric value of the character in octal.
```

и т. д. В заключение, как правило, даются сведения о найденных ошибках (Bug Report) и приведен список man-страниц, тематически связанных с данной (See also), с указанием номера группы, к которой они принадлежат.

Большинство man-страниц занимают более одного экрана. В этом случае просмотр их осуществляется либо построчно (нажатием клавиши <Enter> — только вперед, клавиш <Down> и <Up> — как вперед, так и назад), либо

постранично (клавишей <Spacebar> — вперед, клавишами <PageDown> и <PageUp> — соответственно вперед и назад). Впрочем, условия просмотра определяются программой, установленной данным пользователем в качестве Pager. Для выхода из просмотра man-страницы предусмотрена клавиша <q>. Кроме того, можно использовать и почти универсальную комбинацию для прекращения выполнения программ — <Ctrl>+<C>.

Местоположение man-страниц — в каталоге /usr/share/man; отдельные страницы man могут быть расположены также в каталогах /usr/man, /usr/local/man, /usr/X11R6/man. Именно в этих каталогах, как правило, и происходит их поиск. Узнать, где именно, можно с помощью команды `/bin/manpath`, определяющей пути для поиска по умолчанию, например:

```
$/bin/manpath
/usr/share/man:/usr/local/man:/usr/X11R6/man:/usr/share/perl/man:/usr/loc
al/lib/
perl5/5.00503/man
```

Пути эти определяются при сборке программы man и могут быть скорректированы, о чем будет сказано в главе о командных оболочках.

Man-страницы разделены на девять нумерованных групп, каждая из которых размещена в собственном подкаталоге: /usr/share/man/man1, /usr/share/man/man2 и т. д. Назначение этих групп следующее:

1. Команды и прикладные программы пользователя.
2. Системные вызовы.
3. Библиотечные вызовы.
4. Драйверы устройств.
5. Форматы файлов.
6. Игры.
7. Различные документы.
8. Команды администрирования системы.
9. Сведения для разработчиков ядра.

Наибольший интерес для пользователя представляют разделы 1 и 5, хотя ему нередко придется обращаться и к разделу 8. Разделы 2—4 предназначены в основном для разработчиков программного обеспечения.

Разные man-страницы с одним и тем же именем могут входить в состав разных групп. Так, в группе 1 имеется man-страница tty (1), посвященная одноименной команде, а в группе 4 — man-страница tty (4), описывающая драйвер для устройства /dev/tty. Именно поэтому в первой строке каждой man-страницы указывается ее принадлежность в группе.

Для вызова определенной группы последняя должна быть задана в явном виде:

```
$man 1 tty
```

или

```
$man 4 tty
```

Такая команда вызовет именно ту страницу, которая требуется в данный момент. При вводе команды `man` без номера группы имя соответствующей страницы ищется в первую очередь в группе 1, затем в группе 8, и затем — во всех остальных в порядке возрастания номеров.

Обращение к man-страницам позволяет получить практически исчерпывающую информацию по любым командам FreeBSD, но только в том случае, если пользователь знает название команды, требующейся в данном случае. Однако можно прибегнуть к поиску man-страниц для команды по ключевым словам, отражающим ее функции. Например, команда

```
$man -k print
```

выведет на экран список всех man-страниц, описывающих команды, имеющие отношение к печати:

```
ac                (1)  - print statistics about users' connect time
arch              (1)  - print machine architecture
banner            (6)  - print large banner on printer
```

и т. д. (он может быть очень длинным) с номерами их групп и краткими пояснениями.

Исчерпывающим руководством по использованию системы Manual Pages является ее собственная man-страница. Доступ к ней осуществляется по команде

```
$man man
```

которая выводит на экран man-страницу, содержащую описание этой команды:

```
MAN(1)     FreeBSD General Commands Manual      MAN(1)
NAME
  man - format and display the on-line manual pages
SYNOPSIS
  man [-adfhkotw] [-m system] [-p string] [-M path] [-P pager] [-S list]
      [section] name ...
```

Система man-страниц весьма подробна, но в некоторых случаях может показаться избыточной. Например, если требуется только уточнить формат команды или список возможных ее опций, нет необходимости в подробном описании. В этом случае есть смысл прибегнуть к системе оперативной по-

мощи help, предусмотренной для большинства (хотя и не для всех) команд FreeBSD.

Система help в FreeBSD весьма отличается от одноименной системы в Windows (последняя по замыслу, но не по реализации, сходна скорее с Manual Pages). В сущности, она не является системой, поскольку ее содержание (и даже наличие) целиком зависит от доброй воли разработчиков конкретной команды или программы.

В большинстве случаев оперативная подсказка по данной команде может быть получена следующей конструкцией:

```
$имя_команды --help
```

или

```
$имя_команды -h
```

Часто вывод на экран сообщений help происходит, если команда, требующая обязательного использования параметров и (или) аргументов, введена без них (или параметры и аргументы введены неправильно). Например, для команды `more`, после которой в качестве аргумента не указано имя просматриваемого файла (а именно для этого она и предназначена), ответом будет сообщение:

```
usage: more [-dflpcsu] [+linenum | +/pattern] name1 name2 ...
```

т. е. краткий (но часто достаточный для уточнения мелких деталей) конспект по использованию команды.

В состав FreeBSD входят и иные формы документации, например, принятая в проекте GNU система info. Большинство дополнительных программ, устанавливаемых из портов или пакетов, сопровождается той формой документации, которой ее снабдили разработчики.

Кое-что о текстовых редакторах

Почти любые действия по настройке FreeBSD требуют обращения к текстовому редактору. И потому в данном конспекте следует дать вводные сведения о некоторых из них.

Если читатель уже имеет опыт работы в UNIX-системах, наверняка у него есть любимый текстовый редактор для любых целей (в том числе и конфигурационных). И он, естественно, озаботился его установкой на стадии инсталляции системы. Если же нет — можно воспользоваться штатным средством, предлагаемым базовым комплектом FreeBSD. Это — очень простой в использовании текстовый редактор ее (Easy Editor).

Запускается он одноименной командой

```
$ee file
```

где `file` (необязательный аргумент) — имя файла, подлежащего редактированию. После этого открываются два окна редактора. В нижнем — редактируемый текст (если файла с указанным именем не существует, или редактор ее запущен без аргумента — окно это остается пустым), в верхнем — краткая, но практически исчерпывающая справка по использованию программы.

Навигация по тексту осуществляется или обычным способом — с помощью клавиш управления курсором, <PageUp>, <PageDown>, <Home>, <End> и т. д., либо — специфичными для редактора ее управляющими (или командными) комбинациями клавиши <Ctrl> с какой-либо из литерных. Так, комбинация <Ctrl>+<f> (от *forward*) перемещает курсор на один символ вправо (т. е. вперед), <Ctrl>+<b> (от *bacward*) — на один символ влево (т. е. назад), и т. д. Причем возможности управляющих комбинаций шире — например, они позволяют перейти к следующему слову (<Ctrl>+<z>), перейти в начало (<Ctrl>+<t>) или конец (<Ctrl>+<u>) экрана.

Аналогично и редактирование текста можно выполнять двумя способами: посредством клавиш редактирования (<Backspace>, <Delete>) или такими же управляющими комбинациями. Причем последний способ, как и в случае навигации, — более эффективен, позволяя удалить не только отдельный символ (<Ctrl>+<d>), но и целиком слово (<Ctrl>+<w>) или строку (<Ctrl>+<k>).

На последнее я хотел бы обратить особое внимание пользователей Windows, т. к. управляющие комбинации (иногда реализованные в виде специального командного режима) — отличительная особенность всех нормальных UNIX-редакторов. Поначалу это кажется непривычным, но таким способом достигается много большая скорость обработки текста, чем это возможно в редакторах, основанных на системе меню.

Командные комбинации, во-первых, делают ненужным обращение к специальным клавишам редактирования и управления курсором, что позволяет не отрывать руки от основной части клавиатуры. Тем более, что на многих типах терминалов (а из истории мы помним, что изначально BSD-системы обязаны были работать на весьма разных аппаратных платформах) специальные клавиши далеко не всегда действуют идентичного. В частности, в FreeBSD по умолчанию (при подключении кириллической раскладки) клавиша <Detete> работает как <Backspace>.

Во-вторых, командные комбинации по сравнению со стандартными клавишами управления и редактирования предоставляют множество дополнительных возможностей. В редакторе ее, в силу его примитивности, это не столь бросается в глаза. Но в главе о текстовых редакторах я продемонстрирую это на более представительных примерах.

Однако я отвлекся. Редактор ее не является истинно командным, и основные действия в нем осуществляются через меню, вызываемое нажатием клавиши <Escape>.

Пункты главного меню — следующие:

```
+------------------------+
| a) выйти из редактора  |
| b) подсказка           |
| c) операции с файлами  |
| d) обновить экран      |
| e) параметры           |
| f) поиск               |
| g) разное              |
+------------------------+
```

Смысл всех пунктов вполне понятен, и пользователь легко освоит их эмпирическим методом (тем более, что на данном этапе многого и не требуется). Скажу только, что при выходе из редактора (пункт a) выдается запрос на сохранение изменений в файле, буде таковые производились:

```
+------------------------+
| Файл изменён!          |
|                        |
| a) сохранить и выйти   |
| b) не сохранять        |
|                        |
| для отмены нажмите Esc |
+------------------------+
```

И еще следует заметить, что вывод подсказок и сообщений на русском языке (как в приведенном примере) происходит только при правильной локализации, т. н. установке русской локали (locale), которая у нас пока не выполнена. И, соответственно, при первом запуске программы все ее сообщения будут англоязычными.

Редактор ее — прост и незатейлив и именно поэтому рекомендуется как базовый в FreeBSD. Со своей стороны, для пользователей с опытом работы в DOS и Norton Commander я посоветовал бы обратить внимание на редактор le. Это — чистый меню-ориентированный редактор, идеологически (да и по интерфейсу, вплоть до расцветки по умолчанию) близкий к Norton Editor (NE). Или — к встроенному в Midnight Commander для Linux редактору (порт MC для FreeBSD встроенного редактора не содержит). Описывать его я здесь не буду (на сей предмет в *главе 17* выделен специальный раздел). Скажу только, что пользователь NE не столкнется в нем ни с какими неожиданностями. Единственно, нужно будет не забыть установить его из коллекции пакетов при инсталляции системы (или позднее).

Представление о файловых системах

Следующее, о чем необходимо иметь некоторое представление перед началом знакомства с FreeBSD, — это ее файловая система. Термин этот употребляется в двух значениях: во-первых, как указание на физический способ хранения информации на носителях, и во-вторых — как логическую организацию каталогов и файлов. В первом случае говорят о файловой системе BSD, именуемой UFS (UNIX File System) или, более конкретно, для FreeBSD — FFS (Fast File System). Во втором же — о корневой файловой системе и ее ветвях, которые могут быть весьма многочисленными.

Рассмотрению файловых систем будет посвящена отдельная глава. Пока же достаточно уяснить, что все, что имеется в системе FreeBSD, суть не что иное как файлы. То есть не только исполнимые программы, библиотеки, страницы документации, пользовательские данные, но и терминалы, сетевые карты, дисковые накопители, принтеры и любые другие устройства — не более чем элементы логической файловой системы.

Поскольку файлы FreeBSD — суть разнородные физические объекты, они разделяются на несколько типов. Подробный разговор об этом впереди. Пока же неплохо запомнить (и это потребуется уже в процессе установки), что среди этих типов, кроме обычных файлов (исполнимых бинарных, пользовательских сценариев, текстовых файлов, файлов данных в формате различных программ) и каталогов (списков входящих в их состав файлов), имеются также файлы устройств и т. н. символические ссылки, которые с точки зрения пользователя выглядят как обычные файлы (хотя таковыми не являются).

Все эти файлы организованы в виде иерархического дерева каталогов, начинающегося с корня (каталог / — root, не путать с суперпользователем — root-оператором, и его домашним каталогом /root). И для файлов разных типов и назначения в этой иерархии отведено строго определенное место.

Так, исполнимые файлы общего назначения расположены в каталогах /bin и /sbin, общесистемные конфигурационные файлы — в каталоге /etc, файлы устройств — в каталоге /dev, прикладные программы с их библиотеками, документацией и прочим — в каталоге /usr, пользовательские данные — в каталоге /home (где каждому пользователю выделяется отдельный подкаталог вида /home/username).

Дерево логической файловой системы может включать в себя различные разделы (partition) жесткого диска и даже разные физические накопители (жесткие диски, Zip- и флоппи-носители, CD-ROM и т. д.). Они даже не обязаны находиться на данной локальной машине — в FreeBSD возможен доступ к накопителям машин, объединенных в сеть. При этом физические файловые системы на них — не обязательно FFS, они могут быть различными — FAT любого рода, ext2fs (файловая система Linux), ISO9660

(файловая система большинства дисков CD-ROM) или любыми другими из списка (весьма, кстати, длинного) поддерживаемых ядром FreeBSD.

Обращаю внимание, что в FreeBSD нет никаких буквенных обозначений для накопителей (типа A: или C: в DOS/Windows) — они идентифицируются исключительно по каталогу логической файловой системы, в который включены (как выражаются в UNIX, смонтированы) расположенные на них файлы и подкаталоги. Так, например, содержимое каталога /home вполне может располагаться на втором физическом винчестере. В этом случае говорят, что он (винчестер, вернее, соответствующий ему файл устройства) смонтирован в каталог /home (или имеет точку монтирования /home.

Файловые системы, используемые постоянно, обычно монтируются автоматически при загрузке системы. Сменные же носители (и файловые системы, требуемые лишь эпизодически) должны быть смонтированы вручную. Для этого предназначена команда `mount`, формат которой —

```
mount /dev/имя_устройства /точка_монтирования
```

где точка монтирования — это тот самый каталог, к которому приписывается содержимое монтируемого устройства. Что касается имени устройства — обсуждение этого вопроса, ввиду его важности, вынесено в отдельный раздел.

По отпадении надобности файловые системы сменных носителей должны быть в обязательном порядке размонтированы перед их извлечением командой

```
umount /точка_монтирования
```

где стоит заострить внимание на форме написания команды (именно `umount`) и том, что указывать имя устройства не следует.

Интермедия о дисках

Установка FreeBSD — процесс сложный, требующий ответственного подхода (впрочем, то же можно сказать и о любой другой ОС). Особенно — на машину, уже имеющую какую-либо операционную систему и содержащую наработанные в ней данные. Ведь ошибка в инсталляции на чистую машину грозит лишь потерей времени, тогда как потеря данных, возможно, окажется невосполнимой.

Примечание

Однажды, лет эдак... ах да, в год выпуска Windows 95, я чуть было не угробил все свои данные — результаты пяти лет работы. Это было во времена, когда архив мой умещался на 30 примерно дискетах (ныне под него такого количества CD не хватит), и два его экземпляра казались стопроцентной гарантией надежности. Так вот, после очередного upgrade восстанавливал я его на свежесобранную машину. Естественно, в нетерпении поглядеть, что же за зверь

Pentium 100, я проверил из железа все. Кроме самой мелочи — трехдюймового дисковода. Который и оказался бракованным, при первом же обращении к дискете напрочь сдирая на ней дорожки.

Брак был признан продавцом, дисковод — заменен. Но радости от этого мне было мало — обе дискеты с оглавлением архива не читались ни коим образом, ни на чем. В итоге, конечно, проблема решилась почти без потерь за одну ночь — посредством крепкой кавказской водки кислярки, досадной укушетки для размышлений и призванного в их результате Norton Disk Editor. Однако при воспоминании об этом меня по сию пору чуть не холодный пот прошибает...

Ну, это опять лирическое отступление "Из мемуаров зникейщика". Тем более, что как раз на тему сосуществования FreeBSD с прочими системами в этом разделе я говорить не буду — тому отдельная глава (будет) выделена. Пока же ограничусь трюизмами вроде того, что, прежде-чем-устанавливать-FreeBSD-на-машину-с-DOS-Windows-Linux-etc-необходимо-создать-резервные-копии-всех-данных, "и так далее, и так далее, и так далее".

Тем более, что "никого еще опыт не спасал от беды". Чужой, конечно, — через эти сумерки морали нужно пройти самому. И вот только тогда ты обретаешь способность делить данные на сбэкапленные и несбэкапленные, мерять объем их не гигабайтами, а кубометрами (резервных носителей) и научаешься смелости сначала копировать, а уж потом думать — зачем...

"Лиризм, на все способный, знать, у меня в крови" — снова отвлекся от темы. А должен-то рассказ пойти по преимуществу о дисках. Вернее, о трактовке дисковых накопителей в FreeBSD — без этого не то, что думать о сосуществовании с другими ОС — и саму FreeBSD не установить по уму.

Нужно сказать, что организация и номенклатура дисковых накопителей предельно логична — много логичней, чем в DOS/Windows или даже в Linux. Однако терминология ее столь сильно отличается от них обеих, что осознание этой логики может занять немало времени.

Ниже я буду говорить только об устройствах массовой памяти с интерфейсом IDE/EIDE/ATA и т. д. Устройства с интерфейсом SCSI имеют свою специфику, но я с ними не общался, не знаком и ничего путного сказать не могу.

Как известно, все устройства в UNIX-системах (и FreeBSD здесь не исключение) — не более, чем файлы в соответствующем каталоге (в FreeBSD это — просто /dev, без вложенных подкаталогов). И, соответственно, номенклатура дисковых накопителей — это просто система именеобразования соответствующих файлов.

Так вот, в первой позиции имени файла любого IDE-накопителя, будь то винчестер, CD-ROM, Zip, присутствует буква a — как можно догадаться, от названия ATA-интерфейса. Дальше — все просто и логично: винчестеры именуются *ad*, *лазерные диски — acd*, *Zip-диски — afd* (*от — ATA Disk*, ATAPI Compact Disk, ATA Floppy Disk соответственно). Что, согласитесь, очень легко запомнить.

Далее, файлы IDE-устройства каждого типа обязательно идентифицируются своими номерами. Так, для винчестеров Master-диск на первом IDE-канале имеет номер 0, Slave на нем же — номер 1, и т. д. (как это обычно принято в UNIX, начиная с нуля). Нумерация дисков жестко закреплена за IDE-каналами и состоянием перемычек Master/Slave на накопителях. Она не зависит от количества реально присутствующих дисков. То есть если в системе имеется два винчестера, один — подключенный как ведущий к первому IDE-разъему, и второй — ведомый на втором разъеме, первый получит имя ad0, а второй ad3, тогда как все промежуточные номера останутся незанятыми.

Впрочем, пример мой — сугубо искусственный. В отличие от Windows, в FreeBSD устройство, включенное как ведомое, просто не заработает при отсутствии ведущего устройства. То есть в указанном примере перемычки на втором диске следует выставить в положение Master, и имя его будет, соответственно, ad2.

Несколько замечаний о нумерации устройств на дополнительных IDE-разъемах, типа получивших в последнее время распространение IDE-RAID-контроллеров. Никакой информации на сей счет в литературе я не нашел, потому могу основываться только на собственном опыте. В частности, строки эти пишутся на машине с IDE-RAID Promise, встроенном в материнскую плату Soltek 85SD+. В системе имеются два винчестера, включенные как ведущие на первом канале основного IDE-контроллера и на первом канале IDE-RAID.

Первый из дисков, как и положено, имеет имя ad0. А вот второй — существует как бы в двух лицах — как ad4 и ar0 — вероятно, от ATA RAID. Как легко догадаться, никакого собственно RAID-массива у меня нет и в помине. Причем смонтировать второй диск можно любым способом — указав в качестве аргумента как ad4, так и ar0. Впрочем, это, возможно, особенность Lite-версии BIOS данного RAID-контроллера.

Все сказанное выше (за исключением буквенного наполнения) — не уникальная особенность FreeBSD, аналогичная нумерация дисков существует и в Linux (и, насколько я понимаю, в прочих UNIX-системах). Однако идентификация накопителей этим не ограничивается, за номером устройства следует еще буква s с цифрой и (обычно) буквой. И вот тут-то и начинается своеобразие FreeBSD.

Сочетание s# (где # — цифра от 1 до 4) идентифицирует физический раздел диска (Primary Partition, в понимании DOS/Windows и Linux), которых на PC, как известно, может быть не более четырех. Заострю внимание на том, что физические разделы, вопреки обыкновению, нумеруются, начиная с *единицы*. *А последняя буква (в диапазоне от a до h) — идентификатор раз*дела логического, примерно соответствующего (но — не идентичного!) логическим томам расширенного DOS-раздела в DOS/Windows и Linux.

В терминологии FreeBSD физические разделы именуются слайсами (slice — в данном контексте наиболее адекватным переводом будет "отрезок"), а логические разделы, напротив, партициями (Partition). Такая номенклатура, как я понял, восходит к прошлому BSD-систем (а мы помним из истории, что первые BSD-ленты появились лет за пять до первой IBM PC). Видимо, архитектура машин, на которых работали в Беркли (не PDP ли сколько-нибудь это были?), позволяла иметь на винчестере только один физический раздел, делившийся на логические. Которые и получили обычное название — Partitions. С переносом же на PC FreeBSD проявила законное желание использовать предоставляемую этой архитектурой возможность иметь четыре физических дисковых раздела. Для которых потребовалось придумать новое название, чтобы не путать их с традиционными BSD-партициями. Впрочем, применительно к логическим разделам, говорят и просто о файловых системах, хотя это и не совсем идентичные понятия.

Нужно сказать, что представившуюся на PC возможность FreeBSD использовала на всю катушку, гораздо полнее, чем даже Linux (не говоря уже о DOS/Windows). Она не только понимает все четыре теоретически возможных физических раздела (это может и Linux). Но и позволяет создавать логические разделы в каждом из них, тогда как Linux — только в одном (Extended DOS Partition). DOS/Windows же вообще знает только два раздела — Primary и Extended, и только второй из них умеет делить на логические тома.

Таким образом, каждый доступный для FreeBSD слайс (остановимся на этом термине, коротком и звучном) способен выступать в качестве контейнера для логических разделов (зарезервируем это слово за BSD Partitions, тем более, что оно именно это и означает). Число которых на каждом слайсе можно было бы, казалось, определить, пересчитав буквы от a до h, как восемь. Однако это не совсем так (а обычно — совсем не так).

Дело в том, что, перво-наперво, один из разделов, за которым зарезервирована буква c, имеет неизменный тип файловой системы — unused, т. е. реально использоваться не может. Почему? Да потому, что это — именно тот контейнер, содержащий метаинформацию о логических разделах, аналогично Extended Partition в DOS и Linux.

Далее, литера b зарезервирована за swap-разделом (т. н. разделом подкачки). Каковой, конечно, может быть создан в каждом из возможных слайсов (хотя это и бессмысленно, как будет показано в одной из следующих глав), но для хранения данных использоваться не будет.

И, наконец, предопределение литеры a — маркировать т. н. корневой раздел файловой системы FreeBSD. А сколько бы ни было в машине физических дисков, слайсов на них и логических разделов в слайсах — все они образуют единую (логически) иерархическую, или древовидную, файловую систему. И, как легко догадаться, корень у нее может быть один-единственный.

Исходя из сказанного, легко подсчитать, сколько разделов может быть на единственном диске в системе — 21. Два физических диска могут содержать 41 раздел, три — 61 и т. д. — на один меньше, чем сумма разделов каждого. Такой подсчет имеет целью не только упражнение в арифметике (тоже не лишне — а то скоро на 10 будем на калькуляторе умножать): как будет показано позднее, в ряде случаев целесообразно разделить файловую систему на несколько ветвей, размещенных в изолированных друг от друга разделах. И потому знать, сколько таких разделов может быть всего — отнюдь не вредно.

В результате полное имя файла дискового устройства (которое необходимо указывать, например, при его монтировании или описании в конфигурационном файле /etc/fstab) имеет следующий вид:

```
/dev/ad0s1a  1-й (корневой) раздел 1-го слайса
    Master-диска на 1-м основном IDE-канале
/dev/ad0s1b  swap-раздел там же
/dev/ad1s2d  1-й (понятно, почему?) раздел 2-го слайса
    Slave-диска на 1-м основном IDE-канале
```

и т. д. Все сказанное о слайсах и разделах относилось пока к винчестерам. Однако первое понятие приложимо и к другим накопителям — например, Zip, LS-120. Ведь любой из них содержит минимум один физический раздел, т. е. слайс. Тогда как логические разделы на них вряд ли кто формирует. И, соответственно, Zip (точнее, файл этого устройства) имеет вид /dev/afd0s4. Цифра четыре стоит здесь потому, что хотя Zip-диск "из коробки" форматируется фабричным способом под FAT, размечается на нем именно четвертый физический раздел. Очевидно, что средствами DOS/Windows этого сделать нельзя. И потому Zip-диск, прошедший по какой-либо причине через FDISK, входящий в комплект любой из этих систем, переразметится под первый раздел и приобретет вид /dev/afd0s1.

А вот на CD-ROM, напротив, слайсов не обнаруживается: соответствующий ему файл выглядит как /dev/acd0c.

Излишне говорить, что слайсы могут содержать не только файловую систему FreeBSD. Кстати, называется она UFS (UNIX File System) или FFS (Fast File System): первое название, как легко догадаться — родовое для группы файловых систем ряда UNIX, второе — конкретная реализация берклианских отпрысков. Кроме того, файловая система именно FreeBSD, для отличия от родственных OpenBSD и NetBSD, идентифицируется также своим номером — 165. Наконец, в номенклатуре Linux (с которой, как ни странно, совпадает OpenBSD) ее идентификатор — A5 (в шестнадцатеричном исчислении).

Так вот, любой из четырех слайсов, кроме FFS (чаще всего используется именно это название), может содержать файловую систему FAT16/VFAT/FAT32, Ext2fs, NTFS и т. д. Конечно, разбиение внутри слайса

будет подчиняться законам собственной операционной системы. Более того, FreeBSD вполне уверенно (на чтение и на запись) работает с рядом таких чуждых файловых систем — в частности, FAT любого рода и Ext2fs. Для NTFS возможен только режим чтения. Ну а понимание современных журналируемых систем Linux (в частности, ReiserFS, Ext3fs воспринимается просто как разновидность Ext2fs) со стороны FreeBSD пока не достигнуто.

Осталось рассмотреть вопрос о том, как создаются слайсы, разделы, файловые системы. В FreeBSD процесс этот происходит в три (а не в два, как в Linux или DOS/Windows) этапа. Сначала программой fdisk винчестер режется на слайсы, затем с помощью команды `disklabel` слайсы разбиваются на разделы, и, наконец, команда `newfs` создает на разделе файловую систему (то, что в DOS/Windows называется форматированием).

Однако в ходе начальной установки (да и потом, при добавлении дисков в систему) проще воспользоваться дежурной программой sysinstall, которая в данном случае выступает как оболочка для перечисленных выше программ командной строки. И о которой будет рассказываться в одной из ближайших глав.

Я еще вернусь к более глубокому рассмотрению устройства файловой системы FreeBSD. Приведенных же сведений, думается, достаточно для осуществления той цели, ради которой этот раздел написан — выполнения инсталляции системы.

О правах доступа

Вернемся, однако, к нашему конспекту. Следующий необходимый элемент вводной информации — права доступа к файлам.

Понятия принадлежности файлов и прав доступа к ним принадлежат к ключевым в UNIX. Однако именно они — наиболее непривычны для пользователя ОС линии DOS/Windows. В UNIX принадлежность и права доступа — неотъемлемые атрибуты всех каталогов и файлов, в том числе и файлов устройств. Атрибуты принадлежности определяют для файла его владельца (owner, как правило, это — его создатель), группу пользователей (group) и всех остальных (other). Атрибуты доступа — это право на чтение (read), изменение или запись (write) и исполнение (execute). Последнее право в отношении обычного файла понятно — оно определено для исполнимых файлов, бинарных или текстовых (сценарии оболочки). Для каталога же право на исполнение означает возможность входа в него (например, командой `cd имя_каталога`) и просмотра содержимого (командой `ls имя_каталога`).

Просмотреть права доступа к файлу можно командой

```
$ls -l file
```

Владелец файла может изменить права доступа к нему командой `chmod`. Она требует указания категории принадлежности и, со знаком + (присвоить) или – (отнять) — категории действия. Например, команда

```
$chmod a+r file1
```

дает право на чтение (`r` — read) файла всем (`a` — all) пользователям, а команда

```
$chmod g-w file2
```

отнимает у группы (`g` — group), к которой этот файл приписан, возможность его модификации (`w` — write). Команда вида

```
$chmod u+x file3
```

превращает файл в исполняемый (`x` — execute) для его владельца (`u` — user), что необходимо, например, для созданных пользователем сценариев (скриптов).

Владельцем большинства файлов за пределами домашних каталогов пользователя является администратор. И для всех, кроме него, они обычно доступны только для чтения и исполнения (и то — не все), но не изменения. Однако при необходимости, например, внесения изменений в настройки, пользователь может временно получить права администратора. Для этого служит команда `su` с дальнейшим вводом суперпользовательского пароля. По завершении необходимых действий следует вернуться в обычный пользовательский режим командой `exit`.

Команда `su` позволяет получить права не только администратора, но и любого другого пользователя. Для этого в качестве ее аргумента задается имя пользователя

```
$su username
```

после чего вводится соответствующий пароль.

О графическом режиме работы

И в заключение вводного конспекта — коротко о работе в графическом режиме. В FreeBSD она осуществляется посредством метаинтерфейса системы X Window (оконной системы X), вернее, ее свободно распространяемого варианта — XFree86. Оконная система X не является составной частью FreeBSD, однако она включена в ее базовый комплект.

Запуск сеанса XFree86 можно осуществить различными способами, однако обычно для этого используется команда `startx`. По умолчанию она захватывает 9-ю виртуальную консоль. Консоль, с которой запущен сеанс графического режима, в текстовом режиме становится недоступной. Из графической консоли можно переключиться в любую текстовую комбинацией клавиш

<Alt>+<Ctrl>+<F#>. Возврат в графическую консоль осуществляется с помощью комбинации клавиш <Alt>+<F9>. Выход из графического режима осуществляется, помимо штатных средств оконных менеджеров (о них — см. далее), комбинацией клавиш <Alt>+<Ctrl>+<Backspace>.

Система XFree86 сама по себе не дает пользователю возможности запускать программы, управлять окнами открытых приложений и не предоставляет ему доступа к интерфейсным элементам. Для этого требуется запуск программ, именуемых оконными менеджерами и интегрированными графическими средами.

Оконный менеджер — это программа, обеспечивающая внешний вид элементов графического интерфейса (окна, его заголовка, рамок, меню и т. д.), а также их функциональность (минимизацию/максимизацию окон, сворачивание в пиктограмму, изменение размера и перемещение). Интегрированные среды, кроме этого, предоставляют доступ к наборам специально написанных утилит и приложений и имеют унифицированные средства управления ими.

В состав дистрибутива FreeBSD (хотя и не в базовый комплект) входят три интегрированные графические среды — KDE, GNOME и XFce, и множество оконных менеджеров: WindowMaker, IceWM, Blackbox, FVWM, Enlightenment и др. Пользователь может выбрать любой из них, так же как и командную оболочку в текстовом режиме. Оконные менеджеры и интегрированные среды различаются как функционально, так и внешним видом. Однако с точки зрения приемов работы большинство из них не покажется непривычным пользователю Windows.

Дополнительные источники

Для начального знакомства с ОС FreeBSD (и UNIX) лучше всего обратиться к следующим руководствам:

- Андрей Федоров. Операционная система FreeBSD — **http://www.anriintern.com/computer/freebsd/** и **http://www.citforum.ru/operating_systems/freebsdc/index.shtml**;
- Операционная система UNIX (руководство пользователя) — **http://alone.dubna.ru:8080/faq_guide/newbies/unixuser_null/contents.html**;
- в качестве элементарного введения в предмет можно использовать также главу 3 книги Зубков С. В. Linux. Русские версии. — М.: ДМК Пресс, 2000, содержащую описание базовых понятий UNIX;
- необходимый минимум сведений о работе в графическом режиме приведен в книге: Федорчук А. В. Офис, графика, Web в Linux — СПб.: БХВ-Петербург, 2001.

Глава 3

Подготовка к установке

Вооружившись необходимым минимумом познаний о UNIX-системах вообще и FreeBSD в частности, можно приступить и к установке этой системы. Однако прежде необходимо провести некоторый комплекс подготовительных мероприятий.

Машина для FreeBSD

Первый этап подготовки — позаботиться о том, чтобы имеющаяся у вас в наличии аппаратура удовлетворяла требованиям, предъявляемым FreeBSD.

Обсуждать вопрос об аппаратной конфигурации для работы FreeBSD в деталях вряд ли имеет смысл. Во-первых, эта система отличается тем, что практически без проблем устанавливается на любое более-менее современное оборудование и в дальнейшем успешно на нем работает. Пресловутым минимальным требованиям для современных ее версий — все тот же любой 32-разрядный процессор Intel (или совместимый), 8 Мбайт оперативной памяти и пара сотен мегабайт на жестком диске, — не удовлетворяют нынче разве что музейные экспонаты.

Конечно, работа на минимальной конфигурации радости не доставит — такие машины используются обычно для специальных целей в сетевых системах. Однако с любой иной конфигурацией последних трех-четырех лет проблем не возникнет. Единственно, что FreeBSD, как и все UNIX-системы, любит вдоволь оперативной памяти — комфортный минимум для работы в текстовом режиме я определил бы в 32 Мбайт. Хотя и при нехватке памяти FreeBSD способна работать более эффективно, чем большинство ее сородичей.

Кроме того, для полного изучения возможностей FreeBSD и ее приложений желательно не стеснять ее в отношении дискового пространства. Установка *дополнительного софта из коллекции портов вместе с исходными текстами* (а именно в последних во многом заключается прелесть системы портов) вполне может занять от одного до полутора Гбайт, а то и больше.

Не должно возникнуть проблем и с периферийными устройствами. В FreeBSD поддерживаются все типы мышей (от древних шинных до новейших USB-моделей) и клавиатур (в том числе современных беспроводных компактных, совмещенных с указательным устройством). Распространенные современные модели SCSI-адаптеров, сетевых карт, аудиоустройств для шины PCI распознаются системой обычно безошибочно, поддержка старых ISA-устройств реализована также нормально. Исключение — внутренние PCI-модемы: большинство ныне продающихся моделей относятся к классу Windows-устройств и, скорее всего, в FreeBSD работать не будут. Не следует ожидать и поддержки т. н. GDI- (или Windows-) принтеров.

Еще одна проблема может возникнуть со старыми CD-приводами, имеющими оригинальный (не ATAPI и не SCSI) интерфейс. Они не поддерживаются программой установки FreeBSD, хотя в дальнейшем, после перекомпиляции ядра, вполне в ней работоспособны. Впрочем, думается, что большинство таких приводов уже скончалось естественным образом...

Отдельные требования к оборудованию предъявляет система X Window. Во-первых, использование графического режима удваивает (а в некоторых случаях и учетверяет) потребность в памяти: работа в таких интегрированных графических средах, как KDE и GNOME, не вызывает отвращения только при ОЗУ от 128 Мбайт и более. Во-вторых, естественно, потребуется дополнительное место на диске. И в-третьих, графический режим системы X Window предъявляет претензии к видеосистеме компьютера.

Причем — различные для различных ее версий. Прошли те времена, когда вероятность работоспособности XFree была тем выше, чем древнее был возраст видеокарты. Ныне текущая ее версия (4.x.x) поддерживает все современные достижения в этой области. Однако взамен в версиях 4.1.x снята поддержка многих старых видеокарт, в частности — основанных на популярном некогда чипе S3 Trio. Хотя поддержка такого антиквариата и сохраняется в XFree версии 3.3.6 (а именно она входит в состав базовой системы), однако XFree 3.3.6 ныне выглядит архаичной — в частности, такая обычная для мира Windows или Macintosh вещь, как antialiasing (сглаживание, или устранение контурных неровностей), в ней отсутствует напрочь. Так что при желании пользоваться последними достижениями графического режима системы X Window, следует подумать о не очень древней видеокарте.

Далее — монитор: мало-мальски комфортная работа в X Window начинается с разрешения 1024×768 (а для среды KDE, например, это практически необходимый минимум). Так что требуется монитор, поддерживающий таковое при достаточных эргономических частотных характеристиках — скорее всего, 17-дюймовый. Впрочем, монитор с большой диагональю не будет лишним и при работе в текстовом режиме — консоль FreeBSD позволяет настраивать ее разрешения в очень широких пределах.

Получение дистрибутива

Первое, что необходимо для успешной установки, — это ее предмет, т. е. собственно дистрибутив FreeBSD. Который, как уже говорилось, можно получить либо в виде CD-диска (или их набора), либо образа (образов) таких дисков по FTP, либо, наконец, просто скачать с FTP-сервера в виде отдельных компонентов. Наконец, возможна и просто установка системы по FTP-протоколу, без дистрибутива на локальной машине.

Каждый из источников имеет свои достоинства и недостатки. Установка с фирменных (или таковыми прикидывающихся) CD-дисков — наиболее проста и лучше всего подходит для первоначального ознакомления с системой. Однако в наших условиях (за пределами Москвы, Петербурга и еще некоторых городов, где не существуют системы торговли через Интернет с курьерской доставкой заказа) получение оригинального дистрибутива само по себе может представлять проблему. Не говоря уж о том, что диски стоят денег, хотя и не очень больших.

В этом случае предпочтительным может оказаться скачивание из Интернета образов дисков с дальнейшим изготовлением их посредством любого пишущего привода CD-R/RW и программы для записи iso-имиджей. Разумеется, для этого необходимо также подключение к Сети. И, желательно, оплата его не из собственного кармана: как уже говорилось, полный комплект дистрибутива FreeBSD включает пять дисков, из которых четыре превышают по объему 600 Мбайт каждый. Однако необходимости в полном наборе дисков нет — при наличии подключения к сети (не обязательно постоянного) вполне достаточно одного диска, именуемого инсталляционным (Install), образ которого, в зависимости от версии, составляет 600—650 Мбайт.

Более того, в силу причин, которые я затрону позднее, мне представляется оптимальным скачивать даже не инсталляционный, а т. н. mini-диск, содержащий базовые компоненты системы (хотя и в некотором избытке). Объем его в последних версиях — около 180 Мбайт, т. е. вполне посилен если не для домашнего Dial-Up, то для любой госбюджетной организации с подключением к Сети по выделенной линии.

Хотя ниже я буду говорить в основном именно об установке с CD (безразлично, оригинального или самодельного), способ этот имеет один недостаток: и фабричные диски, и их образы заведомо не отвечают современному состоянию системы. Над текущим релизом (STABLE, не путать с разрабатываемой версией CURRENT) работа не прекращается никогда. Небольшие по объему, но часто существенные изменения и исправления вносятся и в базовые компоненты системы, и, особенно, в коллекции портов и пакетов. Образы же дисков (не говоря о тиражированных наборах), насколько мне удалось установить, законсервированы в этом отношении на уровне момента выпуска текущего релиза.

Это, вместе с избыточностью даже mini-диска, может подвигнуть на установку FreeBSD через Интернет — по протоколу FTP. Благо инсталляционная программа поддерживает не только активный FTP-режим, но и так называемый пассивный (когда локальная сеть подключена к Интернету через брандмауэр — а у здравомыслящих администраторов обычно так и есть). Единственное, что для этого требуется — две чистые дискеты для первичной загрузки FreeBSD (программа установки этой ОС почему-то именно под ней и работает) и их предварительно скачанные образы объемом ровно 1 474 560 байт каждая. Первая из дискет — собственно загрузочная (kern.flp), вторая содержит т. н. корневую файловую систему (mfsroot.flp). Образы их расположены на FTP-сервере проекта **FTP://FTP.freebsd.org/pub/FreeBSD/releases/i386/xx-RELEASE/floppies**.

Образы эти не предназначены для прямого копирования на дискеты, их надлежит перенести на носители с помощью специальных программ. В DOS/Windows для этой цели служит программа RAWRITE.EXE, расположенная по адресу **FTP://FTP.freebsd.org/pub/FreeBSD/tools** на FTP-сервере проекта (содержащем разнообразные DOS-утилиты). Она входит также в состав любого дистрибутива Linux.

Хотя, каюсь, преувеличил (вернее, преуменьшил): для установки по FTP требуется еще и подключение к Интернету. Причем даже не столько уж очень быстрое, сколько устойчивое. Правда, я слышал, что FreeBSD удавалось установить таким образом даже при модемном подключении. Однако мой опыт (на службе, по выделенной линии) оказался неудачным. И даже не столько из-за недостаточной скорости подключения, сколько из-за ее неравномерности. И потому дальше об этом способе установки я буду говорить только вскользь.

Наконец, можно до начала установки скачать с FTP-сервера только жизненно необходимые компоненты системы, разместить их на FAT-разделе жесткого диска и проводить установку с него — и это позволяется инсталляционной программой.

Разумеется, и для установки с FAT-раздела потребуются установочные дискеты — те же самые, что и для установки по FTP-протоколу. Кроме этого, необходимо скачать и еще ряд файлов, расположенных по адресу **FTP://FTP.freebsd.org/pub/FreeBSD/releases/i386/XX-RELEASE**. Можно, конечно, обратиться и к разрабатываемой (CURRENT) версии, но для первого знакомства с системой это далеко не лучший выбор.

Структура вложенных подкаталогов указанного раздела FTP-сервера (**FTP://FTP.freebsd.org/pub/FreeBSD/current/i386/**) повторяет структуру инсталляционного диска. Далее приведен состав интересующих нас каталогов FTP-сервера.

- XF86xxx/ содержит файлы системы X Window (xxx — номер версии XFree86, в текущем релизе FreeBSD — 3.3.6), не являющейся, строго го-

воря, частью системы. В каталоге bin/ — основные компоненты FreeBSD, необходимые для ее функционирования;

- compat1x/ — compat4x/ включают в себя модули совместимости с ранними версиями BSD-систем;
- catpages/, doc/, info/ и manpages/ содержат документацию в различных форматах;
- dict/ — компоненты, необходимые для функционирования словарей;
- proflibs/ — библиотеки специального назначения;
- src/ — исходные тексты для программ базовой комплектации;
- floppies/ и tools/ — я уже говорил;
- games/ ясно из его названия;
- packages/ и ports/ содержат дополнительное программное обеспечение, не являющееся частью системы FreeBSD. Первый — это скомпилированные бинарные пакеты, второй — т. н. коллекция портов, о которой речь пойдет в соответствующей главе.

Следует заметить, что имена ряда перечисленных каталогов можно обнаружить в различных местах дерева каталогов FTP-сервера FreeBSD. Это — не повторения и не различные версии, а просто символические ссылки, выглядящие с точки зрения пользователя, заходящего на FTP-сервер из Windows, как копии оригинала.

Содержимое большинства каталогов — файлы вида *.aa, *.ab, *.ac и так далее (где * — соответствует имени родительского каталога), размером ровно 240 640 байт. На деле это просто архивы tgz (созданные программой архивации tar и сжатые компрессором gzip), разбитые с помощью специальной программы на части. Некоторые каталоги (XFree86/, ports/, packages/) содержат просто один или несколько сжатых архивов tgz. Кроме того, в каждом каталоге имеется минимум два служебных файла — install.sh (скрипт для установки) и CHECKSUM.MD5 (контрольная сумма, обеспечивающая проверку целостности данных).

Не трудно догадаться, что абсолютно обязательными для установки и запуска FreeBSD являются все файлы каталога bin/. Далее, любому пользователю, а особенно начинающему, практически невозможно обойтись без документации, по крайней мере в виде man-страниц (каталог manpages/). Поскольку процедура перекомпиляции ядра в FreeBSD также практически неизбежна, потребуются его исходные тексты (файлы вида ssys.?? из каталога src/). И, наконец, для установки дополнительных программ крайне желательно скачать коллекцию портов. Все это файловое хозяйство в сумме потянет (в текущей версии — 4.6) примерно на 80 Мбайт. Что, согласитесь, несколько меньше, чем даже объем mini-диска...

Без всего прочего на первом этапе можно обойтись (хотя это не значит, что все прочее — бесполезный балласт). Исключение — некоторые файлы из

коллекции пакетов. Такие программы, как любимый текстовый редактор и командная оболочка, окажутся необходимыми уже на первых стадиях настройки. И если машина, на которую предполагается установка FreeBSD, не имеет подключения в Интернету вообще, их следует скачать заблаговременно, отыскав в тематических подкаталогах каталога packages (packages/editors, packages/shells и т. д.).

Полученные из Сети компоненты могут быть размещены на разных носителях — DOS- или UFS-разделе жесткого диска, CD ROM, вплоть до дискет или стриммерной ленты, в том числе и на удаленной машине в локальной сети, с доступом через FTP- или HTTP-протокол или файловую систему NFS. Однако практический интерес для нас в данный момент представляет только размещение на DOS-разделе: для установки с UFS-раздела следует иметь установленную FreeBSD (как и для установки по сети через NFS), а мы только приступаем к этому процессу, CD, изготовленный в Windows из набора файлов, а не из образа диска, не будет правильно опознан программой установки, FTP- или HTTP-доступ в локальной сети оправдан только при необходимости массированной инсталляции на несколько машин. Ну, а о дискетах или стриммерной ленте даже и говорить не хочется...

Дальнейшая подготовка

Дальнейшие шаги на пути к установке зависят, во-первых, от носителя дистрибутива, во-вторых, устанавливается ли FreeBSD как единственная ОС на "чистую" машину или в дополнение к уже существующей операционке (догадайтесь, какой). Если предполагается первый вариант (т. е. "чистая" машина) и имеется дистрибутивный CD — никаких более действий не потребуется, и дальнейшее содержание раздела можно не читать. В любом же ином случае — необходима еще пара-тройка подготовительных действий.

Как уже говорилось, для установки с любого носителя, отличного от дистрибутивного CD, потребуются загрузочные дискеты: программа установки FreeBSD (вот странно-то) запускается под управлением этой же ОС. В принципе, теоретически существует утилита для запуска инсталляционной программы из чистого DOS, но в состав текущих версий она не входит, я с ней не знаком и говорить о ней не буду.

Итак, шаг первый — установочные дискеты. Для их изготовления используется DOS-утилита RAWRITE.EXE. Ее можно запустить из командной строки (желательно в "чистом" DOS, а не в DOS-окне из-под Windows) без параметров и аргументов. В этом случае будет запрошено имя образа дискеты (и полный путь к нему в нотации DOS, т. е. : C:\имя_каталога\имя_файла) и целевой флоппи-дисковод для записи (рискну предположить, что это будет A:\).

А можно задать имя образа диска в качестве опции и целевой дисковод как аргумент:

```
RAWRITE ФАЙЛ_ОБРАЗА ЦЕЛЕВОЙ_ДИСКОВОД
```

Разумеется, верхний регистр необязателен — ведь DOS их не различает, — и приведен здесь только для унификации.

Под Linux загрузочные дискеты изготовляются обычным образом, с помощью команды `dd`:

```
dd of=/path/disk_image if=/dev/fd0
```

возможно — с указанием размера блока (например, `bs=512`). Иными словами — точно так же как изготовляются загрузочные (и resque) дискеты для Linux.

Очень важно при этом, чтобы дискеты были предварительно проверены на предмет испорченных блоков — ни `RAWRITE`, ни `dd` по умолчанию этого не делают, а с дискеты, содержащей испорченные блоки, загрузиться, скорее всего, не удастся. В DOS/Windows такая проверка осуществляется стандартным SCANDISK с включенной опцией "Полная проверка" (или — чем-нибудь вроде утилит командира Нортона), в Linux — командой `badblocks`.

После изготовления дискеты перед нами опять же две возможности: приступить к установке через FTP на "чистую" машину или (в любом ином случае) заняться подготовкой диска. Для установки FreeBSD как единственной ОС с FAT-раздела достаточно создать последний — обычным образом, DOS-командами `FDISK` и `FORMAT`. После чего любым возможным способом (с привода Zip, съемного винчестера и т. д.) записать на него скачанные файлы, необходимые для установки FreeBSD.

Несколько сложнее — в случае, когда предполагается совместное использование FreeBSD с другими операционными системами. Если в качестве второй из них будет выступать Windows, то сначала ее внутренними средствами (теми же `FDISK` и `FORMAT`) нужно создать требуемые для нее разделы (первичный и, при необходимости, расширенный с логическими томами), не забыв оставить достаточное количество неразбитого дискового пространства для FreeBSD. После этого (но обязательно — до установки FreeBSD!) устанавливается Windows — и все, можно приступать к инсталляции FreeBSD.

Необходимость указанной последовательности установки систем (сначала — Windows, затем FreeBSD) определяется тем, что ОС линии DOS/Windows имеют обыкновение изменять содержание главной загрузочной записи жесткого диска (Master Boot Record, MBR), что делает невозможным загрузку любой другой ОС. Конечно, ситуация эта исправима (как будет показано в главе о совместном использовании FreeBSD с другими ОС), однако лучше в

нее не попадать — просто во избежание лишних действий. Замечу, что на системы линии NT/2000 это не распространяется. Более того, загрузчик NT может быть использован и для старта других операционок (в том числе, вероятно, и FreeBSD — сам я этого не проверял за ненадобностью).

При совместном использовании FreeBSD и Linux последовательность установки значения не имеет. И дисковые разделы для них могут быть созданы средствами любой из этих ОС. Однако, если разделы создаются с помощью fdisk или cfdisk из Linux (а также с помощью любой из специализированных программ установки таких дистрибутивов, как Linux Mandrake, Red Hat, Altlinux, ASPLinux и т. д.), следует помнить, что под Linux следует отвести не более трех первичных (primary) разделов диска: поскольку по известным архитектурным ограничениям PC таких разделов в принципе может быть не более четырех, один обязательно должен быть оставлен под слайс FreeBSD. Благо, для файловых систем Linux могут использоваться логические тома Extended-раздела.

Хуже, если Linux уже установлен на машине, занимает весь диск и, к тому же, задействует под свои файловые системы все возможные первичные разделы. В этом случае, если нельзя пожертвовать одним из них, потребуется прибегнуть к какому-либо специализированному дисковому менеджеру. Единственный из таковых, с которым мне довелось сталкиваться, — это Acronis OS Selector (**http://www.acronis.ru**). Он справляется с задачей вполне успешно, позволяя изменить объем Linux-разделов и (или) перенести их (как бы скопировать) из первичного раздела в логический том раздела расширенного. Причем — без потери данных (хотя и не сказать, чтобы очень быстро). А далее, после успешной установки второй ОС, способен выполнять роль мультисистемного загрузчика (собственно, это и есть его основное предназначение).

Правда, Acronis OS Selector не принадлежит к категории свободных (цена программы в коробочном исполнении — около 500 руб.). К тому же для своей работы он требует наличия на диске хоть небольшого FAT-раздела. При отсутствии такового FAT-раздел (по умолчанию — объемом 16 Мбайт) создается собственными средствами Acronis, сокращая доступное для других ОС количество первичных разделов (сиречь слайсов FreeBSD) до трех.

Вероятно, для перераспределения дисковых разделов могут использоваться и такие коммерческие программы, как Partition Magic, однако я с ними дела не имел и ничего сказать по сему поводу не могу. Отсылая читателя к соответствующей документации, имеющейся (вместе с самой программой), например, на дисках, идущих в комплекте с современными материнскими платами Soltek.

И, наконец, последняя ситуация — установка FreeBSD на машину с Windows и диском, полностью занятым разделом (разделами) FAT16/VFAT/FAT32. Тут

уж, если нет желания переразбивать диск и переустанавливать Windows, следует сначала освободить пространство для FreeBSD. В случае единственного (естественно, первичного) FATxx-раздела это делается DOS-утилитой FIPS.EXE, входящей в дистрибутив FreeBSD (в каталоге tools/) или любой дистрибутив Linux. Она позволяет уменьшить размер раздела FAT16 или (в современных своих версиях) FAT32 без потери информации (правда, и без изменения размера кластера).

Предварительно FAT-раздел должен быть дефрагментирован (иначе FIPS просто откажется запускаться) и критически важные данные с него — скопированы: вопреки бытующему мнению, само по себе применение программы FIPS при должной аккуратности никакой опасности для данных не представляет — я использовал ее многократно, и вреда ни малейшего. Однако любые аппаратные сбои (например, питания) в процессе ее работы повлекут за собой необратимое разрушение существующей файловой системы. А поскольку, как сказал бы Козьма Прутков, "не шутите с данными — эти шутки глупы и неприличны", пренебрегать такой вероятностью, сколь бы мала она ни была, не следует.

Далее следует загрузиться в режиме эмуляции DOS, а еще лучше — с дискеты чистого DOS, на которую предварительно скопирована программа FIPS, и запустить ее в командной строке. После этого следует вопрос — отвести ли все свободное место под новый раздел. Чего, разумеется, делать не следует, ведь тогда работа с Windows будет практически невозможна. И потому нужно воспользоваться следующим предложением — изменить соотношения между старым и вновь создаваемым разделами. И, предварительно подумав, такое соотношение задать. Получив в итоге пустое дисковое пространство, не разбитое на разделы.

Недостаток программы FIPS — она не позволяет изменять размер логических томов в расширенном разделе DOS (и к тому же не понимает файловой системы NTFS). Так что буде такая необходимость возникнет — придется обращаться к специализированным дисковым менеджерам типа упомянутого выше Acronis OSS. Последний, кроме всего прочего, позволит в дальнейшем запускать Windows (и почти любые из известных мне ОС) даже со второго физического диска. Или — из расширенного раздела DOS. И еще — обладает способностью к регенерации: после переустановки Windows (согласитесь, не самая редкая процедура при использовании этой ОС) он собственными средствами восстанавливается в загрузочном секторе.

Дополнительные источники

- Главный источник сведений по установке FreeBSD — Иллюстрированное Руководство по Установке FreeBSD Рэнди Пратта, недавно появившееся в русском переводе на **http://unix1.jinr.ru/~lavr/FreeBSDR/**.

- Список поддерживаемого FreeBSD оборудования приведен на официальном сайте проекта — **http://www.freebsd.org**.
- Много полезных сведений по этому вопросу можно обнаружить в сборниках часто задаваемых вопросов (FAQ) и неофициальных вопросах и ответах (QA) по FreeBSD (рус. пер., например — **http://www.FreeBSD.org.ua/faq/** и **http://www.FreeBSD.org.ua/qa/** соответственно).
- Подготовка к установке FreeBSD лучше всего описана на сайте Дмитрия Карпова — **http://www.pi2.ru:8101/prof/techsupp/fbsdinst.htm** и **http://www.pi2.ru:8101/prof/techsupp/fbsdinst/**.
- Описание работы с утилитой FIPS можно найти почти в любом руководстве по установке Linux, как печатном, так и электронном.

Глава 4

Установка

Как я уже говорил в своем лирическом предисловии, уже на стадии установки FreeBSD пользователя, даже с опытом работы, например, в Linux, но не имевшего ранее удовольствия общаться с этой системой, охватывает чувство растерянности. Крепнущее после успешной, как-будто бы, инсталляции. Поскольку просто не знаешь, за что браться в первую голову. Однако, повторив процедуру установки и начальной настройки несколько раз, можно выработать некую последовательность действий, позволяющую свести сложности знакомства с системой к минимуму.

Алгоритм знакомства

Эх яблочко, ну очень хочется,
Гадом буду, подойду — познакомиться.

Тимур Шаов

Именно такой алгоритм (разумеется, не более чем один из многих возможных) и предлагается вашему вниманию в данной главе. Он требует лишь четкого понимания трех аспектов устройства FreeBSD.

Первое, что следует осознать при знакомстве с FreeBSD, — это явно выраженное двухчастное строение ее дистрибутива: система включает базовые компоненты, именуемые Distributions, и разного рода дополнительный софт, реализуемый через систему пакетов (Packages) и коллекцию портов (Ports).

Базовые компоненты включают в себя то, что обеспечивает целостность и функциональность системы как таковой: ядро, общесистемные библиотеки и компиляторы, системные утилиты, минимум основных приложений, а также систему X Window, хотя последняя составной частью FreeBSD не является. Все компоненты Distributions сопровождаются полным исходным кодом.

Приложения, включенные в FreeBSD в качестве Packages, сами по себе также не являются составной частью системы. Подавляющее большинство их — это утилиты и приложения, разрабатываемые в рамках самостоятель-

ных проектов, общие для всех открытых UNIX-подобных систем (в том числе и Linux). Система FreeBSD лишь обеспечивает их корректную установку и функционирование в собственной среде.

Иными словами, Packages — это набор расклассифицированных по функциональному признаку утилит и приложений в бинарном виде, упакованных в стандартные tgz-архивы. Помимо собственно программных компонентов, архивы включают в себя также файлы сценариев, ответственных за установку в системе и регистрацию в соответствующей базе данных. Исходных текстов пакетов в дистрибутиве нет: при необходимости их следует получить самостоятельно (как правило, с сайтов разработчиков).

Кроме того, дополнительные компоненты FreeBSD могут включаться в нее с помощью коллекции портов. Этим термином в данном случае именуются не порты физические (например, последовательные), а результаты переноса (т. н. портирования) программ с одной платформы на другую.

Вообще-то система портов заслуживает отдельного рассмотрения (которое и будет выполнено в соответствующей главе). Пока замечу только, что сами по себе программы из коллекции портов не представлены в дистрибутиве ни в двоичном виде, ни в исходниках (последние, за редким исключением, доступны на FTP-сервере проекта в каталоге **FTP://FTP.freebsd.org/pub/FreeBSD/distfiles**). Система портов только обеспечивает доступ к исходным текстам соответствующих программ, их скачивание, контроль зависимостей, наложение патчей, компиляцию и установку.

Второй заслуживающий осознания факт — это то, что и установка, и постинсталляционное конфигурирование FreeBSD осуществляются одной и той же программой, именуемой sysinstall. Она автоматически запускается при загрузке с инсталляционного CD или с установочных дискет, но может быть вызвана (из каталога /stand) и в любой момент последующей жизнедеятельности. При этом в обоих случаях и интерфейс программы, и выполняемые ей функции абсолютно идентичны. То есть любое действие, доступное через sysinstall, может быть выполнено как при начальной установке, так и позднее, по мере необходимости.

В функции sysinstall входят не только подготовка дисковых разделов, установка базовых компонентов системы и дополнительных пакетов, но и многие дополнительные настройки, в том числе — стартовых сервисов и X Window. Способна она осуществлять и управление пользователями, хотя по описанным ниже причинам в этом качестве ее лучше не использовать. Наконец, в компетенцию sysinstall входит частично установка языковой поддержки: в частности, она в большинстве случаев уже при установке системы позволяет корректно русифицировать и дисплей, и клавиатуру, как для текстового, так и для графического режима.

Третий важный момент вытекает из первых двух, перечисленных выше: поскольку FreeBSD образована двумя отдельными группами компонентов, во-

первых, и т. к. доступ к программе настройки, идентичной инсталляционной, можно получить в любой момент по желанию — во-вторых, нет никакой необходимости стремиться выполнить установку и настройку в один присест. Более того, в полном соответствии с бинарной (в смысле — двухчастной, а не откомпилированной) идеологией FreeBSD, в общем случае это целесообразно выполнить в два этапа. Именно таким образом можно создать идеальную систему "под себя", не обремененную излишними компонентами, но включающую все требуемое для работы и развлечения. Подобно тому, как в рюкзаке хорошего туриста не должно быть ничего лишнего, но все необходимое должно быть...

Однако прежде чем переходить к описанию собственно алгоритма идеальной установки, полезно ознакомиться со всеми функциями программы sysinstall — к ней еще не раз придется обращаться в дальнейшем.

Программа sysinstall — общая характеристика

Как уже говорилось, программа sysinstall запускается автоматически с установочного CD после перезагрузки машины (при включении соответствующей опции в BIOS Setup, разумеется). Правда, предварительно следует предложение сконфигурировать ядро. Шаг, впрочем, отнюдь не обязательный — установки эти имеют силу только в процессе инсталляции. Более того, в источниках рекомендуется вообще отказаться от конфигурирования, если имеются какие-нибудь проблемы с оборудованием. От себя добавлю, что прибегать к прединсталляционному конфигурированию ядра следует только при наличии исчерпывающих данных о таких аппаратных компонентах, как SCSI- и сетевые адаптеры. Иначе лучше положиться на способности FreeBSD к Plug-and-Play — весьма, кстати, неслабые в отношении современных PCI-карт.

Тем не менее, просмотреть возможности настройки полезно, хотя бы в плане знакомства со списком поддерживаемого (программой инсталляции, а не системой как таковой — возможности последней гораздо шире) оборудования.

А вот уж по завершении (или пропуске) конфигурирования запускается собственно sysinstall. Что выражается в появлении на экране главного меню программы (рис. 4.1). Первым пунктом в нем (**Usage**) следует краткая справка по использованию программы. Ничего сверхъестественного в ней не содержится, но по первому разу ознакомиться с ней не вредно. К слову сказать, sysinstall — программа текстового режима с интерфейсом, оформленным псевдографикой. Выбор опций осуществляется в меню исключительно стрелками перемещения курсора, клавишами <Tab>, <Spacebar> и <Enter> (действие последних двух обычно эквивалентно). Не следует ожидать поддержки мыши — ее не будет (даже если включить, как будет показано ниже, соответствующую опцию при конфигурировании).

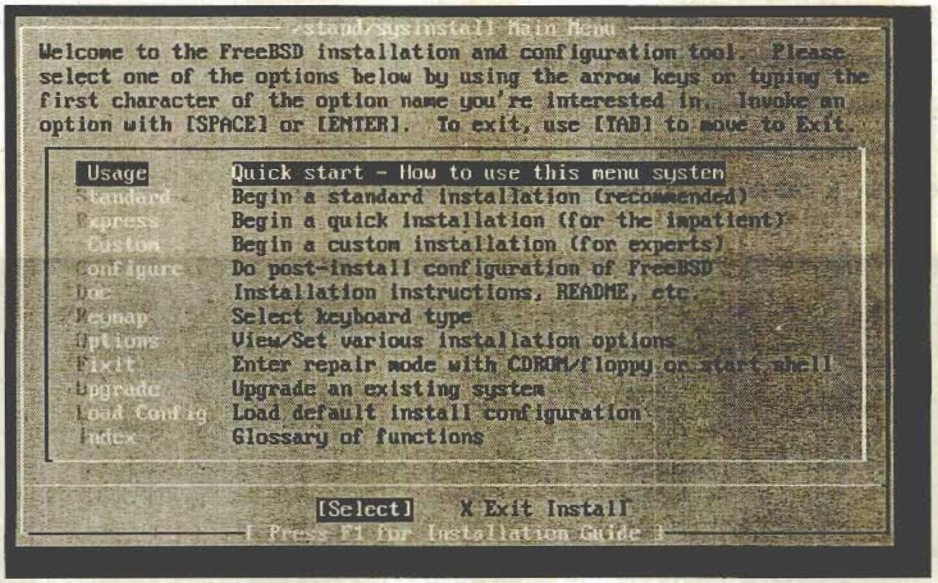


Рис. 4.1. Главное меню программы sysinstall

И, наконец, главное: через пункты **Cancel** (Отмена) или **Exit** (Выход) практически во всех случаях можно вернуться к предыдущей стадии установки. Если же есть подозрение в совершении серьезной, чреватой разрушительными последствиями, ошибки, в запасе всегда имеется выход: нажатием комбинации клавиш <Ctrl>+<C> принудительно оборвать выполнение программы с выходом в командную строку. Откуда sysinstall может быть запущена заново одноименной командой. Правда, все это — до совершения каких-либо необратимых действий вроде создания дисковых разделов. Впрочем, в таких случаях предварительно следуют достаточно внятные предупреждения.

Следующие три пункта **Standard** (Стандартная), **Express** (Быстрая) и **Custom** (Выборочная) — предлагают на выбор три типа установки: стандартный, с некими параметрами по умолчанию, быстрый (для impatient, как сказано в комментарии) и заказной. Первыми двумя я никогда не пользовался (да, честно говоря, даже и не знакомился с ними). И поскольку в итоге нас интересует именно создание идеально сконфигурированной системы "под себя", ниже рассматривается только третий тип установки.

При выборе его вызывается меню выбора опций заказной установки с пунктами:

- **Options** (Опции);
- **Partition** (Создание слайсов);

- **Label** (Создание разделов);
- **Distributions** (Основные компоненты);
- **Media** (Источники установки);
- **Commit** (Подтверждение).

Которые и должно рассмотреть подробнее (рис. 4.2).

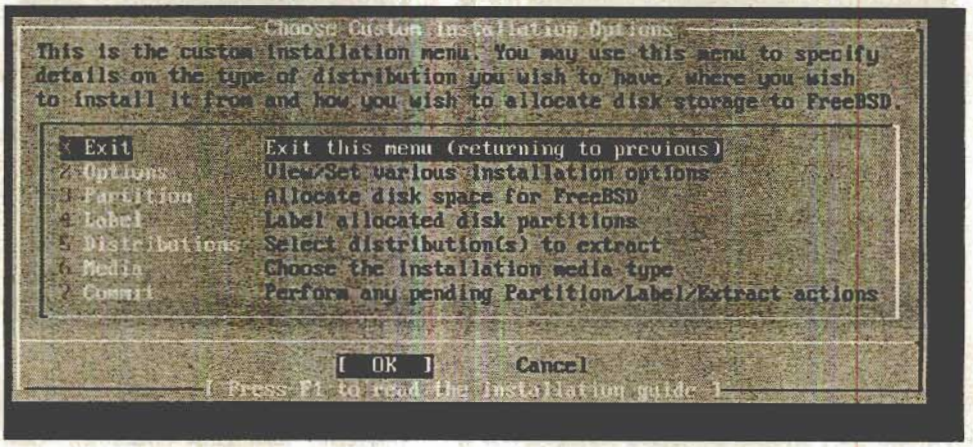


Рис. 4.2. Меню заказной установки

Пункт **Options** (Опции) вызывает редактор опций (рис. 4.3), таких, как сетевые параметры, тип терминала, редактор и браузер по умолчанию и т. д. Редактор и браузер есть смысл определить сразу же, даже если соответствующие программы пока не установлены.

Следующие два пункта **Partition** (Создание слайсов) и **Label** (Создание разделов) посвящены созданию дисковых разделов. Используемая в FreeBSD терминология дисковых накопителей, как уже говорилось, сильно отличается от таковой любых других операционных систем — не только DOS/Windows, но и Linux. К тому же это, пожалуй, единственное действие в процессе всей установки, которое может повлечь особо тяжкие (и необратимые) последствия. Поэтому при необходимости лучше вернуться к соответствующему разделу вводной части и освежить в памяти его содержание. Впрочем, на этом вопросе я остановлюсь подробнее в следующем разделе.

Пункт **Distributions** (Основные компоненты) — центральный в данном меню (рис. 4.4). Это и есть выбор базовых компонентов установки. Все они сгруппированы в некие предопределенные наборы — для разработчиков и пользователей разного профиля, минимально необходимый набор. Есть возможность сделать ручной выбор компонентов или установить весь штатный комплект.

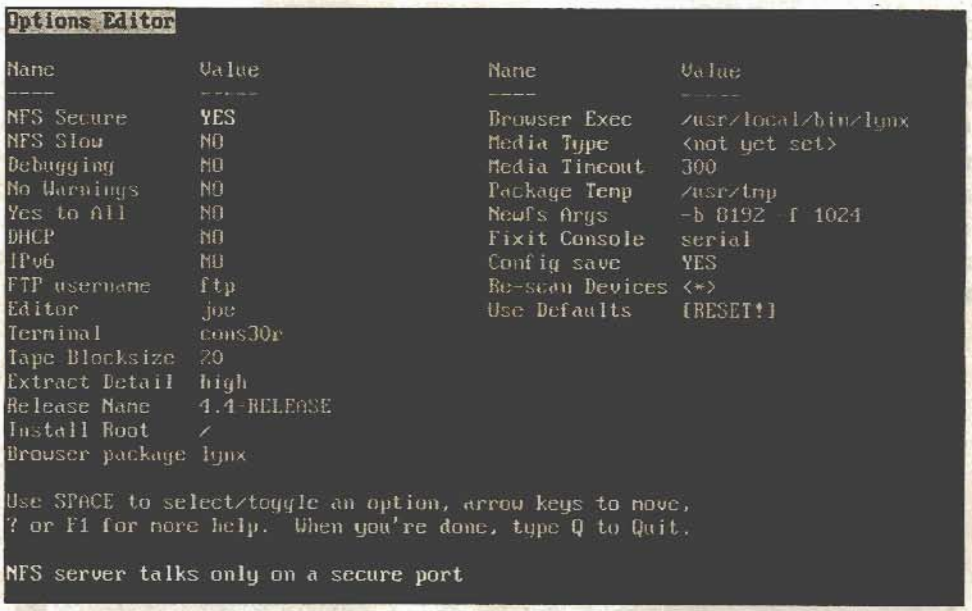


Рис. 4.3. Опции установки FreeBSD

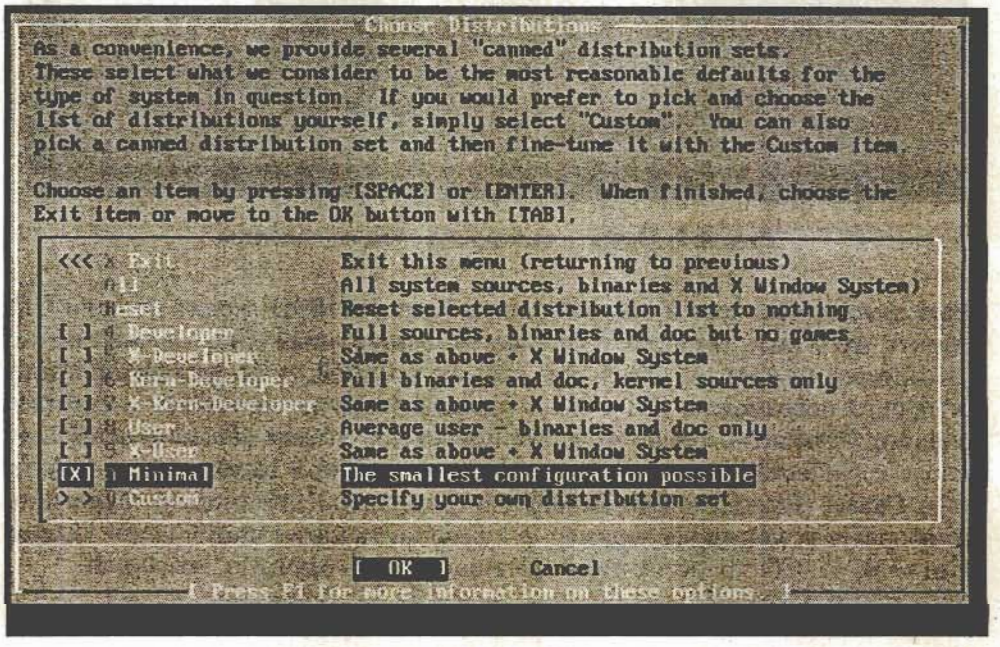


Рис. 4.4. Distributions — выбор базовых компонентов для установки

В пункте **Media** (Источники установки) — выбор источника инсталляции (CD/DVD, FTP в активном или пассивном, через брандмауэр, режиме, DOS-раздел и т. д.) (рис. 4.5). При запуске sysinstall с инсталляционного CD необходимости в выборе первого пункта, разумеется, нет — он включается по умолчанию. Смысл же пункта **Media** (Источники установки) — в возможности смены источника в процессе установки: например, минимально необходимый набор из **Distributions** (Основные компоненты) устанавливается с CD, а остальные необходимые компоненты — с FTP-сервера (где могут оказаться их более свежие версии).

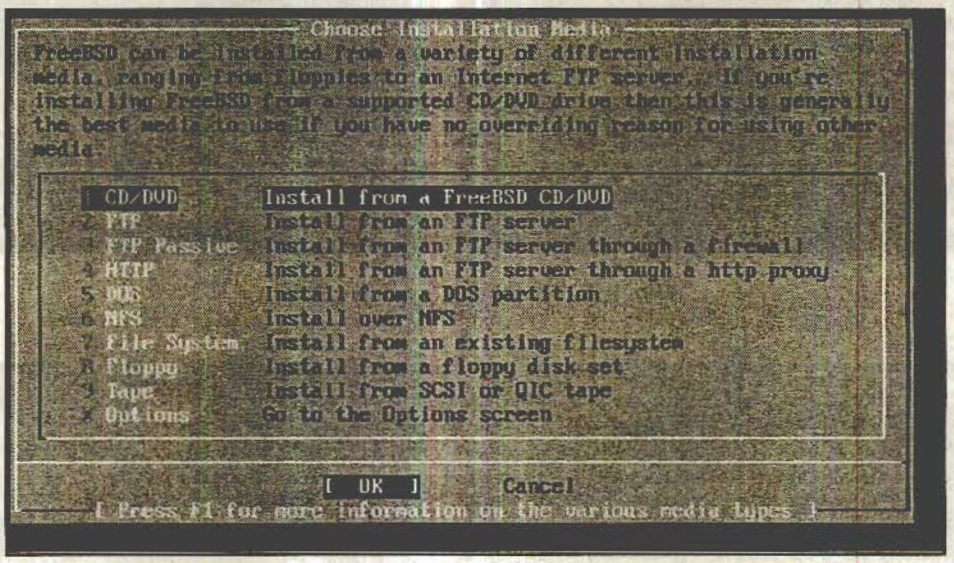


Рис. 4.5. Media — выбор источника инсталляции

Сразу замечу, что, хотя в полный дистрибутив FreeBSD входит не один CD, сменить их в ходе установки не удастся: загрузочный диск автоматически монтируется (в каталог /distr) при запуске sysinstall, и до размонтирования его просто невозможно извлечь из привода (если не прибегать к скрепке, конечно). Размонтирование же диска происходит (также автоматически) при выходе из sysinstall. Причем — только при корректном выходе, через пункты меню **Cancel** (Выход) и **Exit Installation** (Завершение установки). Если просто оборвать установку (например, с помощью нажатия комбинации клавиш <Ctrl>+<C>), CD придется потом размонтировать вручную, командой

```
$umount /distr
```

Обращаю внимание, что в качестве аргумента команды в этом (и только в этом) случае фигурирует каталог /distr: при обычном использовании CD

точкой его монтирования по умолчанию является /cdrom. Впрочем, все сказанное относится уже к запуску sysinstall при установленной FreeBSD — выход из sysinstall при первичной установке завершается автоматическим перезапуском машины.

Наконец, пунктом **Commit** (Подтверждение) завершается процесс базовой установки. Он просто запрашивает подтверждение на выполнение всех заказанных опций и действий. В том числе — и на производство создания дисковых разделов и файловых систем на них. До того как такое подтверждение последует, с дисками физически не происходит ничего, и в любой момент можно прекратить установку, вернув машину к первозданному виду.

Зато по подтверждении предыдущих действий все пути назад отрезаны: диск разбивается на разделы, на них создаются файловые системы, после чего без всякого перерыва происходит установка выбранных компонентов.

Процесс этот, фиксируемый сообщениями на экране, — достаточно длинный, в зависимости от мощности машины и набора компонентов может занять минут 20—30, а то и более. И завершается он выходом в главное меню программы sysinstall.

На этом инсталляция как таковая заканчивается. Можно выйти из программы (через пункт **Exit Installation** (Завершение установки), что автоматически влечет за собой перезагрузку машины (если она была загружена с CD). В процессе которой следует не забыть вынуть CD из привода (сам он оттуда не выпрыгнет) или изменить должным образом установки BIOS. Мы, однако, пока рассмотрим прочие пункты главного меню sysinstall — поскольку пока нашей целью является общее знакомство с этой программой.

Из которых следующим по порядку выступает пункт **Configure** (Настройка) (рис. 4.6). Как можно понять из названия, он посвящен дополнительным настройкам системы, каковые заслуживают подробного рассмотрения. Хотя повторяю, обращаться к ним в процессе первичной инсталляции отнюдь не обязательно — это можно сделать и позднее.

Первый пункт в списке конфигурационных параметров — **Distributions** (Основные компоненты). Содержание его абсолютно идентично одноименному пункту из меню **Custom** (Заказная) — это все те же предопределенные наборы базовых компонентов. Причем обращаю внимание на то, что наборы или отдельные компоненты, выбранные в ходе, скажем, заказной установки, здесь никак не отмечены. Что может создать ложное впечатление ошибочности предыдущих действий. Я, устанавливая FreeBSD впервые в жизни, с испугу повторил здесь весь процесс выбора — и в результате получил еще полчаса повторного переписывания. Это, пожалуй, единственная недоработка программы sysinstall. Тем более странная, что при установке пакетов, как будет сказано абзацем ниже, ничего подобного нет.

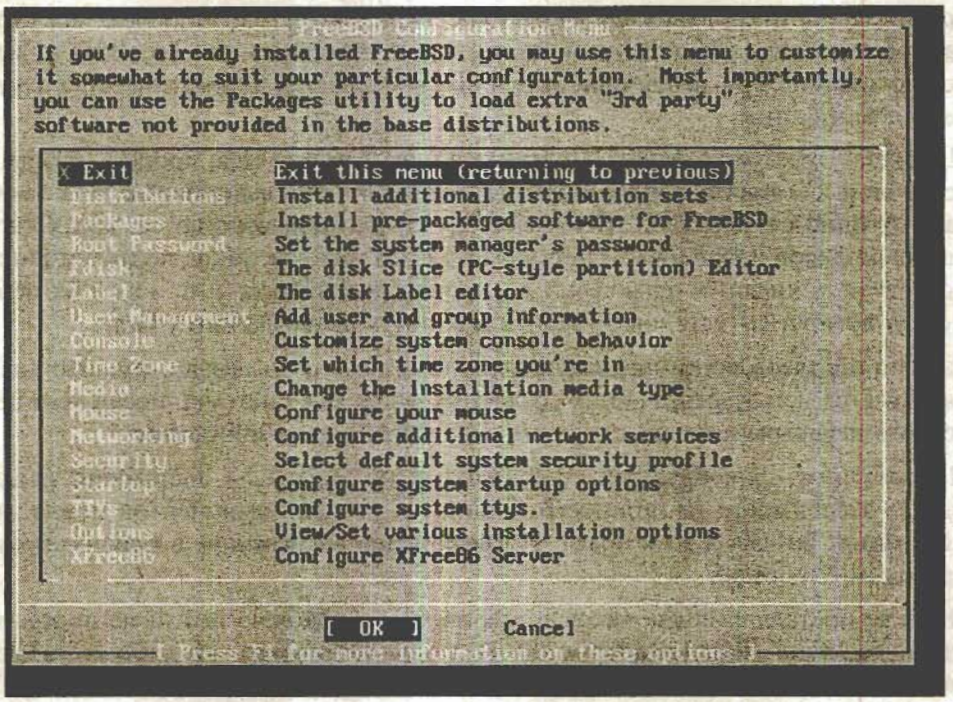


Рис. 4.6. Меню конфигурирования программы sysinstall

Итак, по следующему пункту меню **Configure** (Настройка) — **Packages** (Пакеты) (рис. 4.7). Это — выбор тех самых дополнительных пакетов, о которых я столько говорил раньше. Пакеты объединены в группы по назначению (редакторы, архиваторы, языки программирования и т. д.). Комплектация групп не всегда, на мой взгляд, логична, но вполне понятна.

Однако выбрать всю группу пакетов нельзя (да и не нужно) — это можно сделать только в индивидуальном порядке внутри нее. При этом, вопреки пункту **Distributions** (Основные компоненты), установленные ранее пакеты (например, при начальной инсталляции) оказываются отмеченными. И вторично их устанавливать желания не возникает.

По завершении выбора пакетов начинается их установка (при большом количестве — это также не быстрый процесс), после чего происходит возврат в меню **Configure** (Настройка). Которое следующим пунктом предлагает ввести пароль root-оператора. Что и надлежит сделать, причем — с повторением.

Далее следуют пункты **Fdisk** (Создание слайсов) и **Label** (Создание разделов) — аналоги пунктов **Partition** (Создание слайсов) и **Label** (Создание разделов) из меню **Distributions** (Основные компоненты). Поскольку разбиение диска было выполнено на предыдущем этапе, необходимости обращаться к

ним — нет. Они присутствуют здесь на случай дальнейшего конфигурирования при подключении дополнительного жесткого диска, например.

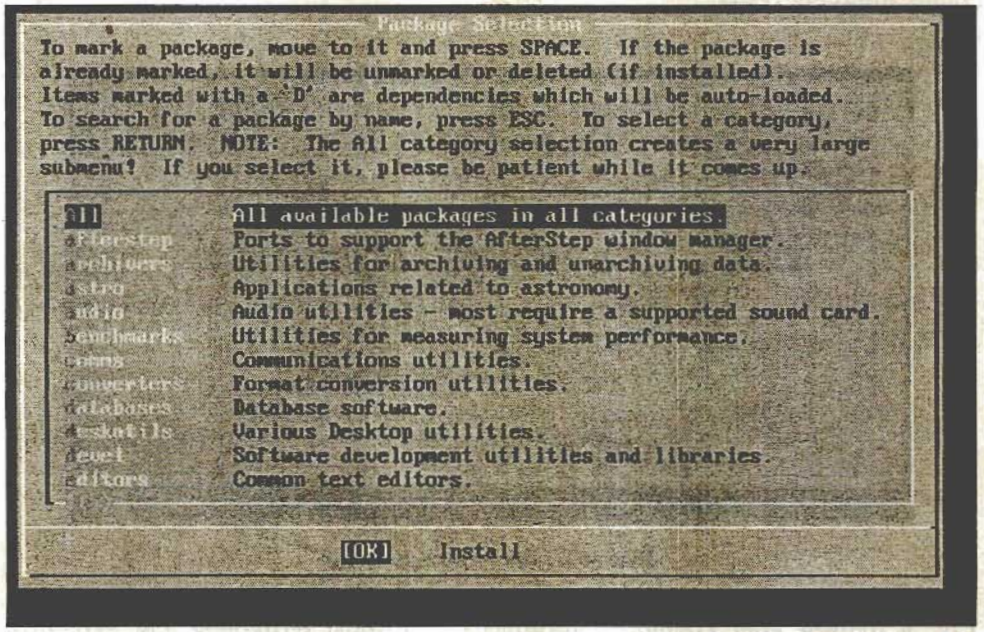


Рис. 4.7. Выбор дополнительных пакетов

Пункт **User Management** (Управление пользователями) предлагает создать учетные записи для пользователей и их групп, именуемые также бюджетами, или аккаунтами (от англ. *account*). Процедура эта заключается в заполнении соответствующих полей двух форм. Для пользователя это — его регистрационное имя (т. н. логин), группа, к которой он принадлежит (по умолчанию — одноименная), идентификатор (генерируемый автоматически), пароль, командная среда и т. д., для группы — название, идентификатор и список пользователей — членов группы.

Если на этом этапе возникнет желание создать пользовательские аккаунты (а в одном из следующих разделов я покажу, почему это делать здесь нецелесообразно), то процедуру следует выполнять в порядке, прямо противоположном предложенному. То есть создать сначала группу для предполагаемого пользователя, а уж затем — его личную учетную запись. Иначе при попытке приписать пользователя к собственной группе последует сообщение об ошибке — ведь ее, группы-то, пока нет.

Следующий пункт, **Console** (Консоль) (рис. 4.8), — серьезный, здесь настраиваются параметры текстового режима, такие как экранные шрифты,

раскладка клавиатуры, т. н. таблица перекодировки. Ну и мелочи вроде реакции клавиш и скринсейверов. А серьезного отношения он заслуживает потому, что позволяет при должном внимании легко провести базовую русификацию системы.

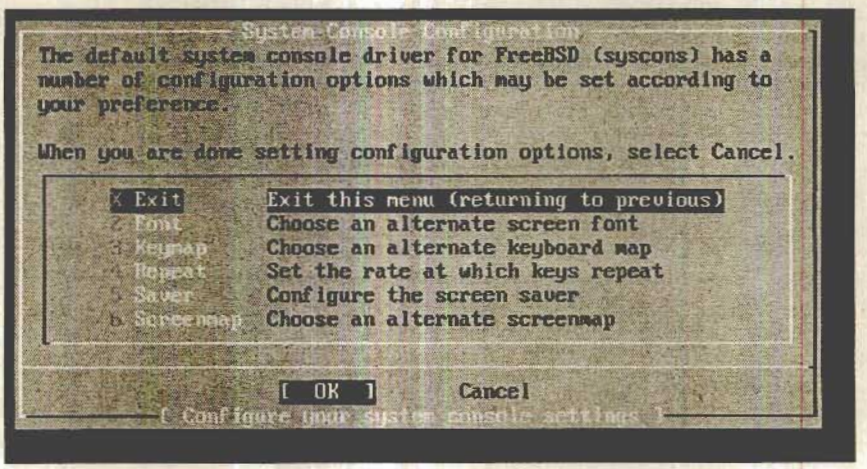


Рис. 4.8. Настройка консоли

Так, в пункте **Font** (Шрифт) выбираются наборы символов для экранных шрифтов (рис. 4.9). Чтобы получить корректный вывод русских букв, следует избрать **IBM 866** — это т. н. альтернативная кодировка кириллицы (известная также как кодировка MS-DOS, или кодовая страница 866). В объяснение причин вдаваться не буду — это тема специального раздела, — пока достаточно это просто запомнить.

Далее, в пункте **Keymap** (Раскладка клавиатуры), устанавливается раскладка клавиатуры (рис. 4.10). Ее можно выбрать из обширного списка, в котором среди всем известных бразильских, хорватских, словенских, венгерских и прочих затесалась и весьма редкая — **Russia KOI8-R**, т. е. русская раскладка в кодировке KOI8.

Этот факт должен несколько насторожить: как же так, кодировка экранных шрифтов, сиречь вывода, — cp866, а кодировка клавиатуры, то бишь ввода, — KOI8. Не получится ли в результате абракадабра из жареных лошадей с кольдкремом? Нет, поскольку посредством карты перекодировки или таблицы соответствия, **Screenmap** (Таблица соответствия), мы приведем их в *соответствие друг с другом: для этого в нем выбирается опция* ***KOI8-R to*** **IBM866** (рис. 4.11). Обоснование чего, опять же, опускаю до лучших времен. Но прошу поверить на слово — все будет почти нормально.

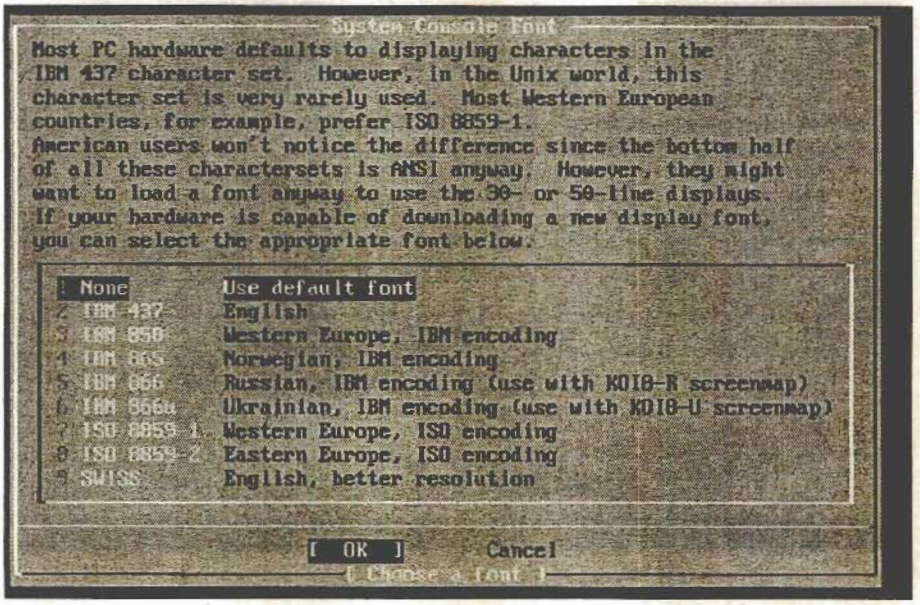


Рис. 4.9. Выбор экранного шрифта

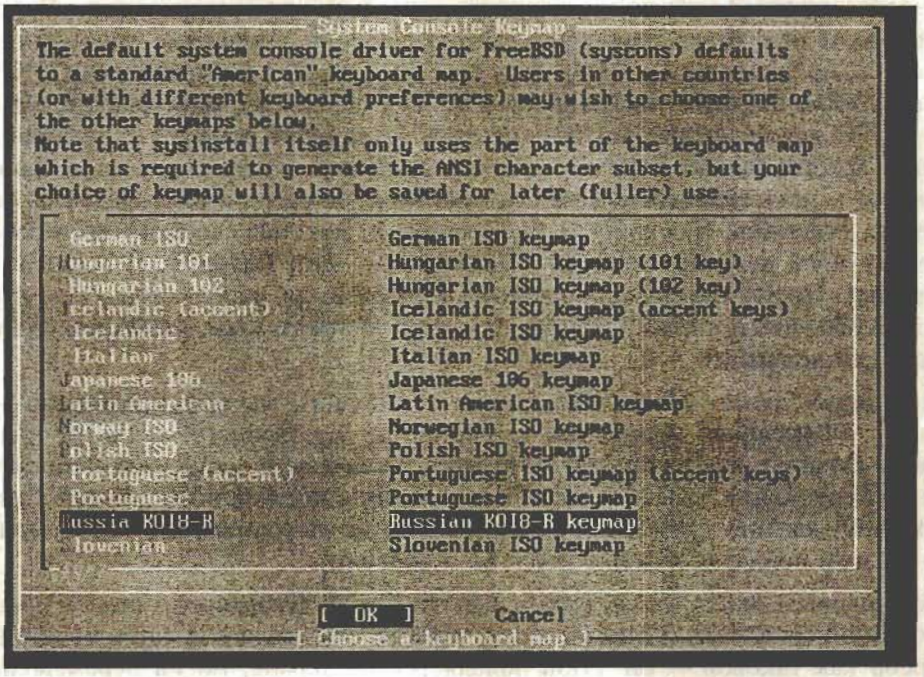


Рис. 4.10. Выбор раскладки клавиатуры

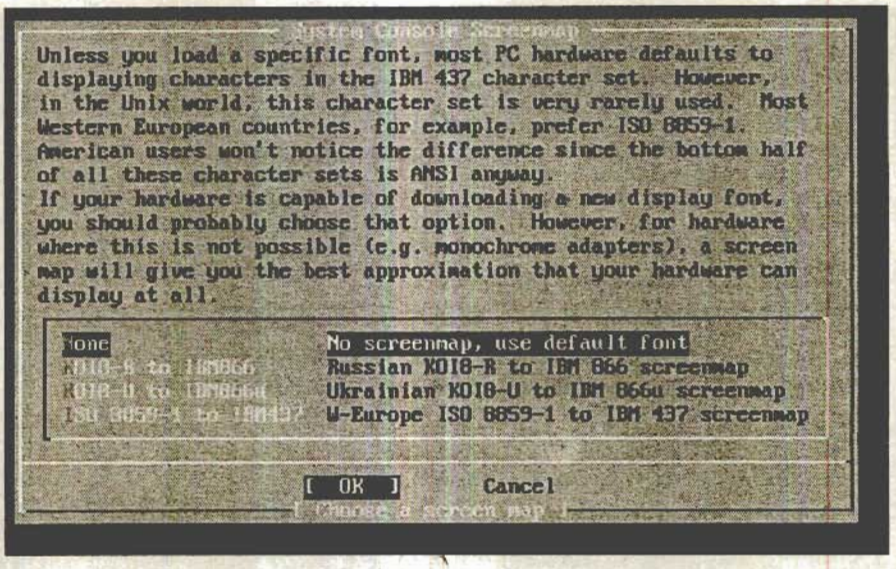


Рис. 4.11. Выбор таблицы соответствия

Вернемся, однако, к меню **Configure** (Настройка). Следующий пункт — установка часового пояса, достаточно тривиален по содержанию. Первонаперво в нем будет спрошено, установлены ли системные часы данной машины на время по Гринвичу или по существу. Для всамделишних UNIX-машин с постоянным подключением к Сети обычен первый вариант. Однако на PC время, скорее всего — местное (этот ответ отмечен и по умолчанию). Так что дальше остается выбрать континент, в нашем случае — **Europe** (Европа — для европейской части, конечно), затем — страну надеюсь, что **Russian Federation** (Российская Федерация) и, наконец, собственно часовой пояс (в моей деревне это будет **Moscow+00 — west Russia**, т. е. время московское). На что будет вопрошаемо — а резонно ли использовать для этого пояса аббревиатуру **MSK**? Почему бы и нет — ответим мы, чтобы покончить с данным вопросом.

Следующий пункт, **Media** (Источники установки), мы уже проходили — в пункте **Distributions** (Основные компоненты) из главного меню. А пункт **Mouse** (Мышь), как не трудно догадаться, посвящается общению с грызунами, именуемыми обычно указательно-позиционирующими устройствами (впрочем, именно в FreeBSD они таковыми не являются ни в коем случае, но об этом пойдет разговор в *главе 12*, о консоли).

Здесь первым делом (посредством пункта **Enable** (Разрешить) делаем мышь доступной (рис. 4.12). Строго говоря, пока не саму мышь, а лишь мышиный курсор как таковой — он лишь появляется на экране, но на перемещения самого устройства никак не реагирует. Однако это, товарищи, правильно —

зато мышь будет работать в консоли после окончания установки и перезагрузки машины (если не прибегнуть далее к пункту **Disable** (Запретить). Затем в пункте **Type** (Тип) устанавливаем протокол (с большой долей вероятности для всех современных мышей подойдет **Auto** (Автоопределение). Далее — порт, т. е. физический интерфейс (COMx, PS/2, специальный шинный). Теоретически можно выставить какой-либо флаг (например, для трансляции вращения колесика скроллирующих мышей), но это — уже детали. Затем курсор мыши можно отключить — в этом случае при перезагрузке машины мышь по умолчанию работать в консольном режиме не будет (даже если предварительно был отмечен пункт **Enable**).

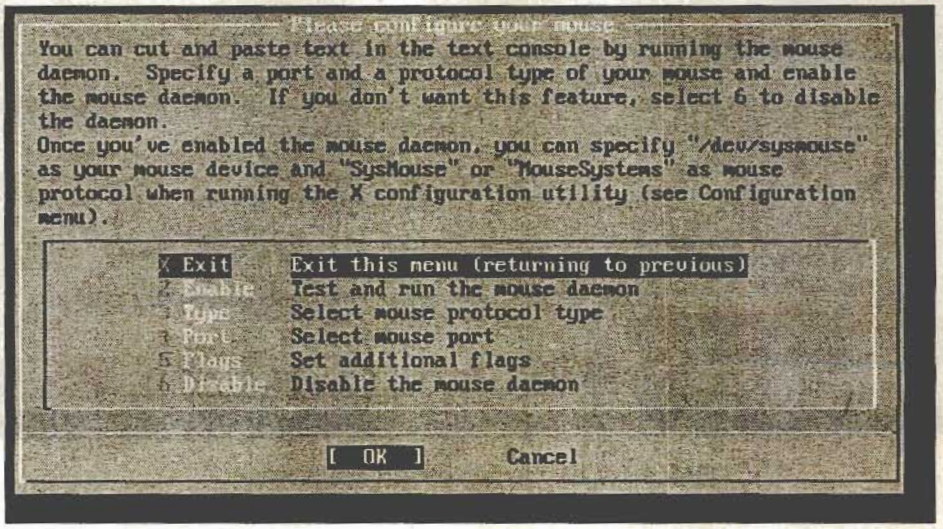


Рис. 4.12. Настройка мыши

Пункт **Networking** (Работа в сети) — это не только настройка сетевых соединений, но также включение всякого рода сетевых служб — анонимного FTP, NFS, роутинга, систем доставки почты и многого другого (рис. 4.13).

Впрочем, есть тут и пункт **Interfaces** (Интерфейс) — это именно настройка доступных физически сетевых соединений. При наличии сетевой карты (и правильном ее определении, что для PCI-устройств более чем вероятно) следует вопрос об использовании DHCP — Dynamic Host Configuration Protocol. Если машина подключена к локальной сети предприятия, скорее всего ответ должен быть положительным, после чего все происходит почти автоматически. *Если нет* — следует обратиться к системному администратору.

Далее, в этом же пункте, **Interfaces** (Интерфейс), можно настроить и нетрадиционные сетевые соединения, через COM- или LPT-порт — возмож-

ность, небезынтересная для конечного пользователя, особенно — в домашних условиях.

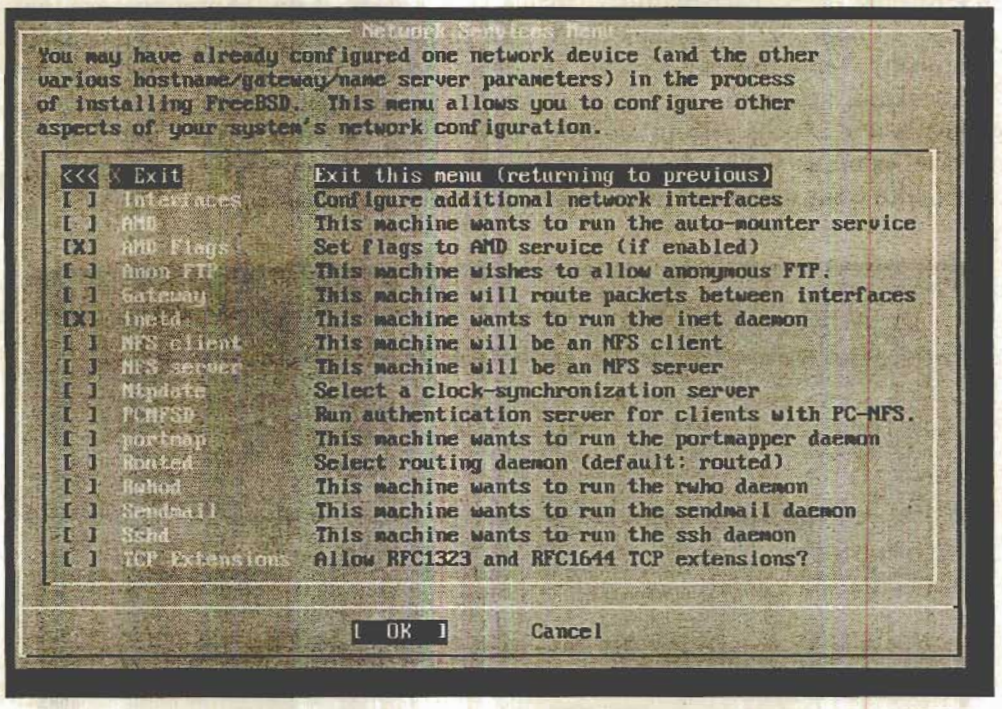


Рис. 4.13. Настройка сетевых соединений

Смысл пункта **Security** (Безопасность) — понятен из названия. Впрочем, уровней безопасности — всего два **Medium** (Средний) и **Extreme** (Экстремальный). Причем если не обращаться к этому пункту вообще, по умолчанию, без участия пользователя, будет установлен первый.

В пункте **Startup** (Начальные установки) включаются/отключаются стартовые сервисы, которых весьма много, и рассматривать которые полагаю неуместным. Замечу только, что именно здесь можно установить двоичную совместимость с Linux (т. е. возможность запуска скомпилированных под Linux приложений), о чем со временем расскажу поподробнее.

Пункт **TTYs** — достижение последних версий FreeBSD (начиная, если не ошибаюсь, с версии 4.4). Что ничуть не уменьшает его полезности — здесь настраиваются параметры виртуальных консолей, о чем речь пойдет в соответствующем разделе, где и поговорим подробнее на эту тему. Пока замечу только, что тут можно внести последние штрихи в русификацию системы, заменив значения `cons25` на `cons25r`. Смысл этого оставим пока в тени, но

поверьте — это очень даже не лишнее, если консоль ранее была нами русифицирована (рис. 4.14).

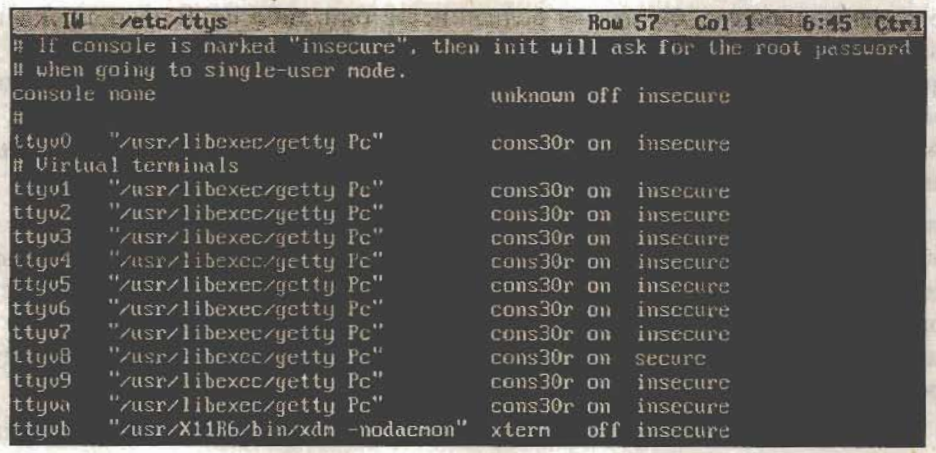


Рис. 4.14. Настройка виртуальных консолей

В **Options** (Опции) — опять же перепевы былого одноименного пункта из меню **Distributions** (Основные компоненты). После чего в нашей программе конфигурирования остается лишь настройка системы X Window. Разумеется, это деяние имеет смысл только в том случае, если при разборке с **Distributions** (Основные компоненты) из базового комплекта таковая была установлена. Что — отнюдь не обязательно: в базовом комплекте представлены "иксы" версии 3.3.6 изрядного (я бы сказал, далеко забальзаковского) возраста. И пользоваться ими имеет смысл только в том случае, если видеокарта системы не поддерживается текущими (4.x.x) версиями. Каковые имеют место быть только среди **Packages** (Пакетов).

Конечно, ничто не мешает заказать установку системы X Window и из пакетов. Однако настроить их штатными средствами sysinstall, скорее всего, не удастся. А если и удастся — все равно потребуется ручная доводка, так что лучше, наверное, пока в это дело не лезть. Все равно к настройке XFree 4-й версии я еще вернусь *(в главе 5)*.

Что же касается третьей версии — они конфигурируются штатными XF86Setup (в графическом режиме) или xf86config (в режиме текстовом). И тот, и другой способ многократно описывались в литературе, и я на них задерживаться не буду. Тем более, что "иксы" — строго говоря, не FreeBSD (да и возвращаться к этому вопросу еще придется). Замечу только, что после успешной (надеюсь) их настройки остается выбрать оконный менеджер по умолчанию. Разумеется, если хоть какой из списка был установлен из

Packages — в базовом комплекте имеет место быть только безальтернативный FVWM, так что выбирать не из чего.

С пунктом **Configure** (Настройка) покончено. По оставшимся пунктам главного меню пройдусь бегло — в большинстве они либо повторяют пройденные этапы большого пути (как **Keymap** (Раскладка клавитатуры) и **Options** (Опции) — надоело повторять, перепевами каких предыдущих пунктов они являются), либо совершенно тривиальны (**Doc** (Документация) и **Load Config** (Загрузка конфигурации) — т. е. чтение документации и считывание "умолчальной" конфигурации соответственно).

Два оставшихся пункта — **Fixit** (Исправление) и **Upgrade** (Обновление) — собственно к инсталляции отношения не имеют. Первый — это аварийный вход в систему, если она по каким-то причинам не желает грузиться с жесткого диска. В этом случае можно загрузиться с инсталляционного CD и в главном меню sysinstall выбрать пункт **Fixit** (Исправление). После чего будут предложены три варианта спасения утопающих:

- использование второго CD из дистрибутивного комплекта (он так и называется — Fixit, правда, не на обложке, а внутри);
- использование fixit-дискеты (о которой я вскользь упоминал ранее);
- выход в среду, именуемую Holographic Shell, что я применительно к случаю трактовал бы как "голый Shell" — вполне соответствует.

Первые два варианта поддерживают т. н. Live Filesystem, в которой можно продемонстрировать любые другие файловые системы, например, раздел жесткого диска. Хотя и третий вариант допускает кое-какие спасательные манипуляции. Впрочем, все это — тема инструкции по технике безопасности, размещенной в соответствующем разделе.

А смысл пункта **Upgrade** (Обновление) интуитивно понятен. Теоретически в нем можно переустановить базовые компоненты системы при смене версии. На практике, однако, как говорят знающие люди, процедуру эту лучше выполнять вручную.

Марафон по программе sysinstall подошел к концу, и можно подвести итоги. На первый взгляд организация ее кажется бессистемной: одни и те же опции дублируются в разных пунктах главного меню. Однако "ежли в корень посмотреть", во всем этом есть великая сермяжная правда.

Как я неоднократно подчеркивал, sysinstall — не только средство установки системы, но и практически универсальный инструмент для ее конфигурирования. В процессе которого можно поправить очень многие огрехи первичной инсталляции, буде таковые имели место. И потому представляется резонным, что к одним и тем же группам опций можно получить доступ на самых разных стадиях установки и конфигурирования.

Конечно, на мой взгляд, такими дублирующими функциями sysinstall несколько перегружен. Но зато в нем достигнут практически идеальный баланс между средствами ручного и, так сказать, полуавтоматического конфигурирования. Именно полуавтоматического — ни одно действие в sysinstall не происходит без участия пользователя. И если он не всегда может повернуть процесс в требуемом направлении — а это касается и русификации, и управления пользователями, и еще пары-тройки аспектов, — то зато он всегда может отказаться от нежелательного выбора в пользу позднейшей ручной доводки. А, на мой взгляд, ручное конфигурирование "с нуля" — зло много меньшее, чем разбирательство с неправильными настройками "в автомате".

В общем, это — эмоции: осознать мощь sysinstall можно только в процессе ее реального применения. Для чего в следующем разделе я продемонстрирую попытку достижения идеала в конфигурировании системы (для моих личных целей, потребностей и возможностей, разумеется).

Алгоритм идеальной установки

В этом разделе я, наконец, изложу свои представления об идеальной установке FreeBSD. С оговоркой, что система инсталлируется на новую (в смысле — не содержащую другую операционную систему) и типичную современную машину (класса Celeron/P-III/Duron/Athlon/P-4) для настольного применения, без всяких экзотических устройств и SCSI-адаптеров. Установка осуществляется с дистрибутивного CD.

К вопросу о CD — для идеальной установки более чем достаточно одного инсталляционного (Install) диска. А можно обойтись и CD с базовыми компонентами, т. н. mini-диска. От инсталляционного он отличается только включением XFree86 (версии 3.3.6) в базовый комплект и наличием коллекции пакетов. Однако устанавливать заведомо устаревшую версию системы X Window я полагаю нецелесообразным, а весь прикладной софт лучше устанавливать из коллекции портов. Те же несколько приложений, которые потребуются сразу же после инсталляции, для начальной настройки (такие как подходящая командная оболочка и любимый текстовый редактор), можно скачать в индивидуальном порядке (и даже — заранее, не обязательно из FreeBSD).

Для достижения поставленной цели есть смысл, выбрав заказной вариант установки, на первом этапе ограничиться абсолютно необходимыми для запуска и функционирования системы мероприятиями, а именно:

- разметкой диска;
- установкой базовых компонентов;
- установкой абсолютно необходимых пакетов (packages), буде таковые потребуются;
- самыми насущными из настроек (в числе коих — русификация консоли).

Как и в любой другой системе, первой стадией начального этапа установки FreeBSD является подготовка диска, т. е. создание слайсов, разделов и файловых систем для них. О технике этого дела выше (во вводном конспекте) говорилось достаточно подробно. Поэтому, выбрав диск для разметки (если есть из чего выбирать, рис. 4.15) уделим внимание стратегии разметки.

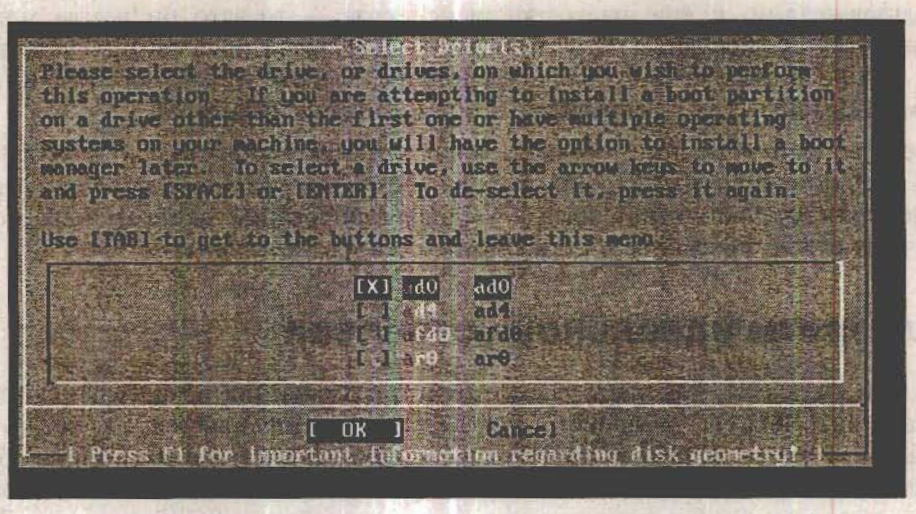


Рис. 4.15. Выбор диска для разметки

Совершенно очевидно, что, поскольку на компьютере не предполагается использование иных ОС, можно ограничиться одним слайсом для FreeBSD. Казалось бы, можно было бы создать отдельный слайс под прикладные программы (/usr или /usr/local) и пользовательские данные (/home), однако достигаемая этим их пущая сохранность — иллюзия: разрушение корневой файловой системы (а именно она больше всего подвержена напастям) все равно приведет к невозможности прочтения этих разделов обычными средствами.

Так что можно смело выбирать опцию **Entire disk** (Полный диск) (рис. 4.16). Тем более что в современных версиях это не влечет за собой эксклюзивного режима разметки (**Dangerously Dedicated**), делавшего ранее невозможным использование FreeBSD с другими ОС. Ныне же (начиная с версии 4.4) при любом способе нарезания слайсов первые 64 сектора (0—63) все равно резервируются под внешний загрузчик. И в дальнейшем, буде такое желание появится, ничего не мешает изменить размер BSD-слайса, создать на освободившемся месте новый раздел и инсталлировать туда хоть Windows, хоть Linux. Правда, единственная способная на такие действия программа, которую я пока видел, — это Acronis OS Selector. Но зато она может служить и внешним загрузчиком. Хотя со стандартным сочетанием FreeBSD—Linux—

Windows прекрасно справляется штатный загрузчик FreeBSD. Единственно, что установка Windows после FreeBSD его уничтожит (Acronis обладает замечательной способностью к регенерации, чтобы Windows ни делала с главной загрузочной записью). Но есть простые способы его восстановления.

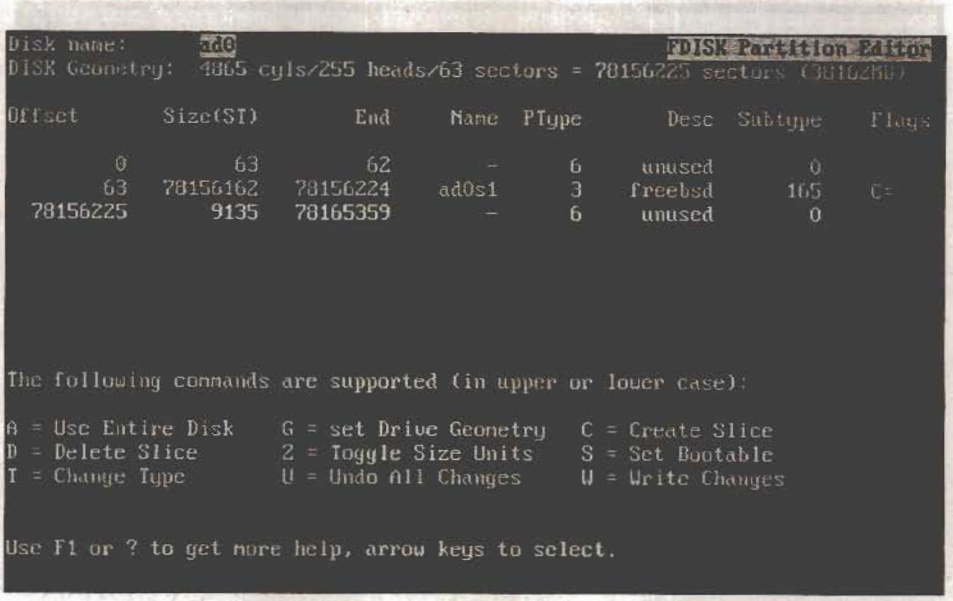


Рис. 4.16. Разметка диска: создание слайсов (slices)

Кстати о загрузчике — установить его предлагается сразу же по завершении создания слайсов (рис. 4.17). Из возможных опций можно без трепета выбирать запись загрузчика в MBR — уверяю, вреда ни малейшего.

Теперь слайсы нужно разбить на разделы (рис. 4.18). Вот тут скупиться не стоит — многие ветви файловой системы резонно обособить друг от друга физически. Для чего можно принять за основу вариант разбиения, предлагаемый по умолчанию опция **Auto Defaults** (Автоматически по умолчанию). Таким путем автоматически создаются (в версии 4.5, в других версиях набор разделов и их размеры могут отличаться:

- корневой раздел / объемом 128 Мбайт;
- swap-раздел, равный удвоенному объему оперативной памяти;
- разделы /var и /tmp объемом по 128 Мбайт;
- раздел /usr на всем оставшемся пространстве.

Попробуем примерить это разбиение к своим потребностям. Ста мегабайт с лишним для корневого раздела — вполне достаточно, в нем в итоге останется

не так уж и много: ядро, каталог устройств, каталог конфигурационных файлов, бинарные программы общего назначения и еще кое-какие мелочи (по объему, но не по значению).

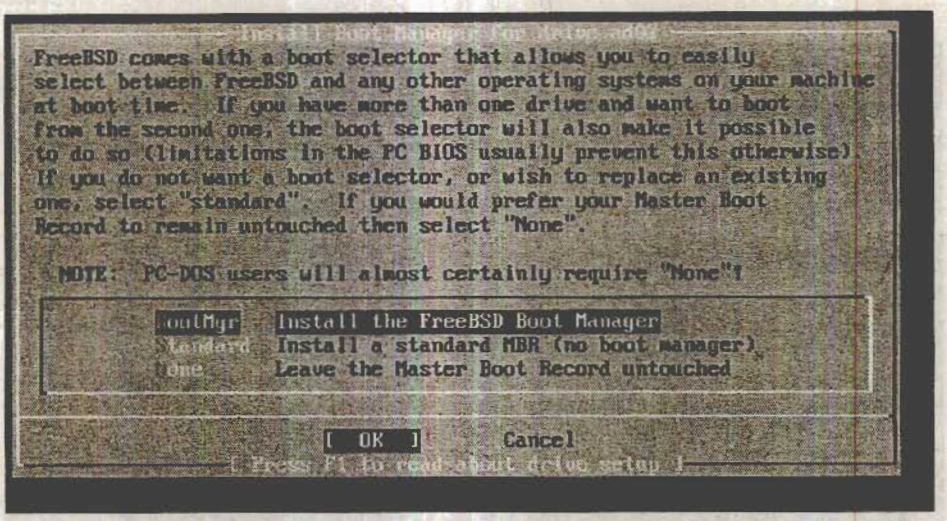


Рис. 4.17. Инсталляция начального загрузчика

Рис. 4.18. Разметка диска: создание разделов (partitions)

Относительно swap-раздела — вся документация по FreeBSD настоятельно рекомендует именно такой его размер. Впрочем, писалась она задолго до того, как нормой стало столько памяти, сколько сможешь унести с базара. Не знаю, насколько сейчас целесообразно отводить под swap-раздел от половины до полутора гигабайт. При памяти 768 Мбайт я ни разу не заметил за системой желания что-либо свопировать. Да и при 512 Мбайт ОЗУ размер свободной памяти приближался к исчерпанию (но — не исчерпывался) только при работе с полутораста мегабайтными изображениями в GIMP.

Необходимость размещения каталога /var в отдельном разделе обосновывается тем, что в него помещаются всякие файлы преходящего характера (почта, системные журналы и т. д.). И при большом количестве пользователей это легко может вызвать переполнение корневой файловой системы, если он находится с ней в одном разделе. Думается, на локальной машине такой проблемы возникнуть не должно. Но при нынешних объемах дисков сотня-другая мегабайт — такая мелочь, что об этом смешно говорить.

С другой стороны, в каталог /var по умолчанию пишутся и временные файлы некоторых пользовательских программ, например, утилиты convert из комплекта ImageMagik. И при больших объемах конвертируемых файлов он исчерпывается просто мгновенно. А поскольку суть оной программы — именно в конвертации большого количества больших файлов, при отсутствии дефицита дискового пространства есть смысл сделать раздел /var побольше.

Относительно раздела /tmp никаких наблюдений я не имею. Вероятно, можно оставить его таким, как предлагается по умолчанию. Хотя в более ранних версиях (4.2, 4.4) такого раздела вообще не создавалось — вероятно, обязательным он не является.

А вот для каталога /usr всего оставшегося места — это слишком жирно. Перво-наперво от него нужно отрезать изрядную часть под раздел /home — именно туда будут помещаться все пользовательские данные. А уж какой объем они займут — это каждый должен решать в меру своей испорченности.

Кроме того, можно выделить раздел для каталога /usr/local — в том случае, если предполагается устанавливать большое количество пакетов, по умолчанию все Ports и Packages инсталлируются именно туда. Наконец, если ориентироваться в основном на графические приложения, можно обособить в своем разделе /usr/X11R6 — в этот каталог попадают "иксы" со всеми их причиндалами и все "иксовые" приложения, из Distributions ли они взялись, из Ports или из Packages. Однако в любом случае под суммарное содержимое каталога /usr целесообразно будет отвести пару-тройку гигабайт, по нынешним временам — просто смех.

Опять-таки, излишняя экономия на суммарном объеме раздела /usr неоправданна, особенно если предполагается устанавливать приложения преимущественно из портов, а не из пакетов. В этом случае на локальный диск

помещаются не только скомпилированные программы, но и их исходные тексты — причем не только в архивированном виде, но и распакованные. Конечно, по окончании инсталляции порта исходники можно удалить. Но в этом случае теряется значительная часть смысла самой системы портов — при пересборке системы (например, обновлении версии) их потребуется скачивать и распаковывать заново. Так что с разделом /usr жадничать не стоит — иначе в ходе инсталляции очередного порта можно получить сообщение о переполнении файловой системы.

И последнее, что нужно иметь ввиду при создании разделов: файловая система FreeBSD сильно теряет в быстродействии при недостатке дискового пространства. Эмпирически установлено, что эффективное ее использование возможно максимум при 90-%-ном заполнении разделов. И потому любые сомнения в их размере должны решаться в бóльшую сторону — при объемах современных дисков это не будет очень обременительно.

После создания разделов следует обратить внимание на опцию **S = Toggle Softupdates**. Это штука чрезвычайно полезная. Во-первых, она ускоряет (иногда на порядок) все файловые операции (действуя, насколько я могу судить, сходно с кэшированием диска в Linux). Во-вторых, **Softupdates** повышает устойчивость файловой системы к сбоям работы машины (типа отключения питания, некорректного завершения сеанса, просто мертвого зависания). Поддержку **Softupdates** нужно обязательно включить сразу и для всех разделов — потом это будет сделать намного сложнее. Впрочем, в версии 4.5 почти так и сделано по умолчанию — опция **Softupdates** выставляется для всех создаваемых разделов. Правда, кроме корневого — существует мнение, что поддержка ее в этом разделе представляет некоторую опасность. Хотя я так и не понял, в чем она заключается, и, используя **Softupdates** для всех разделов, с неприятностями пока не столкнулся.

Теперь перейдем к установке базовых компонентов. Увлекаться этим занятием не следует: во-первых, их всегда можно добавить по мере необходимости, во-вторых, не существует простого способа чистого их удаления из системы (по крайней мере, мне такой способ не известен). И потому в соответствующем разделе меню логично выбрать пункт **Minimal** (Минимум) — это гарантирует, что все, без чего запуск системы невозможен, будет присутствовать. Не могу не порадовать — список этот до смешного короток, включая только **Binary base distribution** (Базовая система) и **Basic encryption services** (Базовые средства защиты); впрочем, подозреваю, что и без последнего можно было бы обойтись. После этого можно перейти к пункту **Custom** (Заказная) и тщательно прошерстить его на предмет полезности/избыточности на данном этапе.

К выбору аскета на предыдущей стадии здесь можно добавить кое-какую документацию (по крайней мере man-страницы), коллекцию портов (повторяю, это не реальные приложения, а только скелеты для их установ-

ки) и кое-что из исходников как минимум, исходные тексты ядра и заголовочные файлы (подпункты sys и include пункта src). Прочее можно отложить до тех времен, когда в этом прочем появится непреодолимая (материальная или духовная, как сказал бы Выбегалло) потребность.

В частности, я не стал бы пока устанавливать XFree — в базовом комплекте она идет за номером 3.3.6, а отказываться от преимуществ 4-й версии (хотя бы ради антиалиасинга шрифтов в KDE) — не вижу никаких причин. Впрочем, это — тема совершенно особого разговора.

Все, с установкой как с классом покончено. Обращаемся к конфигурированию. Где с негодованием отметаем пункт **Distributions** (Основные компоненты) (это, братцы, пройденный этап), но задумываемся над следующим — **Packages** (Пакеты). С одной стороны, и он не должен быть предметом излишнего энтузиазма. Во-первых, любые пакеты легко устанавливаются по мере необходимости. Во-вторых, установка из пакетов — это все же дополнительный способ наращивания системы против основного — компиляции из исходников посредством системы портов. И предназначен он в первую очередь для ознакомления с возможностями программ, что лучше делать после завершения настроек, в спокойной и дружеской обстановке. Единственно, что в полной текущей версии для поиска нужного пакета придется рыскать по двум-трем CD-дискам. Однако при современных объемах винчестеров никто не помешает нам потом переписать все пакеты туда, откуда и устанавливать.

Тем не менее, некоторое время пакетам уделить придется. Я уже как-то обмолвился, что инструментальный набор пользователя FreeBSD по умолчанию — достоин даже не аскета, а дзэн-буддиста, готового к переходу в состояние нирваны. А именно — в качестве командной среды предлагаются классические /bin/sh для обычного пользователя и csh — для администратора, редакторы — опять же классический vi достаточно старой версии и ее, простой в обращении, но имеющий очень ограниченные возможности. Так что если такой набор средств (а они понадобятся тут же, на стадии начального конфигурирования) покажется недостаточно комфортным — установка соответствующих аналогов из пакетов неизбежна. Хотя это можно сделать и не только средствами sysinstall, и не обязательно с дистрибутивного CD, о чем я скажу позднее.

Кроме того, можно установить также и XFree86 4-й версии — она также представлена только в форме package. Хотя лучше скачать ее с сайта проекта (**http://www.xfree86.org**) — там наверняка окажется более современный вариант, в том числе и в бинарном, откомпилированном специально для FreeBSD, виде. И установить собственными его (т. е. XFree86) средствами.

Далее, безусловно необходимо задать пароль для root-оператора. А вот от создания обычного пользователя пока лучше воздержаться. Конечно, работа от лица администратора — занятие нездоровое (а в FreeBSD еще и неудоб-

ное — по умолчанию на консоль root выводится масса системных сведений, например, о том, кто, где и когда авторизовался). Однако пока примириться с этим можно — иначе в дальнейшем потребуется довольно много ручной работы по настройке пользовательского окружения.

Следующие пункты — настройка консоли и часового пояса — принимаются. В частности, здесь можно полностью русифицировать консоль, указав кириллические шрифты для загрузки, раскладку клавиатуры и карту перекодировки KOI8-R в CP866. Последнее — практически необходимо, если предполагается использование мыши в текстовом режиме, иначе русские буквы будут изуродованы. К слову сказать — мышь тоже можно настроить на этой стадии, если она не принадлежит к семейству USB-моделей; в последнем случае все равно потребуется ручная доводка.

Далее — настройка сетевого интерфейса. Для машины в локальной сети, подключенной к Интернету через сервер DHCP (а в большинстве случаев это так и есть), дело это не простое, а очень простое. Система сама определит тип сетевой карты для шины PCI и устройство для нее (например, NE1000/2000 и /dev/ed0 соответственно), затем — имя домена, IP-адрес сервера, локальной машины и маску подсети. Единственное, что потребуется от пользователя — задать имя локальной машины (например, mycomputer). Если этого не произойдет — придется обратиться за этими сведениями к сетевому администратору.

Не сложнее дело обстоит и с локальной сетью без подключения к Интернету (например, домашней). Правда, IP-адреса здесь придется задавать вручную, из массива, зарезервированного для внутреннего употребления. Например, в качестве IP-адреса машин можно использовать адреса вида 192.168.1.0, 192.168.1.1 и т. д., а в качестве маски подсети — 255.255.255.0.

Вместо обычного сетевого Ethernet-соединения в домашних условиях можно, при отсутствии сетевых карт, объединить машины в сеть TCP/IP через последовательные или параллельные порты. Это потребует соответствующих кабелей (нуль-модемного или LPT-кабеля). Далее же все происходит так же, как и при настройке интерфейса для локальной сети.

Настройка сетевых служб, уровня защиты, стартовых сервисов — по желанию и потребности. А вот в пункте настройки терминалов (**TTYs**) должно довести до ума кириллизацию консоли посредством правки файла /etc/ttys. Это требует только смены типа терминала с cons25 на cons25r для всех задействованных виртуальных консолей. Заодно и количество их можно прибавить, если включенных по умолчанию восьми покажется недостаточно (об этом подробно будет говориться в *части II*).

Ну и последнее — настройка системы X Window и выбор оконного менеджера по умолчанию. Делать это имеет смысл только при установке XFree86 3.3.6; для 4-й версии нормальной настройки все равно не получится, лучше отложить этот процесс на потом.

Все — остальные пункты, как уже говорилось, сейчас неактуальны. Конечно, полезно было бы сделать т. н. fixit-дискету, однако при наличии полного дистрибутива в этом нет необходимости: там как fixit-носитель выступает цельный CD. Так что — выход из программы инсталляции и перезагрузка.

Начальные настройки

После перезагрузки, которая, надеюсь, проходит благополучно, система в принципе почти пригодна к работе. Обращаю особое внимание — результат этот был достигнут почти исключительно средствами sysinstall, лишь в одном случае осуществлялась прямая правка конфигурационного файла. Теперь же остаются финальные штрихи начальной настройки, которые потребуют уже обращения к текстовому редактору.

Для выполнения любых настроек после перезагрузки потребуется авторизация от лица root-оператора — что естественно, если на этапе установки мы отказались от создания обычных пользователей.

Введя логин администратора (root) и его пароль (разумеется, правильно — забывать таковой весьма нежелательно, хотя и не смертельно), можно увидеть приглашение командной строки пользовательской оболочки (командной среды, командного интерпретатора), имеющее по умолчанию вид #, где и следует вводить команды.

В качестве командной среды root-оператора по умолчанию выступает программа /bin/csh, каковая на деле представляет собой реализацию среды tcsh со всеми ее (весьма широкими) интерактивными возможностями. Так что ею и можно воспользоваться после некоторых настроек. Однако при желании (или наличии привычки) среду можно сменить, скажем, на bash, более привычную пользователям Linux. Для этого следует (предполагается, что поклонники bash озаботились установкой ее из пакета на стадии инсталляции), во-первых, проверить файл /etc/shells, содержащий список всех допустимых командных сред, с помощью команды `less` (например)

```
$less /etc/shells
```

на наличие в нем строки вида

```
/usr/local/bin/bash
```

И если таковой там не окажется — вызвать этот файл в текстовом редакторе:

```
$ee /etc/shells
```

и внести эту строку вручную. Далее, поскольку командная среда — один из атрибутов учетной записи пользователя (пользовательского аккаунта), в данном случае администратора, следует внести в нее соответствующие изменения. Это можно сделать различными способами, из них самый

простой — прямой командой pw, о которой подробнее будет рассказано в *главе 11*:

```
pw usermod root -s /user/local/bin/bash
```

Как "умолчальная" csh, так и любая из сменных командных сред потребует для комфортной работы некоторой настройки, что достигается правкой соответствующего конфигурационного файла, именуемого также dot-файлом или rc-файлом (от *resoucer control*). Для csh это /.cshrc или /root/.cshrc, для bash пока используем /.profile или /root/.profile — одноименные файлы в корневом каталоге и каталоге /root идентичны по содержанию и представляют собой жесткие ссылки на одни и те же данные, так что редактировать можно любой из них, изменения автоматически отразятся в обоих (подробности о жестких ссылках и вообще организации файлов будут рассмотрены в *главе 9*).

Подробный разговор о настройках командных сред — впереди. Пока же обойдемся элементарным минимумом изменений, способным, однако, здорово облегчить жизнь, а именно: сменой "умолчальных" редактора и pager-программы, предназначенной для просмотра текстовых файлов, занимающих более одного экрана монитора (в частности — man-страниц и прочей документации). По умолчанию в качестве таковых по старой UNIX-традиции выступают, соответственно, vi и more.

Редактор vi (вернее, его современные модификации типа Vim) — чрезвычайно могучее средство для работы с текстами любого рода. Однако пользователю без должного его знания (и практических навыков на рефлекторном уровне) он может показаться порождением больного ума садомазохиста. И потому на первом этапе знакомства с системой (дабы не привить отвращения, причем — абсолютно неоправданного, к этому прекрасному инструменту) его лучше заменить на что-либо более простое, например, тот же ее (или, при желании, le). Возможно, после изучения этого редактора (чем может несколько поспособствовать изучение соответствующей главы *части II*) у вас и появится желание вернуться к нему — но уже на новом этапе (когда, надеюсь, потребности в моих рекомендациях не будет).

Что же касается more, то использование этой программы не оправдано ничем, кроме традиции: утилита проекта GNU под названием less (стандартный, кстати, pager во всех дистрибутивах Linux) обладает всеми ее возможностями плюс многими другими (хотя бы — возможностью двунаправленной навигации по тексту).

Тем паче, что особых усилий это не требует: открываем в редакторе ее необходимый файл и переопределяем переменные среды командной оболочки (variable environment, что это такое — будет рассказано в главе о Shell). В /.cshrc (или, что без разницы, в /root/.cshrc) ищутся строки вида

```
Setenv  EDITOR  vi
Setenv  PAGER  more
```

определяющие редактор и pager, и меняем значения переменных EDITOR и PAGER на ее и less. В /.profile (/root/.profile) строки

```
EDITOR=vi
export EDITOR

PAGER=more
export PAGER
```

преобразуем в

```
EDITOR=ee
export EDITOR

PAGER=less
export PAGER
```

обращая внимание на то, что каждое определение переменной среды должно сопровождаться ее экспортом. Далее, возможно, следует поправить пути для поиска исполнимых файлов. Дело в том, что в FreeBSD (и прочих UNIX) по умолчанию на этот предмет не просматривается текущий каталог, что вызывает недоумение у пользователей DOS/Windows. Чтобы это происходило, обозначение текущего каталога (символ точки — ".") нужно добавить явным образом в список путей, имеющий в /.cshrc (/root/.cshrc) вид

```
set path = (/sbin /bin /usr/sbin... /usr/X11R6/bin)
```

а в /.profile (/root/.profile) —

```
PATH=/sbin:/bin:/usr/sbin:...:/usr/X11R6/bin
export PATH
```

Нужно помнить только, что добавление текущего каталога для конфигурационного файла оболочки администратора (и, в меньшей мере, обычного пользователя) в некоторых случаях представляет собой потенциальную угрозу безопасности системы и потому не рекомендуется (хотя, думаю, на локальной домашней машине допустимо). А вот освободить пути поиска от лишних компонентов (например, /games, появляющегося там подчас, даже если игры и не устанавливались) — дело святое...

Завершив редактирование файла конфигурации оболочки, нужно завершить сеанс работы root-оператора (командой `logout` или `exit` — результат их в принципе несколько различен, но в данном случае они идентичны) и авторизоваться в системе заново — изменения вступят в силу, и комфортная среда для работы администратора создана. Теперь-то самое время вспомнить, что работать от лица последнего — занятие нездоровое и неудобное. А избежать его можно созданием обычного, непривилегированного, пользователя.

Если в системе предполагается только один пользователь (помимо root) — делаем это сразу. На сей предмет существуют различные средства, пока воспользуемся самым простым — командой

```
$adduser
```

Запущенная впервые, она запросит нас для начала о некоторых общих атрибутах, которые будут в дальнейшем распространяться по умолчанию на все создаваемые учетные записи. Среди них — определение предпочтительной командной оболочки. По умолчанию в этой роли предлагается /bin/sh, но ее интерактивные возможности очень ограничены, и лучше сразу заменить ее на /bin/csh, /usr/local/bin/bash, /usr/local/bin/tcsh или /usr/local/bin/zsh (разумеется, предварительно они должны быть установлены в системе и внесены в файл /etc/shells, как было описано выше).

Далее, будет вопрошено, следует ли копировать некий предопределенный набор конфигурационных файлов из каталога /etc/skel в домашние каталоги создаваемых пользователей. На своих личных машинах я всегда от этого отказываюсь — проще вручную скопировать и отредактировать необходимые именно мне файлы. Однако если предполагается создание нескольких пользователей, такая возможность может оказаться полезной. Нужно только заранее определить набор файлов каталога /etc/skel и их содержимое (к чему я вернусь чуть ниже). В числе общих последуют вопросы об отправке сообщений создаваемому пользователю (от чего я отказываюсь), использовании ими пароля и общем пути к домашнему каталогу (/home по умолчанию) — причин отвергать и то, и другое нет ни малейших.

Все общие атрибуты сохраняются в файле /etc/adduser.conf и введенные нами ответы на их запрос в дальнейшем будут предлагаться по умолчанию. После этого наступает апофеоз — собственно создание пользователя посредством определения:

- его регистрационного имени — login, теоретически любой набор буквенных и цифровых (но не специальных) символов, но обычно рекомендуется что-нибудь подобное собственному имени;
- полного (паспортного) имени — впрочем, документов система не спрашивает и сохранить инкогнито тоже не возбраняет;
- командной оболочки (в скобках — вариант по умолчанию, определенный ранее для его выбора достаточно нажать <Enter>);
- домашнего каталога — /home/username, где последний член совпадает с регистрационным именем;
- идентификатора пользователя (*UID*, *U*ser *ID*entificator, по которому он собственно и опознается системой) — числа вида 1xxx; по умолчанию первый создаваемый пользователь получает UID 1000, второй — 1001 и т. д.;

- класса пользователя; это — очень важный атрибут, и если вместо предлагаемого здесь класса по умолчанию (default) задать класс `russian`, автоматически установятся все локально-зависимые переменные среды (т. е. будет выполнена русская локализация); это — одна из причин, почему следовало отказаться от создания пользовательского аккаунта средствами sysinstall — в них определение класса пользователя не предусмотрено;
- основной группы пользователя (по умолчанию совпадает с его регистрационным именем;
- дополнительных групп, в которые включается создаваемый пользователь; вместо (или вместе) с предлагаемой здесь по умолчанию группой `guest` здесь следует обязательно прописать (для себя, любимого, конечно) и группу `wheel` — только ее члены имеют возможность временно получить права администратора командой `su`.

Напоследок нужно ввести (с повторением, во избежание ошибок) пароль пользователя, который при этом не отображается на экране даже в виде звездочек. После этого на экран выводятся все введенные атрибуты пользователя (за исключением пароля, разумеется) и задается вопрос об их правильности. Если с этим согласиться, пользовательский аккаунт будет создан (при отказе — процедуру придется повторить с самого начала, никаких следов от ранее введенных данных не останется).

Теперь можно спокойно оборвать сеанс администратора (командой `logout` или `exit`) и авторизоваться в качестве только что созданного пользователя. Поскольку при необходимости всегда можно получить права root-оператора командой

```
$su
```

или

```
$su -l
```

Первая просто дает нам полномочия администратора, сохраняя все настройки среды пользователя, эту команду отдавшего. Команда же `su` с опцией `-l` приводит к перечитыванию конфигурационного файла оболочки пользователя (/root/.csh или /root/.profile). В зависимости от контекста предпочтительной может быть та или другая форма. В частности, вторая форма используется в том случае, если требуется получить доступ к переменным окружения администратора в полном объеме (в частности, к путям поиска исполнимых файлов, которые для root и обычного пользователя могут быть разными).

Так что воспользуемся полученной возможностью и, получив права суперпользователя (в данном случае — любым способом), продолжим начальные настройки. Следующим на очереди у нас — редактирование главного конфигурационного файла системы, /etc/rc.conf, или сценария начальной загрузки.

Следует заметить для начала, что в FreeBSD очень последовательно проводится идея двухуровневого конфигурирования. То есть, например, файл rc.conf существует в двух вариантах: как /etc/defaults/rc.conf и как /etc/rc.conf. Первый из них, для редактирования не предназначенный, описывает все возможные опции начальной загрузки, и присвоенные им значения выступают в качестве таковых по умолчанию. В файле же /etc/rc.conf описываются только те опции, значения которых в данной системе должны отличаться от "умолчальных" — именно этот файл изменяется при настройках через sysinstall или вручную, в текстовом редакторе. Последним мы и займемся.

Первое, что тут может потребоваться, — это некое косметическое действие. Как мы помним, на стадии начальной установки было предложено указать на необходимость загрузки кириллических экранных шрифтов (в кодировке CP866), клавиатурной раскладки (в кодировке KOI8-R) и таблицы преобразования кодировки ввода в кодировку вывода (т. н. screenmap). В результате чего сразу же после установки появляется возможность работать с русскими документами, например, в текстовом редакторе. А после регистрации обычного пользователя и установки для него русской локали символы кириллицы будут видны и в командной оболочке. Например, в ответ на команду

```
$date
```

последует вывод текущего числа, месяца, года, дня недели и времени по-русски и в формате, принятом в нашей стране:

```
сб  23 фев 2002 04:26:11 MSK
```

поскольку формат представления времени, как и десятичных дробей, валют и прочего также входит в понятие локали.

Однако вернемся к нашим баранам, т. е. к файлу /etc/rc.conf. Возможность работы с символами кириллицы описывается следующими его строками:

```
font8x14="cp866-8x14"
font8x16="cp866b-8x16"
font8x8="cp866-8x8"
...
keymap="ru.koi8-r.kbd"
...
scrnmap="koi8-r2cp866"
```

сгенерированными программой sysinstall. Первые три строки отвечают за загрузку шрифтов, предназначенных для разных экранных разрешений. Подробно эта тема рассматривается в главе о системной консоли FreeBSD. Пока же замечу, что необходимости в шрифте cp866-8x14 нет вообще (он предназначался для антикварных EGA-мониторов), а шрифт cp866-8x8 используется лишь для работы с высокой плотностью строк (более 30). Так что

реально нужен обычно только шрифт с матрицей 8×16. Однако умолчальный его вариант выглядит крайне неэстетично, и из соображений изящества его лучше заменить на cp866-8x16.

В дальнейшем я расскажу, как добиться максимально привлекательного вида текстовой консоли. И вообще, к редактированию /etc/rc.conf придется еще не раз обращаться по различным поводам. Пока же — последний штрих, настройка мыши или какого-либо иного позиционирующего устройства (трекбола, трекпойнта, тачпада). Ведь это — вещь абсолютно необходимая для подавляющего большинства настольных применений.

Конечно, велика вероятность того, что мышь благополучно заработает сразу после установки. Это относится к стандартным сериальным мышам и PS/2-моделям. Трекпойнты и тушпады ноутбуков также обычно используют протокол PS/2, и с ними проблем возникнуть не должно. Программой sysinstall поддерживаются также т. н. шинные мыши (bus-mouse), подсоединяемые к собственной плате расширения, и фирменные протоколы некоторых графических планшетов. Так что практически настройки могут потребовать только мыши для шины USB, получившие в последнее время широкое распространение.

За поддержку мыши отвечают следующие, сгенерированые программой sysinstall, строки файла rc.conf:

```
moused_enable="YES"
moused_type="auto"
moused_port="/dev/psm0"
```

Приводимый пример относится к мыши PS/2. Если такая мышь не работает после инсталляции — скорее всего, в ходе ее конфигурирования программой sysinstall был активизирован пункт меню **Disable** (Отключить), после чего первая строка получит значение `"NO"`. В этом случае достаточно заменить его на `"YES"` — и при следующем запуске машины курсор мыши волшебным образом возникнет в текстовой консоли. Этого, однако, не случится для мышей с USB-разъемом. Однако дело это поправимое.

Строка `moused_type` определяет протокол, используемый наличествующей мышью. Обычно (в частности, для всех виденных мной мышей PS/2) он автоматически (и — правильно) определяется системой. Если этого по каким-либо причинам не случилось, следует подставить значение для конкретного устройства. Полный список доступных вариантов получается обычным образом — посредством

```
$man 8 moused
```

Список этот весьма обширен, включая протоколы для моделей, которых я в глаза не видел. Так что вероятность быть обладателем исконно чуждого

FreeBSD грызуна — близка к нулю. Однако в означенном manual подчеркивается, что для USB-мышей значение "auto" — единственно возможное.

Конечно, указания на протокол может оказаться недостаточно. И потребуется вручную задать устройство, отвечающее за функционирование мыши, т. е. значение строки `moused_port`. Напомню, что в FreeBSD любое физическое устройство — не более, чем файл в каталоге /dev. В случае с мышами файл устройства корреспондирует именно с физическим типом ее интерфейса. Так, мыши с разъемом PS/2 соответствует файл под именем (вот странно-то) /dev/psm0 (а видел ли кто в машине больше одного PS/2-разъема для мыши?), сериальной мыши — один из файлов вида /dev/cuaa# (где # — номер COM-порта, к которому мышь подсоединена), USB-мыши — файл /dev/usm0 (скорее всего, хотя теоретически можно представить себе целый выводок USB-мышей вокруг одной машины).

Кроме того, в том же каталоге /dev может обнаружиться файл устройства /dev/mouse. Это — не более, чем символическая ссылка на один из перечисленных выше файлов устройств, что легко установить командой `ls -l`. Наконец, имеется еще и файл /dev/sysmouse. Каковой представляет собой нечто вроде виртуального драйвера мыши, требуемого для ее использования в системе X Window, о чем будет говориться в соответствующей главе.

Так вот, именно в случае USB-мыши ручная правка значения `moused_port` неизбежна — если напрячь память, можно припомнить, что при конфигурировании системы через sysinstall USB-порт для мыши просто не предлагался. Так что данная строка должна приобрести вид

```
moused_port="/dev/ums0"
```

после чего дело консольной мыши можно считать законченным. К сожалению, использовать в консоли FreeBSD колесико для скроллинга не удастся (хотя функции средней клавиши оно выполняет исправно) — для этого потребуется запуск системы X Window.

Кстати, о системе X Window — ее тоже можно настроить на этом этапе. Однако, оправдываясь тем, что XFree86 составной частью FreeBSD не является, предпочитаю вы*дели*ть это *дело* в от*дель*ное производство.

Вот, собственно, и все. Комфортная среда обитания создана. И мы готовы к дальнейшим подвигам во славу нашей прекрасной дамы — леди Free...

Дополнительные источники

- Базовые сведения по установке и начальной настройке FreeBSD содержатся в официальном руководстве пользователя (Handbook, рус. пер. — **http://www.FreeBSD.org.ua/handbook/**). Ну, а наиболее подробные сведе-

ния по этому предмету — в Иллюстрированном руководстве по установке FreeBSD Рэнди Пратта, недавно появившийся русский перевод которого можно обнаружить на **http://unix1.jinr.ru/~lavr/** и еще некоторых сайтах (например, на **http://www.opennet.ru**).

- Из прочих источников наиболее подробные сведения по этим вопросам содержатся в Заметках по FreeBSD Андрея Лаврентьева (**http://unix1.jinr.ru/~lavr/**) и на сайте Дмитрия Карпова (**http://www.pi2.ru:8101/prof/techsupp/fbsdinst.htm**). Правда, материалы последнего относятся преимущественно к довольно старым версиям системы.
- Подробные сведения о подготовке дисковых разделов для FreeBSD содержатся в статье Дуга Вайта "Форматирование носителей для использования с FreeBSD" (рус. пер. — **http://www.FreeBSD.org.ua/articles/formatting-media/**).

Глава 5

Установка программ

По завершении начальных настроек пора вспомнить и о том, ради чего устанавливалась FreeBSD — ради практической работы, не правда ли? До сих пор молчаливо подразумевалось, что, за немногим исключением, нам довольно было базового комплекта утилит и приложений, да и то не полностью. Однако чтобы двигаться дальше, потребуется уже изрядное их количество. Так что самое время поговорить о методах установки программ в FreeBSD.

Установка программ в FreeBSD включает в себя ряд моментов. Во-первых, в процессе работы может выявиться необходимость в доустановке базовых компонентов системы. Во-вторых, практически неизбежно потребуются дополнительные программы из дистрибутива, устанавливаемые либо из пакетов, либо из коллекции портов. В-третьих, не исключена возможность того, что требуемая программа не включена в дистрибутив ни в каком виде, но доступна в исходных текстах, предназначенных для некоего абстрактного UNIX. И последнее — установка оконной системы X, которая не является частью FreeBSD и имеет свою специфику. Все эти моменты, а именно:

- доустановка базовых компонентов;
- установка программ из пакетов;
- установка программ через систему портов;
- сборка программ из абстрактных исходников;
- установка оконной системы X

и составят предмет настоящей главы.

Доустановка базовых компонентов

Это — наименьшая проблема из всех в данном круге вопросов. Для добавления базовых компонентов запускается программа

```
$/stand/sysinstall
```

Далее в главном меню ее выбирается пункт **Custom** (Заказная), в нем — подпункт **Distributions** (Основные компоненты), к котором и выбираются необходимые программы, точно так же, как это делалось при инсталляции. Вслед за этим в пункте **Media** (Источники установки) выбирается источник инсталляции. В случае, если таковым является дистрибутивный CD, он должен быть вставлен в привод, но ни в коем случае не смонтирован командой `mount` — программа sysinstall проделывает это автоматически. После этого выбор подтверждается через пункт **Commit** (Подтверждение) — и все отмеченные компоненты устанавливаются в системе.

Здесь следует иметь в виду только два момента. Во-первых, ранее установленные (например, при инсталляции) составляющие базовой системы в меню **Distributions** (Основные компоненты) никак не отмечены — о них нужно просто помнить, чтобы не проделывать дважды одну и ту же работу. Ничего страшного в этом, конечно, нет, но потеря времени может быть ощутимой.

Во-вторых, теоретически для доустановки базовых компонентов предназначается пункт **Configure** (Настройка) с последующим выбором подпункта **Distributions** (Основные компоненты). Однако в ряде версий FreeBSD (например, в текущей на момент написания этих строк — версии 4.5) это не возымеет никаких последствий — ничего из отмеченного таким образом не устанавливается, по завершении выбора происходит просто выход из программы sysinstall. Возможно, в последующих версиях положение изменится.

И вообще, в различных источниках иногда отмечается не вполне надежная работа программы sysinstall на стадии конфигурирования. Я ни с какими проблемами в ней не сталкивался (кроме указанной выше забавной особенности). Однако в ряде случаев, возможно, более надежным окажется ручное извлечение компонентов дистрибутива. Тем более, что таким образом можно установить некую конкретную программу или ее исходный текст.

Для ручного извлечения компонентов следует представить себе структуру дистрибутива. Как уже говорилось, входящие в него файлы имеют вид имя_каталога.??, где ?? — пара латинских литер, начиная с aa, ab, ac и т. д. (исключение — файлы XFree86, соответствующие структуре оригинальных бинарных пакетов). Несмотря на непривычный вид, файлы эти — просто фрагменты единого компрессированного архива с расширением tgz (собранного воедино программой tar и сжатого программой gzip), разбитого на части (размером 240 640 байт) программой split. Поэтому, после объединения их в первозданное состояние, с ними можно обращаться как с обычными архивами — просматривать содержимое, распаковывать полностью или извлекать из архива только отдельные части.

В принципе, процедура слияния, распаковки и установки осуществляется инсталляционным сценарием install.sh, находящимся в том же каталоге, что и файлы расщепленного архива. Именно он вызывается для исполнения

при выборе соответствующего компонента дистрибутива. Однако ничто не мешает выполнить все это и вручную.

Для объединения файлов-фрагментов можно воспользоваться командой `cat`. Например, в ходе освоения FreeBSD мы ощутили непреодолимую потребность посмотреть на входящие в ее комплект игры (games), проигнорированные при установке (хотя сразу предупреждаю — смотреть там абсолютно не на что). Для этого, получив командой `su` права администратора, монтируем дистрибутивный CD:

```
$mount /dev/acd0c /cdrom
```

командой `exit` возвращаемся в режим обычного пользователя, переходим в каталог с играми

```
$cd /cdrom/games
```

и объединяем файлы вида games.??, перенаправляя вывод в файл произвольного каталога (например, своего домашнего) на жестком диске:

```
$cat games.aa games.ab... games.ak > ~/games.tgz
```

Далее содержимое результирующего архива можно просмотреть командой

```
$tar tzf games.tgz | less
```

или распаковать его

```
$tar xvzf games.tgz
```

Впрочем, приведенный пример несколько надуманный. Более реальный случай — извлечение из дистрибутивного архива исходных текстов конкретной программы для внесения в них требуемых изменений. Рассмотрим требующуюся для этого последовательность действий на конкретном примере, попытавшись, заодно, обойтись без записи промежуточного файла, для чего воспользуемся средствами перенаправления и конвейеризации команд (вкратце затронутыми во вводном конспекте и подробнее рассмотренными в главе о командных оболочках).

Итак, чтобы извлечь из дистрибутива некий конкретный файл (или группу связанных файлов), следует для начала отыскать их в дистрибутивных архивах. Например, нам хотелось бы установить исходные тексты командной оболочки /bin/csh. Монтируем дистрибутивный CD-ROM и переходим в каталог с исходными текстами:

```
$cd /cdrom/src
```

Логично предположить, что исходники для программ, содержащихся в установленной системе в каталоге /bin, могут быть обнаружены в файлах вида sbin.?? (благо, их — всего два, sbin.aa и sbin.ab). Проверяем это, объединяя их командой `cat`, результат которой направляем, однако, не в файл, на на ввод команды `tar` с соответствующими опциями для просмотра содержи-

мого архива. А поскольку можно заранее предположить, что список файлов в архиве займет не один экран, вводим третий элемент конвейера в нашу командную конструкцию — команду `less` для постраничного просмотра:

```
$cat sbin.aa sbin.ab | tar tzvf - | less
```

Ответом на данную конструкцию будет список вида:

```
drwxr-xr-x root/wheel        0 28 янв 14:54 2002 bin/
-rw-r--r-- root/wheel      472 31 окт 12:43 2000 bin/Makefile
...
drwxr-xr-x root/wheel        0 28 янв 14:54 2002 bin/csh/
-rw-r--r-- root/wheel     3021 25 апр 13:12 2001 bin/csh/Makefile
```

С радостью обнаруживаем, что требуемые нам файлы bin/csh/* в этом списке присутствуют. Остается извлечь их — с помощью несколько модифицированного конвейера команд:

```
cat sbin.aa sbin.ab | tar xzvf - -C ~/target bin/csh
```

где ~/target — целевой каталог в домашнем каталоге пользователя, куда будут извлечены исходные тексты, а bin/csh — те компоненты архива, которые подлежат извлечению (в данном случае все файлы подкаталога /bin/csh).

Смысл опций команды `tar` я пока описывать не буду — читателю предлагается пока либо принять их на веру, либо обратиться к главе об управлении файлами средствами командной оболочки. Но говорю авторитетно — следствием этих действий будет появление в каталоге ~/target подкаталога ~/target/bin/csh с полным набором исходных текстов данной программы.

Установка из пакетов

Следующее действие, необходимое для доведения системы до полностью рабочего состояния, — это установка дополнительных программ из дистрибутива. Сделать это можно двумя специфичными для FreeBSD средствами — установив их из пакетов или из коллекции портов. При этом основным способом наращивания функциональности системы является второй. Однако установка из пакетов проще и более подходит для первичного ознакомления. Поэтому с нее и начнем.

Пакеты — откомпилированные бинарные программы, собранные в виде компрессированных архивов tgz. Содержащих, кроме собственно файлов программы (исполнимых, документации, необходимых библиотек и прочего), также сценарии для их правильной установки. Они присутствуют на дистрибутивных CD (инсталляционном, 3-м и 4-м) в каталоге packages, разделяемом, в свою очередь, на многочисленные подкаталоги (archivers, audio, editors, graphics и т. д.). Впрочем, содержимое их — просто символические

ссылки на собственно файлы пакетов, которые собраны в каталоге packages/All.

Содержимое каталогов packages/All на отдельных дисках дистрибутива варьирует от версии к версии (как, впрочем, и представительность пакетов). Более того, одни и те же пакеты могут повторяться на разных дисках. Это связано с понятием взаимозависимости пакетов — т. е. одни пакеты требуют для установки и работы наличия в системе других (как правило, системных библиотек и тому подобных разделяемых ресурсов).

Коллекция пакетов на дистрибутивных CD заведомо неполна: новые пакеты появляются постоянно на протяжении всей активной жизни каждой версии. И потому для пополнения ее следует использовать FTP-серверы проекта FreeBSD, где все доступные на данный момент пакеты собраны в каталоге /pub/FreeBSD/stable/packages/All.

Пакеты обычно более или менее жестко привязаны к номеру версии системы. Это связано с той же взаимозависимостью пакетов, т. к. инсталляционные сценарии их, как правило, ссылаются на конкретные номера версий общесистемных библиотек, которые постоянно изменяются. Впрочем, некоторые программы могут быть установлены из пакетов, невзирая на версии дистрибутива (по крайней мере в пределах одной ветви).

Пакеты могут быть установлены двумя способами. Первый — уже описанной программой sysinstall. Для этого требуется вставить какой-либо из дистрибутивных CD в привод и, ни в коем случае не монтируя его, запустить от лица администратора программу

```
$/stand/sysinstall
```

Обращаю внимание на необходимость указания полного пути — каталог /stand не определен в переменной PATH ни пользователей, ни даже root-оператора.

Далее, после запуска программы, выбирается пункт **Configure** (Настройка) главного меню sysinstall, в нем — подпункт **Packages** (Пакеты), определяется источник инсталляции пакетов (например, CD/DVD), после чего в списке пакетов (алфавитном, в пункте **All** (Все), или в тематических подкаталогах) отыскивается требуемый пакет. Отметив все необходимые пакеты, выходим из меню **Packages** (Пакеты) и производим собственно их установку.

Система пакетов полностью берет на себя заботу о контроле зависимостей. Так, если мы устанавливаем графический редактор GIMP, нам достаточно отметить только его пакет: все требуемые для его работы компоненты (системная библиотека GTK, библиотеки для работы с графическими файлами разных форматов) будут найдены и установлены автоматически.

Чтобы избежать перебора дисков в поисках нужного пакета, содержимое каталога packages со всех дистрибутивных CD можно скопировать предварительно на жесткий диск (при отсутствии дефицита дискового пространства,

разумеется). В этом случае после запуска программы /stand/sysinstall в качестве источника инсталляции следует указать соответствующий раздел жесткого диска.

Наконец, при наличии подключения к Сети (не обязательно постоянного, достаточно приличного модемного соединения) пакеты могут быть установлены по FTP-протоколу, как в активном, так и в пассивном (т. е. через брандмауэр) режиме.

Как можно убедиться, установка пакетов с помощью программы sysinstall очень проста, но в некоторых случаях может показаться громоздкой (особенно если требуется установить одну-две программы). И потому часто целесообразней обратиться к системе управления пакетами, тем более, что она предоставляет и многие дополнительные возможности — например, практически "чистого" удаления установленных пакетов.

Система управления пакетами включает в себя восемь утилит, функции которых достаточно прозрачно определяются названием: pkg_add, pkg_create, pkg_info, pkg_update, pkg_check, pkg_delete, pkg_sign, pkg_version. В данный момент нас интересуют только три из них — pkg_info, pkg_add, pkg_delete (к остальным вернемся после рассмотрения системы портов).

Программа pkg_info, как легко догадаться, предназначена для получения информации и пакете (или пакетах), как установленных в системе, так и отсутствующих. Формат ее запуска

```
$pkg_info имя_пакета
```

ответом на что будет краткая характеристика запрошенной программы (информативность оной — исключительно на совести разработчика). Например, чтобы получить представление о командной оболочке zsh, даем команду

```
$pkg_info zsh-4.0.4.tgz
```

Обращаю внимание, что аргументом команды является полное имя файла пакета с указанием версии. В ответ будет выведено сообщение примерно следующего вида:

```
Information for zsh-4.0.4.tgz:
Comment:
The Z shell
Description:
Zsh is a UNIX command interpreter (shell) which of the standard shells
most resembles the Korn shell (ksh), although it is not completely
compatible.  It includes enhancements of many types, notably in the
command-line editor, options for customising its behaviour, filename
globbing, features to make C-shell (csh) users feel more at home and
extra features drawn from tcsh (another 'ustom' shell).
```

И т. д.

Команда `pkg_info` имеет большое количество опций, с которыми можно ознакомиться через ее man-страницу. В числе их — возможность просмотреть описание всех установленных в системе пакетов (`pkg_info-a`), определить каталог (`pkg_info-p имя_пакета`), в который будут записываться отдельные компоненты пакета — т. н. префикс (Prefix, забегая вперед, замечу, что таковым по умолчанию является каталог /usr/local), просмотреть список входящих в пакет файлов с их полными путями (`pkg_info-L имя_пакета`), задать зависимости между установленными пакетами (`pkg_info-R имя_пакета`), и многие другие.

Наиболее полную информацию о пакета можно получить командой

```
$pkg_info -v имя_пакета
```

результат которой рассмотрим на примере пакета scr2png — специфичной для FreeBSD программы, позволяющей получать истинные (графические) скриншоты текстовой консоли. Итак, даем команду

```
$pkg_info -v scr2png-1.1.tgz
```

в ответ на что получаем следующие сведения:

- комментарий, т. е. краткую характеристику функциональности:

  ```
  Converts the output of "vidcontrol -p" to PNG
  ```

- список пакетов, от которых зависит запрошенный пакет:

  ```
  png-1.2.1
  jpeg-6b_1
  ...
  ```

- описание программы, более или менее полное;
- полный список файлов пакета, включая те, которые заносятся в базу данных (о последней — чуть ниже);
- положение этих файлов в общем дереве каталогов корневой файловой системы.

Таким образом, программа pkg_info позволяет получить почти исчерпывающую информацию о назначении и составных частях любой программы, даже не устанавливая ее в системе. Что позволит принять решение о целесообразности собственно установки. Для последней цели предназначена программа pkg_add. Формат ее запуска также очень прост

```
$pkg_add имя_пакета
```

Разумеется, архив пакета должен находиться в текущем каталоге (или — указан полный путь к нему). В качестве аргумента, как и в случае программы pkg_info, необходимо указать полное имя архивного файла, например:

```
$pkg_add scr2png-1.1.tgz
```

после чего пакет будет установлен в системе и доступен для запуска. Правда, до завершения текущего пользовательского сеанса запустить установленную программу можно будет только с указанием полного пути

```
$/usr/local/bin/имя_программы
```

т. к. пути поиска исполнимых файлов будут заново считаны только после перезапуска командной оболочки.

Столь безболезненная установка пакета произойдет только в том случае, если не будет обнаружено нарушение зависимостей пакетов. В отличие от программы sysinstall, pkg_add не обеспечивает автоматического контроля зависимостей, т. е. сама по себе не устанавливает недостающих пакетов. Однако, запущенная без опций, она сообщает о всех нарушениях зависимостей, указывая список пакетов, необходимых для функционирования запрашиваемого. И, соответственно, все они могут быть установлены той же программой pkg_add с указанием имен всех требуемых файлов (разделенных пробелами) в качестве аргументов. Так, для случая пакета scr2png форма его установки в общем случае будет следующей:

```
$pkg_add png-1.2.1.tgz jpeg-6b_1.tgz freetype2-2.0.6.tgz gd-1.8.4_5.tgz
scr2png-1.1.tgz
```

В принципе pkg_add позволяет установить пакет и с нарушением зависимостей, для чего предназначена опция `-f` (от *force*). Разумеется, работоспособность установленной программы в этом случае не гарантируется.

Как и любая уважающая себя UNIX-программа, pkg_add имеет немалое количество опций, которые можно изучить посредством ее man-страницы. Среди них — опция `-p`, задающая префикс для установки (т. е. каталог, корневой по отношению ко всем подкаталогам данного пакета). Принятый по умолчанию во всех пакетах FreeBSD префикс — /usr/local, т. е. исполнимые файлы программы помещаются в каталог /usr/local/bin, разделяемые библиотеки — в каталог /usr/local/lib, страницы экранной документации — в каталог /usr/local/man, и т. д. Исключение — программы графического режима, предназначенные для работы с системе X Window: для них в качестве префикса по умолчанию принимается каталог /usr/X11R6/.

Для записи в любой из указанных префиксных каталогов требуются права администратора. То есть штатным образом установить пакет можно только, авторизовавшись в качестве root-оператора (или получив временно его привилегии командой `su`). Если это невозможно или почему-либо нежелательно, можно в качестве префиксного каталога задать, например, домашний каталог пользователя. Правда, далеко не все программы, будучи установлены в каталоги, отличные от принятых по умолчанию, сохранят работоспособность.

Сценарии установки, входящие в состав пакетных архивов, выполняют, кроме собственно инсталляции программы, еще одну важную функцию — регистри-

руют установленный пакет в соответствующей базе данных. Она расположена в каталоге /var/db/pkg/ и представляет собой серию подкаталогов, имена которых совпадают с именами архивных файлов пакетов (например, каталог /var/db/pkg/scr2png-1.1 соответствует пакету scr2png-1.1.tgz).

Каждый из таких файлов содержит минимум три обязательных файла: +COMMENT, +CONTENTS, +DESC. Все это — простые текстовые файлы, которые можно просмотреть командой `less` или любым текстовым редактором. Следует только учесть, что команду `less` для просмотра такого файла в большинстве командных оболочек потребуется дать в форме

```
$less -- +COMMENT
```

например, где удвоенный символ дефиса предваряет аргумент — иначе последует сообщение об ошибке.

Содержимое первого из указанных файлов — краткая, в одну строку, характеристика пакета, файл +DESC содержит более развернутое описание назначения и возможностей программы; именно содержимое этих двух файлов выводится при исполнении программы pkg_info. В файле же +CONTENTS обнаруживаются:

- указание на префиксный каталог (/usr/local);
- список файлов пакета и подкаталогов для их размещения (с путями относительно префиксного каталога — bin/имя, man/man#/имя.#.gz и т. д.);
- список пакетов, от которых зависит данный пакет (если таковые имеются);
- разного рода дополнительная информация, используемая, в частности, при удалении пакета (см. ниже).

Кроме трех обязательных, подкаталог пакета может содержать и некоторые дополнительные файлы, например, файл +REQUIRED_BY, в котором представлен список пакетов, используемых данным. Список этот пополняется по мере установки пакетов, зависимых от рассматриваемого.

База данных установленных пакетов активно используется не только для выдачи информации о них, но и при удалении пакетов, буде таковая возникнет. Для удаления пакетов предназначена третья из рассматриваемых утилит — pkg_delete, в качестве аргумента которой используется полное имя пакета, например:

```
$pkg_delete scr2png-1.1.tgz
```

каковое и определяется по имени подкаталога в /var/db/pkg/. Удаление происходит весьма чисто — т. е. ему подвергаются все файлы, перечисленные в файле /var/db/pkg/package_name/+CONTENTS. Исключение — если при инсталляции пакета для него создавались какие-либо подкаталоги, автоматически они удалены не будут: для этого потребуется выполнить программу

pkg_delete с опцией -d (от *directory*). И, разумеется, если в удаляемой программе предусмотрены были какие-либо конфигурационные файлы в домашних каталогах пользователей, их нужно будет удалять вручную.

Пакет, от которого зависят другие пакеты из числа установленных в системе (т. е. в подкаталоге /var/db/pkg/package_name/ которого имеется файл +REQUIRED_BY), также не будет удален программой pkg_delete по умолчанию. Это можно сделать только принудительно, указав опцию -f (от *force*). И, разумеется, вряд ли зависимые пакеты после этого сохранят работоспособность.

Коллекция портов

Как можно понять из предыдущего раздела, установка дополнительных программ из коллекции пакетов — занятие весьма простое. Однако способ этот — лишь дополнительный к основному методу наращивания мощи FreeBSD, каковым является коллекция портов. Во-первых, далеко не все приложения существуют в виде пакетов для данной версии (особенно в момент ее появления). Во-вторых, собственно-то и сами пакеты вторичны по отношению к портам, генерируясь из последних (с помощью программы pkg_create). А главное — именно система портов определяет гибкость и наращиваемость FreeBSD, а также ее способность к актуализации базы доступных приложений.

Что такое система портов? В двух словах ответить не трудно: это некий минимальный набор файлов (мрачно именуемый "скелетом"), обеспечивающий получение исходных текстов программы, ее компиляцию и установку в FreeBSD. Впрочем, рассмотреть это проще на конкретном примере.

Коллекция портов (устанавливаемая при начальной инсталляции пакетов или в любой последующий момент времени программой /stand/sysinstall, как и любой иной компонент базового набора дистрибутива) размещается в каталоге /usr/ports. Каталог этот, в свою очередь, содержит множество подкаталогов, объединяющих (по тематическому принципу) отдельные программы, например: /usr/ports/editors (текстовые редакторы), /usr/ports/graphics (программы для работы с графикой) и т. д. Сами же эти программы образуют подкаталоги следующего уровня вложенности: /usr/ports/editors/joe (текстовый редактор Joe), /usr/ports/graphic/ImageMagick (набор утилит для работы с растровой графикой) и пр. Прошу обратить внимание, что, в отличие от имен пакетов, имя порта для конкретной программы в большинстве случаев не содержит указания на ее версию (хотя бывают и исключения) — глубокий смысл этого будет ясен из дальнейшего.

А уж подкаталог порта и содержит собственно файлы скелета. Минимальный их набор — pkg-descr, pkg-comment, pkg-plist, distinfo и, главное,

Makefile. Все это — простые текстовые файлы, которые могут быть просмотрены стандартными средствами (например, командой `less`). Назначение первых трех нетрудно понять по аналогии с системой пакетов. Файл pkg-comment — краткое (однострочное) описание программы, pkg-descr — ее более развернутое описание, pkg-plist — список файлов, входящих в портированную программу. Файл distinfo содержит контрольную сумму архивного файла, содержащего исходные тексты программы. Ну а Makefile — главный файл скелета, определяющий собственно условия установки программы.

Кроме того, в скелет порта могут входить (и, как правило, входят) т. н. патчи (*patches* — заплатки), описывающие, какие изменения должны быть внесены в исходные тексты программы для ее корректной компиляции, установки и работы именно в системе FreeBSD (поскольку большинство портированных программ были созданы не специально для этой ОС, а для некоей абстрактной UNIX-системы). Они собраны в подкаталоге /usr/ports/group_name/port_name/files и имеют вид patch.aa, patch.ab и т. д. Это также обычные текстовые файлы, содержание их традиционно — указания типа "Удалить строку 10" или "Заменить строку 26 на такую..." в файле имя_рек. Они генерируются программой diff, почему и называются иначе diff-файлами.

Как же происходит установка программы из коллекции портов? При наличии постоянного подключения к Сети — проще трудно представить. Достаточно, обладая правами администратора, перейти в каталог порта требуемой программы

```
$cd /usr/ports/group_name/port_name
```

и дать команду

```
$make; make install
```

и после некоторого времени (зависящего от скорости соединения, состава ранее установленных программ, объема устанавливаемого приложения) будет возвращено приглашение командной строки, означающее, что порт благополучно установлен.

Разумеется, прежде чем устанавливать порт программы, его необходимо отыскать в коллекции портов. Сделать это можно или стандартной командой `find`, или специально предназначенной для этого утилитой whereis:

```
$whereis имя_программы
```

где, в отличие от получения информации о пакете, номер версии указывать не нужно. Например, ответом на команду

```
$whereis scr2png
```

будет сообщение вида

```
/usr/ports/graphics/scr2png
```

из которого следует, что порт искомой программы расположен в подкаталоге graphics/ каталога /usr/ports/.

Еще одним способом поиска конкретного порта является использование встроенного средства из коллекции портов. Чтобы воспользоваться им, необходимо перейти в каталог /usr/ports и выполнить команду

```
$make search key=program_name
```

Преимуществом этого способа является то, что он указывает не только местонахождение каталога искомого порта, но предоставляет и дополнительную информацию, в частности, о зависимостях программы, которая может потребоваться при скачивании исходных текстов вручную (о чем я скажу немного позже).

Вместо имени программы можно указать и какое-либо слово, связанное с ее функциональностью, и которое с большой долей вероятности может оказаться ключевым. Например, команда

```
$make search key=mpeg
```

выведет список всех портов, имеющих отношение к mpeg-файлам (включая и средства разработки). Если требуется получить только программы для работы с mpeg-звуком или mpeg-видео, предварительно следует перейти в каталог /usr/ports/audio или /usr/ports/graphics соответственно.

Как собственно устанавливается порт — с точки зрения пользователя, остается за кадром. Однако мы все же рассмотрим этот чрезвычайно интересный процесс.

Перво-наперво, система установит соединение с FTP-узлом, на котором имеются исходные тексты искомой программы. Как правило, это FTP-сервер ее разработчика и его более или менее официальные зеркала. Адреса FTP-серверов задаются в файле Makefile строкой вида

```
MASTER_SITES=FTP://FTP.имя_узла/каталог
```

Для большинства общеупотребимых программ из коллекции портов здесь приводится достаточно большое количество зеркал (впрочем, при необходимости адрес FTP-узла можно задать и вручную, как это будет показано ниже). Далее в указанном каталоге отыскивается архив с исходными текстами требуемой программы. Что именно следует искать — описывается в том же Makefile строками вида

```
PORTNAME=
PORTVERSION=
PORTREVISION=
...
DISTNAME=       ${PORTNAME}${PORTVERSION}
EXTRACT_SUFX=
```

где PORTNAME — имя порта (например, joe), следующие две строки — номера версии и, если требуется, ревизии программы (для приводимого примера редактора joe текущими на момент написания будут значения 2.8 и 3, соответственно). Строка DISTNAME описывает модель образования имени архива (в приводимом примере — joe2.8), а в строке EXTRACT_SUFX дается расширение, определяющее тип архива (tgz, tar.gz, tar.Z и т. д.).

Успешно обнаруженный на FTP-сервере файл (в нашем примере это будет joe2.8.tar.Z) скачивается на локальную машину и помещается в каталог /usr/ports/distfiles. Попутно будут скачаны нужные патчи и все остальное необходимое для сборки, а также обязательно проверены контрольные суммы для скачаных файлов, что а) установит ошибку при передаче, если таковая имела место, и б) не позволит (к примеру) подменить файлы на сервере. Вслед за этим архив распаковывается и исходные тексты программы помещаются во вновь созданный подкаталог work каталога порта (для описываемого случая — /usr/ports/editors/joe/work/joe).

После этого начинается собственно сборка программы из исходных текстов (исполнение команды make), завершаемое размещением ее компонентов по требуемым каталогам (исполнение команды make install) — /usr/local/bin, /usr/local/lib и т. д. К слову сказать, вместо приведенной выше командной конструкции можно ограничиться второй ее составляющей —

```
$make install
```

Результат будет идентичен. В процессе сборки, во-первых, на файлы исходных текстов накладываются все необходимые патчи, содержащиеся в подкаталоге files каталога порта (строго говоря, наложение патчей, разумеется, предшествует процессу сборки).

Во-вторых, если при сборке программы обнаруживается ее зависимость от каких-либо других компонентов, отсутствующих в системе (например, библиотек), компиляция приостанавливается, соответствующая программа отыскивается в коллекции портов и устанавливается тем же порядком, что и основная — процесс этот может быть многоступенчатым, вплоть до полного удовлетворения взаимозависимостей.

Завершающий этап установки — регистрация вновь установленной программы в базе данных /var/db/pkg, точно так же, как это делается в отношении пакетов.

Основной сложностью при установке программ из портов может быть процесс скачивания исходных текстов. Как уже говорилось, в скелете порта в качестве мастер-сайта обычно указывается узел разработчика программы, который может быть труднодоступным из России. В то же время практически все (за исключением тех приложений, для которых такое обращение запрещено их лицензией — но их очень мало) исходные тексты портированных программ имеются на FTP-сервере проекта FreeBSD. Сервер же этот

имеет многочисленные зеркала по всему миру, в том числе и базирующиеся в России. И потому в качестве источника целесообразно использовать один из них.

Это вполне возможно: достаточно запустить команду `make` с явным указанием FTP-адреса требуемого сервера, например:

```
make MASTER_SITE_OVERRIDE= \
FTP://FTP2.ru.FreeBSD.org/pub/FreeBSD/distfiles/ fetch
```

и исходные тексты будут скачиваться с указанного сервера (по моему опыту — одного из самых быстрых российских зеркал) вместо тех, которые прописаны в файле `Makefile`.

Чтобы осознать красоту и удобство системы портов, достаточно один раз ею воспользоваться. Однако возникает резонный вопрос — как этой красотой пользоваться при отсутствии постоянного подключения к Интернету? Возможно и это: все, что требуется — предварительно скачать исходные тексты требуемой программы и разместить ее в каталоге /usr/ports/distfiles, специально для этой цели предназначенном.

Скачивать можно с любого FTP-узла из перечисленных в Makefile или с FTP-серверов проекта FreeBSD. Причем второй источник — предпочтительней. Дело в том, что, хотя обычно архивы с исходниками размещены непосредственно в корне каталога /usr/ports/distfiles, для некоторых программ в нем выделены отдельные подкаталоги. И их структура должна повторять таковую каталога на FTP-сервере.

В качестве примера можно привести текстовый редактор Vim. Под архивы с исходниками различных его версий на FTP-сервере отведен каталог /pub/FreeBSD/distfiles/vi, и именно в его аналог (/usr/ports/distfiles/vi) следует поместить их на локальной машине.

Второе, что следует иметь в виду при установке из портов в режиме off-line — зависимости программ. Исходные тексты всех компонентов, требуемых для сборки данной, также должны быть скачаны заблаговременно и находиться в каталоге /usr/ports/distfiles/. Возникает вопрос, как их выявить? Тут-то и приходит на помощь упомянутое ранее средство поиска —

```
make search key=program_name
```

Например, если мы ищем порт программы scr2png, ответом на указанную команду будет сообщение вида

```
Port:  scr2png-1.1
Path:  /usr/ports/graphics/scr2png
Info:  Converts the output of "vidcontrol -p" to PNG
Maint:  nik@freebsd.org
Index:  graphics
```

```
B-deps:  freetype2-2.0.6 gd-1.8.4_5 jpeg-6b_1 png-1.2.1
R-deps:  freetype2-2.0.6 gd-1.8.4_5 jpeg-6b_1 png-1.2.1
```

из которого нетрудно понять, что для установки ее требуется наличие в системе программ freetype2-2.0.6, gd-1.8.4_5, jpeg-6b_1, png-1.2, получением исходных текстов которых и следует озаботиться (впрочем, они могут быть установлены и из пакетов).

Правда, команда `make search` не всегда обеспечивает полный учет всех зависимостей пакета. Однако для этого есть и другое средство — команда

```
make fetch-recursive-list
```

Впрочем, она выведет список всех требуемых для данной программы компонентов вообще, вне зависимости от того, установлены они в системе или нет, так что разбираться с ним, возможно, придется долго.

Установка программ из коллекции портов требует немало свободного дискового пространства, т. к., кроме собственно программных компонентов, на диск записываются также файлы архивов исходных текстов и результаты их распаковки. Однако по завершении установки хранить все это хозяйство отнюдь не обязательно (хотя и желательно — на случай переустановки или перекомпиляции).

Конечно, все промежуточные компоненты можно удалить вручную, командой `rm`. Однако, поскольку они разбросаны по разным подкаталогам, это займет много времени (и чревато ошибками — можно случайно удалить и что-нибудь нужное). И потому можно воспользоваться средствами автоматизации, предлагаемыми самой системой портов.

Так, для удаления исходных текстов программ, развернутых из архивов, достаточно перейти в каталог /usr/ports и дать команду

```
$make clean
```

которая метлой пройдется по всем нижележащим подкаталогам и удалит все, кроме скелетов каждого порта. Правда, следует учесть, что процесс этот — весьма длительный, и, вне зависимости от мощности машины, может занять не один час — сканироваться будут абсолютно все подкаталоги /usr/ports, вне зависимости от того, если ли там что-либо кроме скелетов, или нет.

А для удаления собственно архивов можно прибегнуть к команде

```
$make distclean
```

Хотя, повторяю, к последней процедуре я обратился бы только в том случае, если архив легко скачать заново (т. е. при очень хорошем соединении) и уж очень большом дефиците дискового пространства.

Не трудно удалить и установленную из коллекции портов программу. Как уже упоминалось, в результате установки порт программы регистрируется в

базе данных как обычный пакет (packages). И может быть удален тем же способом, что и последний. То есть — переходим в каталог /var/db/pkg, смотрим точное название пакета (совпадающее с соответствующим подкаталогом) и даем команду

```
$pkg_delete имя_пакета
```

Есть и альтернативный способ — переход в каталог /usr/ports/groun_name/port_name и команда

```
$make deinstall
```

что приведет к тому же результату.

Вообще говоря, команда `make` в системе портов FreeBSD обеспечивает множество функций. Так, с ее помощью можно за раз установить все существующие порты. Для этого следует только остановиться в каталоге /usr/ports и дать команду `make install`. Аналогично, дав эту команду, находясь в текущем каталоге /usr/ports/editors, можно установить все портированные текстовые редакторы, в каталоге /usr/ports/shells — все командные оболочки, и т. д.

Порты можно устанавливать и в два приема. Сначала, командой

```
$make -DBATCH install
```

те из них, которые не требуют интерактивного взаимодействия с пользователем (например, в ночное время). А затем, с помощью команды

```
$make -DIS_INTERACTIVE install
```

остальные, установка которых требует ответов на кое-какие вопросы. С прочими, весьма многочисленными, возможностями системы портов можно ознакомиться посредством официальной документации проекта FreeBSD.

Установка непортированных программ

Коллекция портов охватывает почти все многообразие открытых и свободных UNIX-приложений. Почти, но не всех: есть вероятность, что жизненно необходимая программа по сию пору не портирована в FreeBSD, хотя и распространяется в исходных текстах, предназначенных для компиляции в абстрактной UNIX-системе. Что делать в этом случае?

Во-первых, можно, конечно, попробовать самому сделать порт такой программы, благо процесс этот очень хорошо документирован в Руководстве FreeBSD по созданию портов. Однако занятие это требует определенных навыков и вряд ли придется по вкусу конечному пользователю.

И потому проще попробовать просто скомпилировать такую программу. Как — однозначный рецепт дать сложно, это зависит от того, какие средства

установки были предусмотрены разработчиком. Однако несколько общих приемов вполне можно указать.

Как правило, исходные тексты любой открытой и свободной программы распространяются в виде компрессированного tar-архива (файла с расширением tar.gz, tgz, tar.bz2 и т. д.). Так что первый шаг — это получение и распаковка такого архива. Место распаковки обычно значения не имеет, это может быть домашний каталог пользователя.

Далее, следует внимательно ознакомиться с файлами документации, сопровождающими программу (обычно они носят имена README, INSTALL или что-нибудь в этом роде). Теоретически (а, как правило, и практически) в них содержатся более или менее внятные описания порядка сборки и установки программы.

В ряде простых случаев достаточно, находясь в каталоге исходных текстов, дать команду `make` или `make all`, а затем — `make install` (последняя обычно требует привилегий root-оператора). Иногда эти действия должны предваряться запуском конфигурационного скрипта (файл вида configure).

Возможно, что интересующая программа не предусматривает средств автоматической (т. е. посредством команды `make install`) установки. В этом случае возникшие после компиляции (исполнения команды `make`) бинарные файлы следует вручную переместить туда, куда нужно (например, исполнимый файл — в каталог /usr/local/bin или $HOME/bin).

В качестве примера опишу установку двух полезных утилит — fookb, переключателя (он же — индикатор) раскладки клавиатуры для оконной системы X, написанной Алексеем Выскубовым, и rusconv, весьма функционального и удобного перекодировщика для кириллических текстов (автор — Олег Паращенко).

Для установки fookb (ее можно обнаружить на **http://triton.itep.ru/pub/linux/itepware/X11/tools/** или на моем сайте) требуется, после распаковки архива (fookb.tar.gz), открыть в текстовом редакторе файл Makefile и внести изменения в строки

```
FLAGS=DWMAKER
```

и

```
FLAGS=
```

(именно так, справа от знака равенства не должно быть ничего). Если предполагается использование fookb с оконным менеджером WindowMaker (для которого эта программа изначально и была предназначена), знак комментария должен быть на второй строке, если с иным оконным менеджером, — на первой. После этого все файлы с расширением xpm (их всего пять, это собственно изображения, индицирующие раскладку клавиатуры) копируются в каталог /usr/local/share/fookb (естественно, сначала его нужно создать).

Ну, а затем программа компилируется (просто командой `make`) и ее исполнимый файл (который так и называется — fookb) перемещается из текущего каталога в каталог /usr/local/bin. Разумеется, копирование файлов в оба каталога возможно только с полномочиями администратора.

Программа rusconv может быть найдена на **http://uucode.com/rusconv/index.html**. Для компиляции ее достаточно распаковать архив (rusconv.tar.gz), зайти в каталог rusconv.3.11/src/ и дать команду `make all`, а затем — `make install` (уже от лица суперпользователя).

Многие программы для UNIX, распространяемые в исходных текстах, имеют в своем составе специальный сценарий configure, предназначенный для приведения их в соответствие с реальной ситуацией на данной машине. В этом случае процесс сборки и установки должен предваряться его запуском из каталога с исходными текстами:

```
$cd ~/prog_dir
$./configure
```

Теоретически рассуждая, после этого компиляция программы должна пройти без проблем. Но все устанавливаемые мной под FreeBSD непортированные программы такого сценария не имели, и потому оценить его эффективность не могу.

Нужно заметить, что оба приведенных мной примера относятся к категории элементарных — эти программы не используют каких-либо специфических библиотек. В общем случае сборка непортированной программы может оказаться занятием нетривиальным, так что, возможно, проще будет дождаться появления соответствующего порта. Или, в случае специфических Linux-программ, использовать средства их запуска под управлением FreeBSD.

Установка оконной системы X

Туз — он и в Африке туз.

Присказка преферансистов

Как известно, оконная система X (X Window System) представляет собой графический интерфейс (вернее, своего рода мета-интерфейс) для большинства UNIX-систем, в том числе и тех, которые стыдливо именуются UNIX-подобными. В частности, свободная ее модификация, XFree86, функционирует над любыми клонами UNIX для платформ с процессорами Intel и совместимыми. И потому бытует мнение, что X, подобно тузу, — он и в Африке X. И установка, и настройка этой системы больше зависит от версии ее, нежели от собственно операционной системы, поверх которой она инсталлируется.

Определенный резон в этом есть — структура конфигурационного файла этой системы, XF86Config, идентична для любых дистрибутивов Linux, Free- или OpenBSD. А формат описания опций определяется тем, имеем ли мы дело в данном случае с представителями 3-й ветви (последняя из здравствующих особей — 3.3.6), или же 4-ой (текущий экземпляр — 4.2.xx). Однако в ряде случаев детали установки и настройки XFree86 разнятся не только от дистрибутива к дистрибутиву Linux (а все дистрибутивы Linux — не более чем внешнее выражение одной и той же операционной системы), но даже и между версиями одного дистрибутива — причем вне зависимости от версии XFree86.

Так что ожидать, что XFree86 будет вести себя идентично в разных операционных системах — было бы опрометчиво. Что и оправдывает обращение к процессу настройки системы X Window в FreeBSD текущих (4.x) версий. Тем более, что это — первые релизы этой ОС, в который "иксы" входят в обоих (3-й и 4-й) своих ипостасях.

Начать с того, что в базовый комплект (т. е. в **Distributions**) включена XFree86 версии 3.3.6. Она устанавливается и настраивается через sysinstall обычным, многократно описанным, образом — ничего нового по этому поводу я сказать не смогу. Почему и перейду сразу к версии 4-й.

В дистрибутивном комплекте она доступна (в виде XFree86-4.1.xx) только как дополнительный пакет (package, хотя имеется, конечно, и в коллекции портов). И может быть установлена как через sysinstall, так и обычным способом добавления пакетов посредством pkg_add. После чего в некоторых версиях FreeBSD может появиться сообщение, что в целях безопасности исполнимому файлу не будут присвоены биты SUID и GUID. Биты эти, присвоенные исполнимому файлу, дают возможность порожденному им процессу получить права доступа не пользователя, запустившего процесс, как обычно, но владельца файла и его группы (о чем будет говориться в одной из ближайших глав). Каковыми в данном случае выступают, естественно, root и группа wheel. И потому запустить (даже после соответствующей настройки, о которой будет сказано чуть позднее) систему X Window обычным способом, командой `startx`, не удастся — система предлагает воспользоваться программой графической авторизации `xdm`.

Конечно, запустить систему X Window можно и командой `startx` (или `xinit`), но только — от лица root, что не есть хорошо. Что же касается `xdm` — мне этот способ активно не нравится. Возможно, он и оправдан при доступе к удаленной машине, но на локальном компьютере — попросту неудобен.

Для решения этой проблемы можно, не мудрствуя лукаво, просто присвоить (командой `chmod`) соответствующие атрибуты файлу XFree86 — не думаю, что на локальной машине это может представлять какую-нибудь угрозу для ее безопасности.

Однако, как я неоднократно замечал ранее, устанавливать систему X из штатного комплекта кажется не очень целесообразным. Лучше обратиться к сайту проекта XFree86 (**http://www.xfree86.org**), где можно получить заведомо более свежую ее версию (на момент создания этого текста — 4.2.x). Она доступна в виде исходных текстов и бинарных пакетов, скомпилированных как для основных дистрибутивов Linux, так и для FreeBSD.

Исходники XFree86 — это три основных архивных файла (X42xscr-1.tgz, X42xscr-2.tgz, X42xscr-3.tgz) и несколько дополнительных (документация, утилиты, контрольные суммы и т. д.), общим объемом около 60 Мбайт. Бинарные же пакеты для FreeBSD, при близком объеме (около 66 Мбайт), включают в себя два десятка архивов tgz (из которых абсолютно необходимы штук пять, но и прочие — ни в коем случае не лишни, включая, например, поддержку кириллицы), документацию по установке и использованию, контрольные суммы, а главное — инсталляционные сценарии.

На установке XFree86 из исходных текстов я останавливаться не буду — знающие, как это делать, в моих советах не нуждаются, прочим же предлагается произвести это на свой страх и риск (каюсь, я сам — не проделывал). Впрочем, мой совет — все же воспользоваться бинарными пакетами, установка коих ни малейшей сложности не содержит.

А именно: достаточно скачать соответствующие tgz-пакеты и два установочных скрипта (Xinstall.bin и Xinstall.sh), поместить их в произвольный каталог (например, собственный пользовательский) и, вооружившись привилегиями root-оператора, запустить из командной строки второй из указанных скриптов. Далее в интерактивном режиме последует серия вопросов, ответы на которые либо очевидны (например, а устанавливать ли "иксы" — как они думают, ради чего мы все это затеяли?), либо даются, исходя из собственных потребностей (об установке серверов печати, шрифтов и т. д.). В любом случае настоятельно советую установить сервер шрифтов (FontServer): он, помимо всего прочего, позволяет использовать в X-системе шрифты TrueType. Ну, а в конце — сообщение об успешной инсталляции.

Впрочем, успех инсталляции не означает, что можно немедленно запустить графический режим. Прежде всего необходимо заняться настройкой системы X Window. Впрочем, это относится и к установке из штатных пакетов дистрибутива FreeBSD: полуавтоматическое конфигурирование системы X Window, доступное из /stand/sysinstall, применимо только для версии 3.3.6. Для 4-х версий к нему лучше и не прибегать — все равно потребуется ручная доводка.

Для конфигурирования 4-й версии XFree86 в их комплекте присутствуют два средства: графическая утилита xf86cfg и традиционный текстовый xf86config (графическая программа настройки XF86Setup работает только с версией 3.x.x и, если последняя не устанавливалась, просто отсутствует в системе). Первое средство приводит к немедленному и мертвому зависанию,

требующему нажатия Reset или сетевого выключателя, со всеми вытекающими последствиями. К слову сказать, та же история наблюдается и в OpenBSD, и в Linux — так что пока о графическом конфигурировании 4-й версии лучше забыть.

Так что, если нет большого желания писать конфигурационный файл оконной системы X — etc/X11/XF86Config, — вручную с нуля, остается прибегнуть к мрачной утилите xf86config. Благо она, несмотря на все неудобства (особенно раздражает, что при любой ошибке весь процесс настройки приходится начинать сначала) служит верой и правдой многим поколениям X-пользователей.

Запуск программы настройки выполняется одноименной командой

```
$xf86config
```

обязательно — от имени администратора. После чего следует экран полезных советов (типа того, что не худо бы знать характеристики своей видеокарты) с предложением нажать клавишу <Enter> для начала конфигурирования. Каковое начинается в выбора типа мыши. Возможны варианты (здесь и далее я описываю опции `xf86config` из поставки проекта XFree86, в дистрибутиве FreeBSD они несколько беднее):

```
1  Microsoft compatible (2-button protocol)
2  Mouse Systems (3-button protocol)
3  Bus Mouse
4  PS/2 Mouse
5  Logitech Mouse (serial, old type, Logitech protocol)
6  Logitech MouseMan (Microsoft compatible)
7  MM Series
8  MM HitTablet
9  Microsoft IntelliMouse
```

Большинство опций из этого списка представляют исторический интерес. С другой стороны, в нем отсутствуют мыши для USB-шины, работа с ними в дальнейшем потребует ручной доводки (и это — не особенность FreeBSD, а недоработка "иксов" текущего сезона).

Далее следует вопрос об эмуляции средней кнопки (нажатием двух остальных). Положительно на него следует отвечать только при наличии простой двухкнопочной мыши: в моделях со скроллирующим колесиком (а без оного нынче мышей почти и не выпускается) оно выполняет функции средней клавиши вполне справно.

Следующий вопрос — об имени файла устройства, используемого для мыши. Тут следует быть внимательным — предлагаемый по умолчанию вариант (/dev/mouse) в FreeBSD не работает ни с PS/2-моделями, ни с мышами для USB-шины, его следует заменить на /dev/sysmouse.

Расправившись с мышью, приступаем к клавиатуре: на выбор предлагается список из 25 пунктов, приводить который полагаю излишним. Тем более что практический смысл нынче имеет, скорее всего, только пункт 4-й —

```
4  Generic 105-key (Intl) PC
```

Конечно, для современных мультимедийных клавиатур можно попробовать и

```
15  Logitech iTouch
```

или что-нибудь еще, соответствующее по названию, но за результат — не ручаюсь. Дальше — интересней, есть возможность выбора национальных клавиатур. В числе коих —

```
52  Russian
53  Russian (cyrillic phonetic)
```

Выбор, обычно, следует остановить на пункте 52; русская фонетическая раскладка (т. н. *YWERTY*) нынче в обыденной жизни почти не встречается. После чего (и это — новшество конфигуратора из 4-й версии оконной системы X) можно настроить множество дополнительных языково-зависимых клавиатурных опций. Первая из них — вариант раскладки клавиатуры (**variant**). Если просто нажать <Enter>, он будет "умолчальным", т. е. совпадающим с DOS-маркировкой клавиш для кириллицы (не путать с DOS-раскладкой клавиатуры как таковой), ныне в продаже почти отсутствующей. Так что не ошибусь, если предложу руками набрать здесь имя требуемого варианта — `winkeys`, т. е. установить Windows-маркировку клавиш. Повторяю, к Windows-кодировке раскладки это не имеет никакого отношения, а означает только, что знаки препинания и прочие служебные символы в кириллической раскладке будут именно там, где они обозначены на клавиатуре фабричным способом.

Далее, очень рекомендуется положительный ответ на следующий вопрос:

```
Please answer the following question with either 'y' or 'n'.
Do you want to select additional XKB options (group switcher,
group indicator, etc.)?
```

Это прежде всего позволит выбрать переключатель раскладки клавиатуры (в нашем случае — латиница/кириллица) из следующих:

```
1  R-Alt switches group while pressed
2  Left Win-key switches group while pressed
3  Right Win-key switches group while pressed
4  Both Win-keys switch group while pressed
5  Right Alt key changes group
6  Caps Lock key changes group
7  Menu key changes group
```

```
8  Left Win-key changes group
9  Right Win-key changes group
10 Both Shift keys together change group
11 Control+Shift changes group
12 Alt+Control changes group
13 Alt+Shift changes group
```

Можно видеть, что количество переключателей — более чем достаточно, и можно выбрать на свой вкус. Следует только учесть, что в консоли FreeBSD — переключатель по умолчанию — клавиша <Caps Lock>, и если в "иксах" избрать иной, то для единообразия придется править терминальную раскладку клавиатуры. Впрочем, процесс этот не сложен, и будет подробно описан в соответствующей главе. Хотя лично для меня исторически традиционный для UNIX-систем переключатель (<Caps Lock>) представляется очень удобным. В один ряд с ним могу поставить только задействование "паразитных" Windows-клавиш...

Далее (и это опять отрадное усовершенствование текущей версии X-конфигуратора) можно выбрать и переключатель третьего уровня для работы с более чем двумя раскладками клавиатуры. В наших условиях это очень не вредно, если предполагается работать не только с латиницей и кириллицей в кодировке KOI8, но и, например, с кодовой страницей Windows (CP1251).

Следующие штрихи настройки "иксовой" клавиатуры — выбор дополнительной клавиши для <Ctrl> (в чем я лично необходимости не ощущаю), и индикатора переключения клавиатурных групп (тех же латиницы/кириллицы, например):

```
1  Num_Lock LED shows alternative group
2  Caps_Lock LED shows alternative group
3  Scroll_Lock LED shows alternative group
```

Конечно, для системы X Window существует несколько в разной степени удобных программ-индикаторов (одна из них, fookb, упоминалась в предыдущем разделе), однако аппаратный индикатор может быть удобен. Тем более, что в консоли в качестве такового выступает индикатор <Caps Lock>.

Далее в клавиатурных настройках можно выбрать режим капитализации символов и его прекращение (разными сочетаниями клавиш <Caps Lock> и <Shift>), установить стандартные функции для Windows-клавиш (<Menu> и <⊞> — я от этого отказываюсь, предпочитая использовать их по собственному усмотрению, для переключения виртуальных консолей), после чего происходит переход к настройкам видеосистемы.

Этот момент требует внимательного отношения (хотя истории о сгоревших от неправильной настройки мониторах нынче ушли в область легенд и ми-

фов Древней Греции). Прежде всего, следует выбрать диапазон частот горизонтальной развертки монитора из фиксированных значений

```
1  31.5; Standard VGA, 640x480 @ 60 Hz
2  31.5 - 35.1; Super VGA, 800x600 @ 56 Hz
3  31.5, 35.5; 8514 Compatible, 1024x768 @ 87 Hz interlaced (no 800x600)
4  31.5, 35.15, 35.5; Super VGA, 1024x768 @ 87 Hz interlaced, 800x600 @
56 Hz
5  31.5 - 37.9; Extended Super VGA, 800x600 @ 60 Hz, 640x480 @ 72 Hz
6  31.5 - 48.5; Non-Interlaced SVGA, 1024x768 @ 60 Hz, 800x600 @ 72 Hz
7  31.5 - 57.0; High Frequency SVGA, 1024x768 @ 70 Hz
8  31.5 - 64.3; Monitor that can do 1280x1024 @ 60 Hz
9  31.5 - 79.0; Monitor that can do 1280x1024 @ 74 Hz
10  31.5 - 82.0; Monitor that can do 1280x1024 @ 76 Hz
```

или задать таковые вручную

```
11  Enter your own horizontal sync range
```

чему (при знании соответствующих характеристик или наличии документации на монитор должно отдать предпочтение). Затем то же проделывается в отношении диапазона вертикальной развертки:

```
1  50-70
2  50-90
3  50-100
4  40-150
5  Enter your own vertical sync range
```

После чего вводится некий идентификатор монитора — в принципе, абсолютно любой, но желательно с неким мнемоническим смыслом, например, просто название конкретной модели.

Следующий шаг, на протяжении веков вызывавший священный трепет X-пользователей, — выбор видеокарты. Ныне прежнего почтения к нему уже нет — современные X-серверы поддерживают все распространенные (в том числе — и новейшие) модели видеокарт. Напротив, в версиях 4.0.x—4.1.x снята поддержка некоторых архаичных карт (типа S3 Trio). Что и является причиной "умолчальной" установки системы X Window версии 3.3.6 в FreeBSD. Ведь ОС эта — в существенной мере серверная, а на серверы здравомыслящие люди ставят обычно любые подручные видеокарты — fps для Quake там вряд ли важен.

Тем не менее, в версии XFree86 4.2 решено вернуться к поддержке старых видеокарт, так что проблем в этом отношении не предвидится. И действительно, нужно очень постараться, чтобы не найти своего видеоадаптера в

списке из более чем полутысячи названий, предлагаемом современным xf86confog...

Так что, покончив с этим, указываем объем видеопамяти в килобайтах — также откровенный анахронизм, карты с ОЗУ менее 16 Мбайт нынче не найти днем с огнем. Затем присваиваем нашей карте идентификатор — по тем же принципам, что и монитору — под этими условными названиями наши устройства будут просто фигурировать в соответствующих секциях конфигурационного файла. После чего устанавливаем видеорежимы — разрешение и глубину цвета.

Для начала — наборы разрешений для каждой из практически имеющих значения глубины цвета — от 8 до 24 бит (или, для некоторых карт, не поддерживающих простой True Color — до 32 бит). Предлагаются варианты — от 640×480 до 1280×1024 в каждом диапазоне цветности. Соглашаться с этим ни в коем случае не надо — значение разрешения, указанное первым, будет принято по умолчанию при запуске "иксов". А в большинстве оконных менеджеров разрешение менее 1024×768 ничего, кроме слез, не вызовет.

Конечно, в системе X Window разрешения монитора переключаются, что называется, "на лету" — комбинациями клавиш <Alt>+<Ctrl>+<GrayPlus> (повышение разрешения) и <Alt>+<Ctrl>+<GrayMinux> (его снижение). Однако лучше для каждого уровня цветности исключить заведомо ненужные значения, а в качестве первого поставить наиболее используемое. Рискну предположить, что для 17-дюймовых моделей это будет 1024×768 или (для потомков Соколиного Глаза) 1280×1024, хотя и 1152×864 никем не возбраняется.

Далее выставляем глубину цвета по умолчанию:

```
1  1 bit (monochrome)
2  4 bits (16 colors)
3  8 bits (256 colors)
4  16 bits (65536 colors)
5  24 bits (16 million colors)
```

Тут никаких рекомендаций дать не могу, кому что лучше — объема видеопамяти нынче хватит на любой режим.

Это — уже финал. Следующий вопрос — записать ли соответствующие изменения в конфигурационный файл, — должен, конечно, получить положительный ответ. Если в процессе конфигурирования возникает подозрение (или уверенность), что что-нибудь сделано не так, единственный выход — оборвать процесс (через <Ctrl>+<C>) и начать все сначала. Так что тут уж плакать нечего — соглашаемся и оказываемся в командной строке. Готовой, между прочем, исполнить и команду `startx` для запуска сессии графического режима.

Наконец, после всех настроек, "иксы" благополучно запускаются с оконным менеджером по умолчанию (таковым здесь выступает fvwm2). Впрочем, никто не неволит им ограничиваться — из пакетов или портов можно установить дополнительно и WindowMaker (со множеством дополнительных "причиндалов"), и IceWM, и Blackbox, и Afterstep, и множество других. Присутствуют и интегрированные среды — KDE версии 2.2_3 и GNOME версии 1.4_3, и XFce — можно выбрать себе по вкусу. После чего создать с пользовательском каталоге файл $HOME/.xinitrc — для начала с единственной строкой

```
exec file_name
```

где в качестве имени будет фигурировать исполнимый файл, запускающий любимую графическую среду — startkde (для KDE), gnome-session (для GNOME), icewm (для IceWM), blackbox (для Blackbox), wmaker (для WindowMaker, впрочем, последний требует дополнительного конфигурирования) или любой другой.

Однако об оконных менеджерах — разговор особый, к нему мы вернемся в главе о приложениях графического режима. А пока осталось последнее — окончательная настройка мыши. Она требуется в двух случаях — при использовании USB-модели и при желании применить в "иксах" скроллирующее колесико, каковое присутствует на 9 из 10 современных мышей (и прижилось даже трекболах).

Настройка мыши в графическом режиме описывается соответствующей секцией файла /etc/X11/XF86Config (название этой секции меняется в разных версиях XFree86, в версии 4.xx это — "InputDevice"). После первичной настройки она имеет вид, подобный следующему:

```
Identifier    "Mouse1"
Driver        "mouse"
Option "Protocol"  "PS/2"
Option "Device"  "/dev/sysmouse"
```

Следует обратить внимание на то, что в качестве опции `Device` в системе X Window используется не имя файла реального устройства (например, /dev/psm0, /dev/usm0 etc.) и не предлагаемый конфигуратором по умолчанию /dev/mouse (как правило, символическая ссылка на один из реальных файлов), а именно виртуальный драйвер /dev/sysmouse. Если это не так — следует внести соответствующее изменение. Иначе попытка запустить оконную систему X с указанием файла реального устройства (или устройства /dev/nouse) вызовет крах X-сервера с сообщением о занятости этого устройства.

Кроме того, для USB-мыши потребуется внести изменение и в строку, описывающую протокол. Так как единственное ее значение, совместимое в этой шиной, — `Auto`.

Дополнительные действия потребуются для настройки работы колеса прокрутки. Это — вполне доступно. Правда, оно вызывает не чистый построчный скроллинг, как в Windows, а лишь эмулирует клавиши <PageUp> и <PageDown>, но и это добавляет удобства при навигации. Для работы скроллирующего колесика требуется, во-первых, чуть подстроить файл /etc/X11/XF86Config, добавив в него строку

```
Option "ZAxisMapping" "4 5"
```

которая именно и предписывает транслировать события от вращения колесика в нажатия виртуальных 4-й и 5-й кнопок мыши, вызывающих пролистывание. Кроме того, если в описании мыши присутствуют какие-либо указания на эмуляцию 3-ей кнопки, скажем, вида

```
Option "Emulate3Buttons"
```

они должны быть изъяты или закомментированы. Второе — требуется установить собственно программу-транслятор, именуемую imwheel, присутствующую как среди пакетов дистрибутива FreeBSD, так и в коллекции портов. Ну и, наконец, программа эта должна быть запущена — вручную или в каком-либо стартовом файле, например, через $HOME/.xinitrc строкой вроде

```
imwheel &
```

перед запуском стартового файла оконного менеджера.

* * *

На этом разговор об установке системы и ее первичном конфигурированим можно временно закончить. Разумеется, дальше придется обращаться к вопросам тонкой настройки как отдельных режимов работы, так и приложений. Однако в результате затраченных усилий мы получили вполне пригодную к использованию систему.

Правда, внутренний смысл многих действий остался не вполне ясным — я не зря просил пока поверить мне на слово. Однако для понимания этого смысла требуется углубиться в детали устройства FreeBSD, чему и будет посвящена следующая глава. Если же необходимости в знаниях этих деталей пока нет — можно обратиться непосредственно к *части II*, посвященной использованию FreeBSD.

Дополнительные источники

- Основной источник сведений об установке программ из пакетов и портов — соответствующие главы руководства пользователя (Handbook — **http://www.FreeBSD.org.ua/handbook/**). Кроме того, на сайте Ивана Пас-

каля (**http://www.tsu.ru/~pascal/**) очень подробно описан процесс доустановки компонентов дистрибутива и установки пакетов.

- Для получения представления об устройстве коллекции портов полезно ознакомиться с вводными разделами Руководства FreeBSD по созданию портов (рус. пер. — **http://www.FreeBSD.org.ua/faq/**).
- Наконец, в Руководстве разработчика FreeBSD (рус. пер. — **http://www.FreeBSD.org.ua/developers-handbook/**) приведены исчерпывающие сведения о работе команды `make` и Make-файлах FreeBSD. Следует только учесть, что два последних руководства отнюдь не рассчитаны на конечного пользователя, и их понимание потребует от него немалых усилий.

Глава 6

Ядро системы

Мы тайны эти с корнем вырвем у ядра,
И вволю выпьем джина из бутылки!

Владимир Высоцкий

В этой главе будут вкратце рассмотрены устройство и функции ядра. Автор не претендует на исчерпывающее изложение предмета: это — дело профессионалов в области Computer Scsiences. Основное внимание же будет сосредоточено на вопросе, важном для пользователя — теории и практике конфигурирования и сборки собственного ядра, отвечающего реалиям конкретной системы.

Ядро и его функции

Как нетрудно догадаться исходя из названия, ядро — основная часть любой операционной системы. Оно обеспечивает базовую ее функциональность — управление процессами, распределение памяти, доступ к файлам и устройствам.

Как известно, ядра UNIX-систем (да и прочих — тоже) бывают монолитными и модульными. В первых поддержку всех требуемых функций необходимо жестко встроить в ядро. В модульных же ядрах (самый известный, и к тому же чуть не единственный реально работающий, пример — операционная система QNX) оно включает в себя только самые необходимые базовые функции управления, а поддержка всех внешних устройств осуществляется специальными программами, подгружаемыми по мере необходимости. Конечно, это — идеализированная схема. И большинство ядер, которые принято относить к категории монолитных, в той или иной мере поддерживают концепцию подгружаемых модулей.

Здесь не место обсуждать достоинства и недостатки того и другого подходов. Однако факт остается фактом: ядро FreeBSD, в отличие от Linux (где модульный подход применяется весьма широко), более монолитное. Некоторое количество загружаемых модулей имеется и здесь, однако поддержка подавляющего большинства важных для пользователя устройств требует включения соответствующих опций непосредственно в ядро.

Ядро FreeBSD (и родственных ей систем линии BSD 4.4) обеспечивает четыре основных системных сервиса (по МакКасику с соавторами, см. заключительный раздел):

- запуск и управление процессами, системными и пользовательскими;
- доступ к файлам и устройствам через унифицированный интерфейс файловой системы;
- механизм межпроцессорных коммуникаций;
- функции управления, такие как запуск системы и распределение памяти.

Некоторые из этих функций будут в ограниченном объеме рассмотрены в этой и последующих главах. Для более глубокого изучения же устройства ядра следует обратиться к источникам, перечисленным в заключительном разделе главы.

Ядро функционирует в пространстве памяти, изолированном от всех прочих задач (именуемых прикладными или пользовательскими). Взаимодействие между ними осуществляется через интерфейс системных вызовов — набора услуг ядра, предоставляемых прикладным задачам. Среди системных вызовов — такие, как `fork` (запуск процесса) и `exec` (запуск пользовательской программы), `open`, `read` и `close` (открытие, чтение и закрытие файла соответственно), и т. д.

Пользователю не вредно знать о существовании системных вызовов. Однако непосредственно с ними он практически не сталкивается, и останавливаться на этом я не буду. Мы же поговорим преимущественно о том, что пользователь должен делать с ядром системы (кроме того, что использовать его, разумеется). А должен он это ядро настроить под свои возможности и потребности, т. е. — перекомпилировать (или, как еще говорят, регенерировать). И это, в сущности, первое его деяние после установки и начальной настройки FreeBSD. Почему? Ведь само слово "перекомпиляция" способно вогнать в дрожь пользователя без должной подготовки. Тем не менее, тому можно назвать как минимум три причины.

Три и одна причина для регенерации ядра

Первая причина — в том, что деяние это все равно неизбежно: в отличие от Linux, где в user-ориентированных дистрибутивах вполне можно работать с ядром, собранным по умолчанию при установке, в FreeBSD это не то что бы нельзя — просто неэффективно (а ведь не ради идеальной адаптации системы мы за нее взялись?).

С одной стороны, FreeBSD изначально ориентирована на то, чтобы устанавливаться практически на любое "железо" с любых носителей, в том числе — и при удаленном доступе. То есть — в "умолчальное" (т. н. GENERIC) ядро включена поддержка почти всех мыслимых SCSI-адаптеров, сетевых

карт, сетевых протоколов и многого иного, что простому советскому человеку не понадобится никогда в этой пост-советской жизни.

С другой стороны, из тех же соображений совместимости (ну и по причине ориентированности на сисадмина-аскета), GENERIC-ядро не поддерживает такие тривиальные (и — жизненно необходимые зверю-юзеру) опции, как звуковые карты, различные видеорежимы консоли (за пределами стандартных 80-колоночных), оптимизацию под тип процессора и т. д., и т. п. Не говоря уж о том, что леди Free (как мы уже говорили, она ведь не добра, и не мила, но зато — не врет!) категорически отказывается брать на себя ответственность за прочее специфическое "железо" пользователя.

Так что процесс регенерации ядра — неизбежен, как распад мировой системы социализма. И потому резонно взяться за него сразу, руководствуясь принципом "раньше сядешь — раньше выпустят".

Вторая причина немедленной регенерации ядра, как ни странно, именно в потенциальной опасности этого процесса — приложив должное старание, в ходе его можно полностью развалить систему. Так не лучше ли заниматься этим с самого начала, на системе, не обросшей настройками, приложениями, данными? Ведь совсем уж "на худой конец есть вареные яйца" — т. е. просто переустановка.

От чего, конечно, — Господь борони: неисправимых ситуаций в FreeBSD не бывает. С помощью fixit-диска, команды `mount` и текстового редактора можно выйти из любого положения.

Третья причина если не пересобрать ядро, то, по крайней мере, изучить этот процесс — та, что знание опций конфигурации ядра потребуется практически при любой дальнейшей настройке. И на протяжении всего повествования время от времени придется к ним обращаться.

Есть, однако, и четвертая причина, достаточно тривиальная: чтобы научиться плавать, нужно лезть в воду. Так что регенерация ядра — самый быстрый способ если не завоевать сердце леди Free сразу, то, по крайней мере, снискать ее благосклонность.

О процессе

— Поручик, любите ли Вы детей?
— Детей — не очень, но процесс...

Из анекдотов про поручика Ржевского

Процесс регенерации ядра — лишь немногим менее увлекателен, чем тот, о котором говорил бравый поручик Ахтырского гусарского полка. Правда, и результат, если он достигнут, — также заслуживает любви. Однако сначала — о некоторых особенностях ядра FreeBSD вообще.

Процесс регенерации ядра FreeBSD, как и любой аналогичной системы, распадается на три этапа — получения исходников, конфигурирования и собственно компиляции и установки. С исходниками — проблем нет: если они не были установлены изначально, посредством /stand/sysinstall это можно сделать в любой момент (и почти мгновенно по времени) — они имеют место быть на первом инсталляционном CD (или — mini-диске). После чего в каталоге /usr/src появляется сложно структурированный подкаталог sys — это и есть полные исходные тексты ядра. При отсутствии диска исходные тексты следует скачать с FTP-сервера проекта — как уже говорилось, это файлы вида ssys.?? в каталоге /pub/release/src.

Конфигурирование опций ядра — центральный этап всей процедуры. В отличие от Linux, в котором имеется аж три специально предназначенных для этого средства (make config, сводящийся к ответу на вопросы, make menuconfig с текстовым меню и make xconfig — с меню графическим), в FreeBSD конфигурирование осуществляется методом лобовой атаки — прямой правкой одного из двух специально предназначенных к тому файлов. Просто и незатейливо — в текстовом редакторе.

И это, товарищи, правильно. По крайней мере, мне такой способ нравится больше. Особенно в сравнении с make config, где необходимость тупо отвечать на вопросы, часто не вполне удобопонятные, — просто удручает. Конечно, ориентированные на меню способы кажутся удобнее. Однако простое редактирование списка опций позволяет, окинув их одним взглядом, представить структуру конфигурации ядра во всей целостности, со всеми взаимосвязями и зависимостями между опциями. Впрочем, это мнение сугубо субъективное.

Файлов, принимаемых за основу для такой правки — два, GENERIC и LINT, расположенных в каталоге /usr/src/sys/i386/conf/, специально для них (и пользовательских файлов конфигурации) предназначенном. В первом файле описана конфигурация того самого ядра, которое было скомпилировано в процессе установки системы. Поскольку инсталляционная программа FreeBSD специально ориентирована на то, чтобы запускаться на практически любом распространенном оборудовании, он содержит множество ненужных опций — поддержку всевозможных сетевых и SCSI-адаптеров, например. И — лишен некоторого количества опций необходимых (по крайней мере, конкретному пользователю), либо отнесенных к категории роскошного, либо способных вызвать проблемы при установке. Однако, если машина после инсталляции системы запустилась — работоспособность ядра в имеющейся конфигурации гарантирована.

Напротив, LINT в первозданном своем виде по определению неработоспособен: он содержит все возможные опции конфигурации ядра, многие из которых несовместимы друг с другом. Однако он, практически, единственный источник сведений о допустимой конфигурации ядра, в том числе — и

благодаря довольно подробным комментариям. Впрочем, и комментариями в файле GENERIC пренебрегать не стоит, иногда в них обнаруживаются сведения, отсутствующие в LINT.

К слову — об источниках сведений. Существует много описаний процесса регенерации ядра, в том числе — и на русском языке (наилучшее из них — на сайте Ивана Паскаля, см. заключительный раздел главы). Однако оно (как, впрочем, и остальные мне известные) относится к довольно старым версиям FreeBSD — 2.x—3.x. И потому любые описания, не в обиду авторам будет сказано, требуют осторожного обращения: опции конфигурации ядра FreeBSD меняются от версии к версии, иногда (например, при переходе от ветки к ветке) довольно значительно. За сменой версий описания не поспевают, и потому требуется постоянная сверка их с LINT — комментарии в нем заведомо относятся к текущей версии. По крайней мере, мне пока разночтения не встретились.

Возникает проблема — какой из файлов конфигурации ядра принять за основу ядра собственного? Впрочем, истинный пользователь любой открытой системы отнесется к этой проблеме как к возможности выбора — какой файл хочу, такой и правлю.

Однозначный ответ дать трудно. При правке файла GENERIC потребуется исключать (для начала — закрывать комментариями) ненужные опции и перетаскивать из LINT недостающие. Это может быть утомительно — структура файлов абсолютно различна. Если в файле LINT опции более или менее, но все же упорядочены по назначению и разбиты на группы, то в GENERIC они расположены сугубо стохастически и лишены рубрикации.

Причина этого явления мне совершенно не понятна: казалось бы, своя рука — владыка, задай жестко структуру конфигурационного файла. Возможно, дело в том, что левая рука не знает, что творит правая: различия в стиле комментариев к файлу LINT явно показывают, что они составлялись разными авторами. А структурирование GENERIC (вернее, полное отсутствие такового), видимо, выполнялось третьими лицами.

При редактировании же файла LINT опции придется только вычеркивать, но зато — уж очень много: он содержит более двух тысяч строк. А формальный, так сказать, типографский, его объем превышает два печатных листа — более 90 Кбайт. Для сравнения — в типичном файле GENERIC менее 10 Кбайт и около 250 строк. Кроме того, без глубокого понимания устройства ядра гарантировать совместимость опций (и, как следствие, работоспособность конфигурации, созданной на основе LINT) довольно трудно. В случае же с GENERIC при должной аккуратности всегда можно хотя бы откатиться к рабочему варианту конфигурации.

И потому я предложил бы все же править файл GENERIC (вернее, разумеется, его переименованную копию). Начав при этом, в соответствии с указаниями председателя Мао, с упорядочивания стиля, т. е. приведения его

структуры в соответствие с таковой файла LINT. Занятие это довольно нудное, но, поверьте, сторицей окупится, не говоря уже о пользе на будущее — ведь наверняка это не последняя пересборка ядра.

Кроме того, по себе знаю, в процессе такого упорядочивания можно узнать много нового. Если же делать такую работу лениво — я ее проделал и готов поделиться результатом — **http://frebsd.by.ru/files/GENERIC_STRUCT.gz**, относящимся к версии 4.4.

Структура конфигурационного файла

Итак, в структуре файла LINT (и в упорядоченной структуре GENERIC) выделяются следующие секции:

- вводная, безымянная в оригинале, секция, где указываются архитектура компьютера, идентификатор ядра и некоторые общесистемные ограничения;
- секция SMP Options, интересная, естественно, только обладателям многопроцессорных машин;
- секция CPU Options — именно здесь можно оптимизировать ядро под конкретный процессор, а также включить, например, поддержку некоторых специальных инструкций;
- секция Compability Options, где устанавливается бинарная совместимость с некоторыми другими ОС;
- секция Debugging Options, содержащая разного рода отладочные опции;
- секция Networking Options, объединяющая опции поддержки сетевых протоколов и интерфейсов;
- секция Filesystems Options, позволяющая включить поддержку многочисленных файловых систем;
- секция POSIX P1003.1B, имеющая отношение к поддержке систем реального времени;
- секция SCSI Devices, описывающая конфигурацию устройств с одноименным интерфейсом; некоторые из драйверов этой секции необходимы для устройств совершенно другого типа;
- секция Miscellaneous Devices and Options, где собраны драйверы всякого рода псевдоустройств;
- секция Hardware device Configuration, содержание которой много шире названия;
- секция Undocumented options for linting, где, как ясно из названия, собраны всякие не вполне внятные опции для экспериментов на свой страх и риск.

Моей целью не является описание всех возможных опций конфигурации ядра — это задача, непосильная для одного человека. В следующих разделах я остановлюсь лишь на тех опциях, которые использую (или, напротив, не использую) в своих ядрах, смысл и взаимосвязь которых понимаю и совместимость (или, как раз наоборот, несовместимость) которых проверена на собственном опыте.

Обзор опций будет дан в порядке секций файла LINT. Некоторые из описанных ниже опций являются абсолютно обязательными, иные же — связаны между собой зависимостями, т. е. включение одной из такой группы требует обязательной установки другой (других). Все такие случаи я буду оговаривать явным образом (если имею соответствующие данные, разумеется). Опции, не сопровождающиеся такими явными оговорками, не являются обязательными. И могут быть безболезненно отключены (если, конечно, в них нет необходимости).

Мои комментарии ниже носят во многом предположительный характер, за что приношу свои извинения. Однако мне показалось бессмысленным с умным видом пересказывать декларативно комментарии, а проверить все опции самолично — никакой жизни не хватит...

Вводная секция

Здесь указываются архитектура компьютера, идентификатор ядра и некоторые общесистемные ограничения.

```
Machine  i386
```

Это — описание архитектуры машины, для которой собирается ядро, т. е. в данном случае — PC (как известно, FreeBSD существует также в версии для Alpha). Обязательная опция.

```
Ident  NEW
```

Это — просто некий идентификатор данной сборки ядра. Значение может быть любым, но быть — обязательно. И, хотя это и не оговорено специально, желательно — уникальным.

```
Maxusers  32
```

Также обязательная опция, теоретически ограничивающая максимально возможное количество пользователей, работающих на данной машине одновременно. Однако она влияет на количество одновременно запущенных процессов, открытых файлов, буферов памяти. И потому даже на индивидуальной машине для эффективного использования многозадачности есть смысл особо не жадничать с этим значением. "Умолчальная" его величина в файле LINT — 10, хотя ядро GENERIC в последних версиях (например, 4.5) собирается со значением 0.

```
Makeoptions  DEBUG=-g
```

При установке этой опции в код ядра включается некая отладочная информация, что влечет увеличение его размера чуть не на пол-порядка. Хотя, как говорят, это и не влияет на производительность.

```
#options  MAXDSIZ="(256*1024*1024)"
#options  MAXSSIZ="(256*1024*1024)"
#options  DFLDSIZ="(256*1024*1024)"
```

Три опции, отсутствующие в ядре GENERIC, включение которых позволяет преодолеть предел ОЗУ, отводимый в FreeBSD для запускаемых приложений. Я пока необходимости в этих опциях не ощутил, но представить себе ситуацию, в которой они понадобятся, — можно.

Далее в LINT приводится еще несколько опций, имеющих отношение к виртуальной памяти и кэш-памяти, общей перекомпиляции системы и прочим высоким материям. Но я про них ничего сказать не могу, отсылая заинтересованных к комментариям к файлу LINT.

Секция SMP Options

За отсутствием опыта работы с многопроцессорными машинами мало чего могу сказать по этому поводу. Очевидным представляется, что все эти опции в однопроцессорной конфигурации должны быть отключены (впрочем, именно это и сделано по умолчанию.

Секция CPU Options

Как уже было сказано, именно здесь можно оптимизировать процессор, а также включить, например, поддержку SSE/MMX2.

```
cpu  I386_CPU
cpu  I486_CPU
cpu  I586_CPU
cpu  I686_CPU
```

Эта группа опций оптимизирует ядро под процессор определенного поколения. Можно включить их все (тогда никакой оптимизации, как я полагаю, выполнено не будет), но лучше указать свой конкретный класс процессора (хотя в FreeBSD оптимизация под конкретный процессор и не дает такого прироста быстродействия, как в Linux). В любом случае, хоть одна опция из этой группы должна быть включена обязательно.

```
Options    MATH_EMULATE
```

Количество бумаги, изведенное на объяснение этой опции в ядре Linux, сравнимо только с возрастом (в миллисекундах) последнего "камня" без встроенного сопроцессора. Короче, о ней пора просто забыть.

```
Options    CPU_ENABLE_SSE
```

Это включение поддержки системы команд SSE/MMX2 для P-II/III и последних Athlon. То есть при наличии в машине одного из их представителей опция эта ни в коей мере помешать не может.

В дополнение к комментариям замечу, что для современных процессоров параметры оптимизации даются очень обобщенно — класс Pentium Pro и баста. Однако есть возможность более точно подогнать систему (причем — не только ядро, но и все приложения из базового набора и портов) под конкретный процессор: внесение соответствующей правки в файл /etc/defaults/make.conf, определяющий условия компиляции из исходников. Там можно указать конкретный тип процессора, например, Intel p4, p3, p2 или AMD Athlon — k7. Впрочем, дабы получить соответствующую оптимизацию, потребуется выполнить полную перекомпиляцию системы (включая собственно компиляторы и системные библиотеки).

И еще — множество опций, относящихся к процессорам типа AMD K-5 и K-6, Cyrix и IBM Bluelighthing, приведено далее в этой секции LINT. Куда и отсылаю счастливых владельцев этого богачества.

Секция Compability Options

Здесь объявляется бинарная совместимость с некоторыми другими ОС.

```
Options     COMPAT_43
```

Сим устанавливается совместимость с BSD 4.3 (не путать с FreeBSD версии 4.3). В комментарии сказано, что эту опцию отключать не следует.

```
Options     SYSVSHM
Options     SYSVMSG
Options     SYSVSEM
```

Эти три опции устанавливают поддержку интерфейсов в стиле System V, которые используются в системе X Windows. Если их отключить — при загрузке системы появляются предупреждающие сообщения (хотя оконная система X запускается благополучно). Так что, от греха подальше, лучше их не трогать (благо, они присутствуют в "умолчальной" конфигурации GENERIC).

```
Options     USER_LDT
```

Опция, требующаяся для использования Local Descriptor Table (LDT). В комментарии сказано, что это хозяйство используется многими портами, а в грядущих версиях FreeBSD потребуется и приложениям базовой системы. Так что лучше включить.

Здесь стоит добавить, что в современных версиях FreeBSD бинарная совместимость с Linux встроена в ядро по умолчанию и специальной опции не требует.

Секция Debugging Options

Это, как понятно из названия, — разного рода отладочные опции, большинство из которых пользователю вряд ли понадобится.

```
Options     UCONSOLE
```

Эта опция разрешает захват консоли обычными пользователями. Судя по комментарию, требуется для запуска ими системы X Window, так что лучше включить. Хотя и без нее "иксы", насколько я смог установить, запускаются без проблем.

В этой секции есть еще множество опций, требующихся для конфигурирования ядра в интерактивном режиме в момент загрузки. Смысл их подробно описан у Ивана Паскаля, хотя в тогдашних версиях они носили иные названия, но догадаться о соответствии можно.

Секция Networking Options

Объединяет опции, касающиеся сетевых протоколов и интерфейсов.

```
Options     INET
```

Эта опция включает поддержку протокола TCP/IP и необходима даже при отсутствии подключения к какой-либо сети. Протокол TCP/IP в обязательном порядке используется многими приложениями и при запуске на локальной машине. В частности, без него невозможна работа системы X Window. Так что, если не ограничиваться текстовым режимом, опция практически обязательна.

```
pseudo-device     loop
```

Поддержка псевдоустройства Network loopback.

```
pseudo-device     ether
```

Поддержка сетевых Ethernet-соединений.

Эти опции необходимы при наличии любого сетевого соединения, даже по параллельному кабелю или через сериальные порты. Из прочих, очень многочисленных, опций, имеющихся в этой секции файла LINT, я ограничиваюсь только включенными по умолчанию в GENERIC.

Секция Filesystems Options

Позволяет включить поддержку многочисленных файловых систем.

```
Options     FFS
```

Это — поддержка Fast File System, родной файловой системы FreeBSD. Разумеется, ни одна операционка не может работать без собственной файловой системы, так что эта опция обязательна.

```
Options     FFS_ROOT
```

Использование FFS как корневого устройства. Думается, что без этого, по понятным причинам, тоже не обойтись.

```
Options     SOFTUPDATES
```

Включение поддержки режима Soft Updates — затрудняюсь дать адекватный контексту русский перевод, но вещь хорошая, не только ускоряющая файловые операции, но и повышающая устойчивость файловой системы к сбоям. Короче говоря, опция если и необязательная, то крайне желательная — без нее можно заснуть при копировании большого количества мелких файлов. Благо, она включена в GENERIC по умолчанию, а активизация Soft Updates возможна еще на стадии установки системы.

```
Options     MSDOSFS
```

Поддержка файловой системы FATxx.

```
Options     CD9660
```

Поддержка файловой системы ISO-9660 — той, что на компакт-дисках; судя по всему, гуртом с расширениями Joliet (для Windows, с длинными именами файлов) и Rock Ridge (для UNIX, с символическими ссылками и атрибутами доступа).

```
Options     EXT2FS
```

Поддержка основной (до сего времени) файловой системы Linux. До сих пор имеет статус не вполне доделанной (experimental), но, по моему опыту, работает более чем исправно и на чтение, и на запись.

В LINT перечислено еще не мерянное количество опций в поддержку файловых систем, как локальных, так и сетевых — я привел лишь те, что сам использую. Причем, по мнению Ивана Паскаля, включать поддержку FAT и ISO-9660 в ядро не обязательно — можно обойтись подгружаемыми модулями для доступа к этим файловым системам.

А вот поддержка Ext2fs в виде модуля не реализована, и потому при потребности в этой файловой системе должна быть жестко встроена в ядро. Раз уж зашла об этом речь — после этого можно монтировать не только ext2fs-разделы, но и разделы с ее журналируемым вариантом — ext3fs. А вот в отношении второй журналируемой системы Linux, ReiserFS, этот номер не проходит.

Секция POSIX P1003.1B

Строки этой секции обеспечивают поддержку расширений для систем реального времени. Используется, насколько я знаю, только специальными программами, и потому вряд ли есть необходимость в их описании.

Секция SCSI Devices

Здесь описывается конфигурация одноименных устройств; некоторые из драйверов этой секции необходимы для устройств совершенно другого типа.

```
Device    scbus
```

Общая поддержка SCSI-устройств.

```
Device    da
```

Поддержка SCSI-устройств прямого доступа (типа накопителей любого рода).

О самих по себе SCSI-устройствах я ничего говорить не собирался за отсутствием опыта общения. Но эти две строки необходимы для поддержки таких накопителей, как Zip на параллельном порту и на шине USB (и некоторых других). Так что если есть желание таковые использовать — эти две строки обязательны. И вообще, в GENERIC (но почему-то не в LINT) первая из этих строк имеет пометку required.

Секция Miscellaneous Devices and Options

Здесь собраны всякого рода псевдоустройства.

```
pseudo-device    pty
```

Поддержка псевдотерминала для удаленного доступа, например, по Telnet. Однако и на локальной машине необходим для работы xterm в системе X Window, так что практически обязательная опция.

В прочих псевдоустройствах из этой секции (например, для запуска упакованных бинарных файлов) необходимости, как будто бы, нет.

Структура секции Hardware device Configuration

О недокументированных опциях я говорить не буду. Ну а секция Hardware device Configuration заслуживает отдельного рассмотрения — она самая большая и не то, чтобы самая важная (мамы всякие нужны, мамы всякие важны), — просто в ней проще всего допустить ошибку. Как по существу, так и в отношении зависимости опций.

В оригинальном файле LINT некоторая структурированность в этой секции устанавливается, но четкой рубрикации, скажем, нет. Поэтому я взял на себя смелость такую рубрикацию сделать, выделив ряд подсекций:

- общая поддержка устройств, таких как системные шины;
- клавиатура и мышь — драйверы соответствующих устройств и их опции;

- видеосистема — в сущности, VGA-драйвер и опции, описывающие его режимы;
- системная консоль — ее драйверы (syscons или pcvt) и многочисленные опции;
- накопители — диски, сменные носители (только с ATA-интерфейсом, SCSI я проигнорировал), дисководы;
- коммуникационные порты — последовательные и параллельные;
- устройства USB — именно они;
- прочие устройства — аудиоустройства PCI, управление питанием.

Я пытался придерживаться последовательности, установленной в LINT, и употребляемой в комментариях терминологии. Однако в ряде случаев от этого приходилось отказываться ради придания пущей логичности.

Разумеется, ниже в каждой субсекции перечислены далеко не все возможные опции из LINT. Кроме этого, в принципе следовало бы выделить субсекции для сетевых карт и аудиоустройств ISA, о которых говорится очень много. Но я это просто опустил — в первых мало чего понимаю, а вторые утратили актуальность.

Общая поддержка устройств

Здесь сгруппированы опции, имеющие отношение к поддержке шин ввода/вывода. Явную экзотику типа шины EISA я рассматривать не буду.

```
Device    isa
```

Эта строка должна быть обязательно, даже если самой шины (как это в обычае на современных материнских платах) и в помине нет. Многие опции, перечисленные в следующих субсекциях, завязаны на нее. В частности — все, относящееся к системной консоли.

```
Device    npx0 at nexus? port IO_NPX irq 13
```

Поддержка сопроцессора. Обязательная опция, даже в научно-фантастическом случае его отсутствия (и отключении эмуляции) — без нее ядро просто не соберется.

```
Device    pci
```

Поддержка шины PCI. Разумеется, обязательна на любой современной машине.

Клавиатура и мышь

Назначение подсекции ясно из названия — это опции, определяющие свойства системной консоли в части ввода.

```
Device    atkbdc0    at isa? port IO_KBD
Device    atkbd0    at atkbdc? irq 1 flags 0x1
```

Этими строками, в сущности, задаются драйверы клавиатуры — PS/2 (первая строка) и AT (вторая). Поскольку без клавиатуры работать сложновато, хоть одна из них должна быть обязательно.

```
Device    psm0 at atkbdc? irq 12
```

Драйвер мыши PS/2. Если таковая присутствует — обязателен.

```
Options   ATKBD_DFLT_KEYMAP
```

Эта опция позволяет включить в ядро некую клавиатурную раскладку по умолчанию.

```
Makeoptions  ATKBD_DFLT_KEYMAP="jp.106"
```

А эта опция именно и устанавливает такую раскладку. В примере из LINT она японская, но, разумеется, может быть и любой другой из присутствующих в каталоге /usr/share/syscons/keymaps/ (например, русской — ru.koi8-r). Включение этой опции делает ненужной подгрузку клавиатурной раскладки в файле /etc/rc.conf, о чем будет говориться в главе о системной консоли, средства которой для кириллизации, однако, видятся предпочтительными по ряду соображений. Так что, скорее всего, к этой опции прибегать не стоит. Соответственно, и предыдущая опция не нужна.

Видеосистема

В эту подсекцию я сгруппировал опции, имеющие отношения к видеорежимам.

```
Device    vga0 at isa?
```

Драйвер стандартного VGA-адаптера. Именно через него функционирует видеосистема в консольном режиме. И он необходим, сколь бы крутая видеокарта не стояла в реальности.

```
pseudo-device  splash
```

Этой строкой в ядро встраивается скринсейвер, который обычно загружается в виде модуля через /etc/rc.conf. Что мне кажется более предпочтительным.

```
Options   VGA_WIDTH90
```

Эта опция включает режим отображения в 90 колонок (т. е. знаков в строке). Необходима, если нет желания ограничиваться стандартными 80-колоночными режимами (что было бы нелепо на мониторах от 17 дюймов и выше).

```
Options   VESA
```

Включение стандартного VESA-режима, позволяющего, в частности, 132-колоночное отображение текста в консоли. Целесообразна для мониторов от 19 дюймов.

```
Options   SC_PIXEL_MODE
```

Поддержка т. н. растрового режима для текста. Позволяет получить нечто вроде графической консоли с единственным, правда, разрешением 800×600 пикселов. И не столь выразительного, как поддерживаемые в Linux.

Системная консоль

Такое название фигурирует в оригинале. Собранные здесь опции имеют отношение к общим свойствам системной консоли (начиная с выбора ее драйвера).

```
Device    sc0  at isa?
```

Включение драйвера системной консоли FreeBSD по умолчанию — syscons. Обязательная строка, если, конечно, не предпочесть т. н. консоль PCVT, которая тоже описана в этой секции. В которой, впрочем, смысла не вижу и потому не рассматриваю.

```
Options    MAXCONS=16
```

Эта опция устанавливает максимальное количество виртуальных консолей. По умолчанию их 16. Вероятно, можно увеличить — если этого вам покажется мало. А можно и уменьшить экономии ресурсов ради...

```
Options    SC_DFLT_FONT
```

Позволяет встроить в ядро шрифты с неким набором символов.

```
Makeoptions  SC_DFLT_FONT=cp850
```

Встраивает в ядро шрифт с конкретным набором символов, в том числе — в какой-либо из кириллических кодировок, присутствующих в каталоге /usr/share/syscons/fonts/. Делает ненужной загрузку шрифтов через /etc/rc.conf, описанную в главе о системной консоли. Относительно этих двух строк можно сказать то же, что и о клавиатурных раскладках — средства системной консоли кажутся мне более гибкими.

```
Options    SC_DISABLE_DDBKEY
```

Насколько мне известно, это нечто вроде "магических клавиш" в Linux, предназначенных для отладочных целей. Если эта опция включена, то в отладочный режим можно перейти нажатием клавишной комбинации <Control>+<PrintScreen> (она же — <SysReq>).

```
Options    SC_DISABLE_REBOOT
```

Опция, запрещающая перезагрузку посредством трех сакраментальных клавиш. Полезна при наличии детей в доме.

```
Options    SC_HISTORY_SIZE=200
```

Этой опцией задается размер (в строках) буфера экрана, доступного для пролистывания клавишами <ScrollLock>—<PageUp>/<PageDown>. Можно

изменить в любую сторону (я обычно выставляю значение 1000). Размер экранного буфера регулируется также средствами системной консоли *(см. главу 12)*, однако — только в сторону уменьшения.

```
Options     SC_MOUSE_CHAR=0x3
```

Включение этой опции подменяет группу кодов из верхней части кодовой таблицы, которые используются как курсором консольной мыши, так и русскими буквами [п, я, р, с] в шрифтах кодировки KOI8-R. Включение ее позволяет использовать мышь при загруженных шрифтах KOI8 (или, напротив, шрифты KOI8 при активизированной консольной мыши). Подробности — в *главе 12* (системной консоли). Я всегда включаю — на случай, если потребуется переопределение ввода/вывода терминала, например, средствами программы mapchan (о которой будет говориться в *главе 19*).

```
Options     SC_NORM_ATTR="(FG_GREEN|BG_BLACK)"
Options     SC_NORM_REV_ATTR="(FG_YELLOW|BG_GREEN)"
Options     SC_KERNEL_CONS_ATTR="(FG_RED|BG_BLACK)"
Options     SC_KERNEL_CONS_REV_ATTR="(FG_BLACK|BG_RED)"
```

Группа опций, позволяющая изменить расцветку консольного экрана по умолчанию. Опции с компонентом `REV` устанавливают реверсивный режим (светлый фон, темный шрифт), без оного — наоборот (темный фон, светлый шрифт). На мой взгляд, при необходимости средствами syscons это сделать проще...

Накопители

В этой подсекции — все опции, имеющие отношение к устройствам массовой памяти.

```
Device      ata
```

Общая поддержка ATA- (они же — EIDE) устройств. Необходима на всех современных машинах (если, конечно, не все накопители — SCSI). Следующие строки — конкретизация в отношении устройств.

```
Device      atadisk
```

Опция поддержки ATA-дисков — т. е. любых современных не-SCSI-винчестеров. И потому обязательна.

```
Device      atapicd
```

Поддержка CD-ROM с ATAPI-интерфейсом (все современные).

```
Device      atapifd
```

Поддержка т. н. ATAPI-floppy — это, по простому говоря, Zip-приводы и т. н. суперфлоппи (типа LS-120).

```
Device      atapist
```

Поддержка ATA-стримеров. Честно говоря, в жизни таких не видел...

Строка, касающаяся дисководов CD-ROM, необходима с вероятностью, близкой к 100%, прочие — при наличии соответствующих устройств.

```
Options     ATA_STATIC_ID
```

Статическая (т. е. не зависящая от переопределения через BIOS Setup) нумерация контроллеров IDE. Мне именно такая, "умолчальная", номенклатура представляется удобной, и оснований менять ее — не вижу.

После этого идет множество опций относительно всяких дисковых накопителей, но относящиеся к устаревшему оборудованию (ST-506-, ESDI- и просто IDE-диски и пр.). Поэтому все вплоть до строк про стандартные флоппи-дисководы, ныне можно не принимать во внимание.

```
Device      fdc0 at isa? port IO_FD1 irq 6 drq 2
```

Общая поддержка стандартных флоппи-дисководов — с нетерпением жду момента, когда это устройство можно будет, широко размахнувшись, выбросить на помойку. Но пока — строка практически необходимая.

```
Device      fd0   at fdc0 drive 0
Device      fd1   at fdc0 drive 1
```

А это — дисководы конкретные. Давненько не доводилось видеть машины с двумя флоппиками, так что достаточно первой строки.

Сетевые карты

В этой подсекции — множество опций, касающихся конкретных устройств, на которых я задерживаться не буду. Но есть и опция общего значения —

```
device      miibus
```

Этот драйвер требуется для любых сетевых соединений (не только Ethernet, но и через COM- или LPT-порты, а также USB-шину).

Коммуникационные порты

Здесь — опции поддержки стандартных портов ввода/вывода, которые пока никто не отменил, а также сетевых соединений.

```
Device      sio0 at isa? port IO_COM1 flags 0x10 irq 4
Device      sio1 at isa? port IO_COM2 irq 3
Device      sio2 at isa? disable port IO_COM3 irq 5
Device      sio3 at isa? disable port IO_COM4 irq 9
```

Поддержка стандартных последовательных портов (COM1—COM4 соответственно). Необходима для сериальных мышей, модемов (кроме USB-моделей). В то же время, если модем внешний и мультикарты для дополни-

тельных портов нет, можно ограничиться двумя первыми строками. При наличии шинной мыши потребуется устройство mse.

Далее следует кошмарное количество строк, относящихся к сетевым картам. Затем — строки, почти столь же многочисленные, описывающие звуковые карты ISA (среди них — и весьма экзотические), в актуальности чего также не уверен. Далее — опции касательно CD-ROM с нестандартными интерфейсами (не-IDE и не-SCSI) — думаю, они все уже вышли из строя физически.

Следующие ниже строки — поддержка параллельного интерфейса и устройств для него.

```
Device    ppc0 at isa? irq 7
```

Общая поддержка параллельного порта.

```
Device    ppi
```

Общая поддержка ввода/вывода через параллельный интерфейс.

```
Device    ppbus
```

По смыслу — шина (?) параллельного порта (комментарий в LINT отсутствует).

Судя по всему, эти три строки необходимы, если параллельный порт используется хоть в каком-то качестве.

```
Device    lpt
```

Поддержка принтера на параллельном порту. Необходима, если есть намерение на таком принтере печатать.

```
Device    plip
```

Поддержка сетевого соединения через параллельные порты. Требует как минимум общей поддержки сетевых соединений, о чем было сказано в соответствующей субсекции.

```
Device    vpo
```

Поддержка внешнего Zip-привода с подключением к параллельному порту. Требует, кроме всего прочего, еще и общей поддержки SCSI-устройств (scbus и da) из секции SCSI Devices.

Для коммуникационных устройств приводится еще изрядное количество опций, касающихся их дебаггинга. Я их не рассматривал и в свое ядро никогда не включал.

Устройства USB

```
Device    usb
Device    uhci
Device    ohci
Device    ugen
```

В этих строках перечисляются драйверы общей поддержки шины USB (первая) и ее контроллеров (вторая и третья), а также общей поддержки USB-устройств (четвертая). Они необходимы для любых USB-устройств.

```
Device    uhid
```

"Гуманистический" интерфейс устройств USB. Я так и не понял, что это такое, но, похоже, без него можно обойтись — у меня USB-мышь без него спокойно работает. Следующие же строки — поддержка отдельных типов устройств на шине USB. Ядро по умолчанию включает их все — так что можно ограничиться только теми, что реально имеются в наличии.

```
Device    ukbd
```

USB-клавиатура.

```
Device    ulpt
```

USB-принтер

```
Device    umass
```

Внешний Zip-привод с USB-интерфейсом. Требуется включение общей поддержки SCSI-устройств (scbus и da) в секции SCSI Devices.

```
Device    umodem
```

USB-модем.

```
Device    ums
```

USB-мышь.

```
Device    uscanner
```

USB-сканер.

Прочие устройства

Поддержка звуковых карт Plug-and-Play на шине PCI (а иные ведь давно перевелись). Для большинства современных аудиоустройств требуется только команда.

```
Device    pcm
```

При желании позабавиться классикой в свободный часок — эту строку нужно включить (по умолчанию в GENERIC она отсутствует).

```
Device    apm0 at nexus? disable flags 0x20
```

Поддержка управления питанием. Этого нам нужно? Думается, что на настольной машине — не очень, хотя на ноутбуке может пригодиться.

Теория и практика конфигурирования

Очень надеюсь, что из моего описания можно составить некоторое представление об основных опциях ядра FreeBSD, интересных для конечного пользователя, и об их взаимосвязях. Пора приступить собственно к конфигурированию. Для чего, запасшись предварительно правами root, например, командой

```
$su -l
```

переходим в соответствующий каталог

```
$cd /usr/src/sys/i386/conf
```

После этого делаем копию файла, принятого за основу грядущей конфигурации ядра, например:

```
$cp GENERIC NEW_KERNEL
```

поскольку работать с оригинальными конфигурационными файлами — занятие, как правило, нездоровое. И, наконец, открываем новообразованный файл в любимом текстовом редакторе:

```
$joe (vi, emacs, ee, jed, le - ненужное зачеркнуть) NEW_KERNEL
```

Не забыв, естественно, о его, редактора, настройках по умолчанию. Ибо если в таковых значится автоматический перенос слов (по границе экрана или числу знаков), — опцию эту немедленно отключаем. Впрочем, мы ведь не забываем, что это — общее правило при работе с любыми конфигурационными файлами...

Во второй виртуальной консоли (или в другом буфере редактора, это уж как кому угодно) хорошо бы при этом открыть и LINT — сверяться с картотекой, так сказать. Причем желательно сделать это не от лица администратора, а прикинувшись обычным пользователем (или воспользоваться не редактором, а командой `less`) — чтобы ненароком не повредить. Полезно держать наготове и третью консоль, для чего — скажу чуть позже.

Дальнейшие действия на ближайшие минуты и часы как будто очевидны — это вычеркивание из своего файла всего лишнего и перетаскивание в него необходимого — из LINT, в соответствии с рекомендациями предыдущих разделов.

А вот теперь, хищно занеся все десять пальцев над клавиатурой, самое время задуматься о глубинной сути своих действий. То есть — о стратегии конфигурирования ядра.

Все известные мне рекомендации по этому поводу можно свести к двум тезисам. Первый — включать не только те опции, которые жизненно необходимы в данный момент, но и те, что могут потребоваться впредь, дабы более к этому вопросу не возвращаться. Второй, напротив, — безжалостное иско-

ренение всех отсутствующих устройств и связанных с ними опций (в той мере, разумеется, в какой это допустимо взаимозависимостями).

Первый подход, вероятно, оправдан при администрировании серверов. Однако второй видится более приемлемым на настольной машине: ведь только таким образом можно добиться идеально настроенной системы для индивидуального использования.

Отдельный вопрос — что делать с устройствами и опциями, назначение которых не вполне понятно. И которое не удалось выяснить сколь угодно глубоким изучением источников и комментариев к LINT. Обычный совет — во всех сомнительных или неясных случаях избирать вариант, предложенный по умолчанию (в соответствие со старой Дальстроевской поговоркой — "Если не знаешь, что делать — делай, что приказано").

Это положение кажется мне неоправданным в гносеологическом аспекте: если доверять, не проверяя, умолчаниям, непонятные опции так навсегда и останутся непонятными. И потому я, напротив, предложил бы (при наличии толики времени и любопытства) иной подход — безжалостное исключение поддержки всех отсутствующих устройств и всех непонятных опций. Ведь, с большой долей вероятности, они потому и непонятны, что отвечают за функции и устройства, которые не нужны или не используются.

Худшее, что может грозить при таком подходе, — сообщения об ошибках при построении зависимостей или то, что ядро откажется компилироваться. И то, и другое не страшно — внимательное чтение сообщений на экране всегда позволит локализовать причину сбоя. После чего можно вернуться к конфигурированию и внести соответствующие коррективы.

Вторая потенциальная опасность — отказ при загрузке ядра. Однако она легко нейтрализуется соблюдением несложных мер техники безопасности, о которых пойдет речь в заключение этой главы. Так что трус в карты не играет и ядро не пересобирает — вперед.

Теперь по существу конфигурирования. Исходя из общих соображений, если есть выбор, включить опцию в ядро или использовать загружаемый модуль, следует предпочесть второй вариант. Так, мне видится бессмысленным конфигурировать ядро с поддержкой скринсейверов — не столь уж он важная птица, этот скринсейвер. С другой стороны, я не уверен, что такие часто требуемые функции, как обращение к DOS-разделам или CD-ROM, будут эффективно использоваться при модульной поддержке. Впрочем, не знаю, не пробовал.

Ну и видится очевидным (по крайней мере мне), что не следует включать в ядро такие функции, которые успешно реализуются через файлы начальной конфигурации, как-то: загрузка экранных шрифтов и раскладок клавиатур, не говоря уже о расцветке экрана и прочем подобном баловстве. Ведь при желании внести изменения в такие настройки (а у меня, например, такое

желание возникает довольно часто) правка /etc/rc.conf — занятие существенно более простое, чем пересборка ядра. Конечно, встроенные в ядро опции такого рода можно просто "забить" изменением конфигурационных файлов, но тогда возникает вопрос: а за каким мы их, собственно, в ядро включали?

Да, чуть не забыл самое главное: перед началом конфигурирования ядра следует внимательно ознакомиться со всем своим оборудованием, во-первых, и с тем, что по поводу его думает FreeBSD — во-вторых. Первое утверждение понятно. А второй вопрос тоже снимается очень просто. При начальной загрузке системы все сведения такого рода, доступные пониманию системы, выводятся на экран ослепительно белым шрифтом.

Разумеется, не обязательно спешно жать три клавиши и пялить глаза в экран, стараясь уследить за мельтешением букв и цифр. Специально для наблюдений такого рода заготовлена команда `dmesg`, которая в любой момент (не требуя даже прав root) готова выдать нам абсолютно ту же информацию, что выводится при загрузке. И даже позволит перенаправить ее в файл

```
$dmesg > ~/my_hardware
```

Каковой целесообразно открыть (вот зачем мы готовили третью консоль) и сверяться с ним в процессе работы.

Но вот последняя опция из LINT перетащена, последний комментарий поставлен. Что делать дальше? Для начала не худо внести в файл (опять же в виде комментариев) информацию о том, когда и на кой ядро пересобиралось. Например, так

```
# 19 янв 2002, включение VGA_90, VESA и raster text mode
```

Это поможет потом, когда пересобранных собственноручно ядер наберется с полдюжины. Во-вторых, все сделанные изменения нужно (да неужели?) сохранить. Я обычно еще и копирую ядро в свой пользовательский каталог под мнемонически понятным именем.

После всего этого остается всего только заключительный этап — сама по себе сборка ядра. Для начала — конфигурирование командой

```
$config NEW_KERNEL
```

чем будет создан каталог /usr/src/sys/compile/NEW_KERNEL со всеми потребными для сборки ядра файлами. Куда и следует отправиться для дальнейших действий:

```
$cd../../compile/NEW_KERNEL
```

А действия эти — довольно стандартны: последовательность команд `make depend`, `make all`, `make install`. При полной уверенности в себе их можно дать одной строкой в связке

```
make depend; make all; make install
```

или

```
make depend && make all && make install
```

В принципе это, как будет показано в главе о Shell, разные формы, но в данном конкретном случае результат будет идентичным, а именно: проверка и создание зависимостей, собственно компиляция и установка двоичных файлов в должное место. В ходе чего можно пить пиво (хотя на P-4/1,9 ГГц процесс займет менее 10 минут — дай бог рюмку водки треснуть да закусить успеть). А, вернувшись — просто перезагрузить машину.

Однако для начала это — не лучший способ: вернувшись, вместо скомпилированного и установленного ядра можно спокойно обнаружить сообщение об ошибке. Выявить которую после процедуры расслабления будет гораздо труднее.

Так что по первости лучше действовать последовательно. И внимательно читать сообщения на экране. В этом случае ошибка на любой стадии локализуется достаточно просто. Как и исправляется — повторным редактированием конфигурационного файла и повторением действий по сборке.

Особенно не следует спешить с командой `make install`: она не только помещает новое ядро (с именем kernel) куда следует (а именно — в корень), но и переименовывает прежнее — в kernel.old. В результате чего после двух неудачных попыток можно получить два вполне негодных к использованию варианта ядра. Так что `make install` следует предварить копированием исходного, заведомо исправного, ядра под другим именем. Желательно — коротким и запоминающимся, время на его набор при перезагрузке ограничено. А именно это придется делать, если вновь собранное (и, казалось бы, успешно) ядро откажется грузиться.

Впрочем, это уже вопрос техники безопасности — и предмет следующего раздела. А пока предположим, что процесс сборки и перезагрузки, с первой ли попытки, или с тридцать первой, завершился успешно. И после авторизации мы имеем удовольствие наблюдать приветственное сообщение системы, в числе прочего сообщающее и о новом ядре в форме:

```
FreeBSD 4.4-RELEASE (NEW_KERNEL) #0: Mon Jan  7 18:52:13 MSK 2002
```

где символы в скобках — в данном случае `NEW_KERNEL`, — и есть имя нового ядра (совпадающее с именем, которое было дано конфигурационному файлу). Что еще осталось сделать для доведения процесса до логического конца?

Вполне возможно, что и ничего. Но, чтобы убедиться в этом, следует вспомнить, сколько и каких новых устройств (т. е. строк вида `device...`) было встроено в ядро в ходе его конфигурирования. И сверить их с именами файлов устройств в каталоге /dev. После чего, как известно, остается два выхода. Если для всех новых устройств соответствующие файлы имеются — значит, все хорошо. Если же нет — их следует создать.

Делается это командой MAKEDEV, имеющей местопребывание в том же каталоге устройств:

```
$/dev/MAKEDEV /dev/имя_файла
```

причем именно так, с указанием абсолютного пути — каталог /dev не прописан в переменной PATH даже суперпользователя (от лица которого только и можно действовать). Если же предварительно перейти в каталог /dev, команду следует дать в форме

```
$./MAKEDEV имя_файла
```

В качестве имени файла устройства, как правило, следует использовать наименование устройства из соответствующей строки, скорее всего — с добавлением порядкового номера. Например... — затрудняюсь привести реальный пример, у меня файлы для всех требуемых устройств были изначально, — вот, придумал: в строке

```
$device    uscanner
```

значение параметра device — uscanner, — будет родовым именем для USB-сканеров. Однако конкретный, только что купленный, сканер с ним идентифицироваться не будет. И потому файл устройства будет создан как

```
$/dev/MAKEDEV /dev/uscanner0
```

для первого и единственного (а у кого их больше одного?) USB-сканера. Кроме того, справку о допустимых аргументах команды MAKEDEV можно получить из комментария в ее теле — это простой текстовый скрипт, который просматривается любым способом, например:

```
$less /dev/MAKEDEV
```

Вот теперь — действительно все, можно наслаждаться жизнью (и новым ядром). Если, конечно, ранее жизнь эта не заставит обратиться к изучению следующего раздела.

Проблемы и решения

Конечно, заниматься изучением техники безопасного... перекомпилирования ядра, конечно, а вы что подумали? — лучше до, а не после (говорят, что вместо — еще спокойней, но уж больно скучно). Почему этот раздел и следовало бы поставить в начало главы. Но ведь чтобы решать проблемы — следует хотя бы примерно знать, где и отчего они могут возникнуть. Так что, получается, именно здесь ему самое место.

Итак, проблемы могут возникнуть а) при сборке ядра и б) при его загрузке. Первая из них — сообщение об ошибке в ходе выполнения команды make depend. Причина, скорее всего, в том, что какая-либо из включенных опций

требует обязательного наличия другой, о которой при конфигурировании было забыто. Некоторые из таких взаимосвязей очевидны — нельзя включить поддержку ATAPI CD-ROM без общей поддержки ATA-интерфейса, и т. д. Другие случаи — не столь прозрачны. Например, включение сетевых соединений любых типов (даже по параллельному или последовательному интерфейсу) требует, чтобы имелась общая поддержка Ethernet-соединения, для работы драйверов внешних Zip-приводов, вне зависимости от интерфейса (LPT ли или USB), требуется общая поддержка устройств SCSI.

Тут могут помочь:

- внимательное чтение сообщений на экране — по ним в большинстве случаев можно определить, на каком именно устройстве случился "затык";
- изучение комментариев в файле LINT — большинство случаев взаимозависимости там документировано;
- обращение к FAQ проекта FreeBSD — с большинством мыслимых проблем пользователи этой системы за века ее существования уже сталкивались, и способы их решения, скорее всего, известны.

Вторая проблема при сборке — отказ ядра компилироваться. Это, скорее всего, вызвано тем, что была удалена (случайно или из неких высших соображений) какая-либо опция, помеченная как обязательная. Например, судя по тому же FAQ, типичный случай — изъятие поддержки сопроцессора. Я тоже по первости на это попался — в силу аберрации сознания строка

```
Device    npx0...
```

проассоциировала у меня с той самой эмуляцией сопроцессора, что набила мне оскомину еще в Linux.

Если проблема в этом — решение также не сложно. Из экранного сообщения можно почти всегда (почти — говорю для страховки, с исключениями пока не сталкивался) определить вызвавшую ошибку опцию. И, вернувшись к тексту LINT, убедиться, что именно она-то и имеет пометку типа `"NOT want to remove"`, `"is mandatory"`, `"keep this!"` или нечто подобное. После чего остается вернуться к своему собственному конфигурационному файлу и внести соответствующие коррективы.

Еще одна возможная причина ошибки при сборке — конфликт опций. Однако я с этим пока не сталкивался, и когда и в какой форме проявляется — не знаю. Исходя из общих соображений, могу только предположить, что вероятные источники таких конфликтов лежат в области сетевых интерфейсов и (или) устройств.

Что же касается проблем при загрузке, то она — одна. А именно, отсутствие загрузки как таковой. В этом плане меня Бог также миловал — если уж ядро

мне собрать удавалось, оно грузилось исправно (в отличие от Linux, где такое случалось часто, особенно — при экспериментах с frame buffer и графическими консолями). Так что опять же могу предположить, что это нечто вроде проявления panic mode.

Однако уж борьба с этим недостатком — просто элементарна. Во-первых, как уже упоминалось, после сборки нового ядра, но до его инсталляции (т. е. после команды `make all`, но до отдачи директивы `make install`) старое работоспособное ядро должно быть скопировано в тот же корневой каталог, но под другим именем:

```
$cp /kernel /OLD
```

Почему и от чего это делается — я уже говорил. А для чего — сейчас скажу. После первой неудачной загрузки нового ядра опять перезагружаем машину и в момент появления приглашения

```
boot:
```

и начала обратного отсчета времени вводим данное нами имя старого ядра — OLD. С коего машина и грузится как ни в чем не бывало. После чего продолжаем эксперименты. Либо, если мера терпения исчерпана, просто переименовываем старое ядро обратно в /kernel и работаем, руководствуясь принципом, что лучше синица в руках, чем вилы (те самые, что у BSD-демона) в бок.

И последнее соображение о безопасности — это уже общего характера. При всех общесистемных настройках (ядра ли, файлов начальной загрузки, загрузчика как такового) следует иметь под рукой fixit-носитель (лучше всего — штатный CD-ROM из комплекта). Загрузившись с него, всегда можно к получившейся файловой системе (т. н. Live File System) подмонтировать бывший корневой раздел жесткого диска и выполнить на нем всю необходимую правку.

Короче говоря, нет трудностей, которые не могли бы преодолеть большевики. А поскольку мы — не большевики, то таких трудностей тем более нет. Даже если мы сами и изобретем их на свою... голову.

Дополнительные источники

- Сведения об устройстве ядра UNIX-систем вообще можно почерпнуть из фундаментальных трудов Мориса Баха "Архитектура операционной системы UNIX" (рус. пер. А. В. Крюкова можно обнаружить, например, на **http://linux.perm.ru/doc/linux/bach/** или в библиотеке Мошкова) и С. Д. Кузнецова "Операционная система UNIX" (в частности, есть на сайте **http://alone.dubna.ru:8080/history/contents.html**).

- Кроме этого, для более углубленного изучения тем, затронутых в этой и последующих главах, можно рекомендовать книгу: Робачевский А. Операционная система UNIX. — СПб.: БХВ-Петербург, 2000.
- Описанию устройства именно систем линии BSD посвящена книга Мак-Касика с соавторами "Архитектура и реализация операционной системы 4.4BSD", переводы отдельных фрагментов которой имеются на сайте **http://www.FreeBSD.org.ua/design-44bsd/**).
- Самое подробное описание процесса регенерации ядра — на сайтах Ивана Паскаля (**http://www.tsu.ru/~pascal/**) и Дмитрия Карпова (**http://www.pi2.ru:8101/prof/techsupp/fbsdinst.htm**). Ну и, разумеется, не следует пренебрегать Handbook (**http://www.FreeBSD.org.ua/handbook/**) и сборником часто задаваемых вопросов (**http://www.FreeBSD.org.ua/faq/**).

Глава 7

Загрузка и останов системы

Первое, что должна сделать операционная система после своей установки — это загрузить как минимум саму себя. Для чего она должна располагать соответствующим средством, а именно — начальным загрузчиком. И FreeBSD в этом смысле — не исключение из общего правила.

Стадии начальной загрузки

Загрузчик FreeBSD в некоторых источниках именуется Easy Boot, что вполне соответствует его сущности — он мал, прост и легок в использовании. Он состоит из трех компонентов, которым соответствуют файлы в каталоге /boot — /boot/boot0, /boot/boot1 и /boot/boot2. И, соответственно, загрузка системы осуществляется в три этапа.

Первый из них управляется компонентом /boot/boot0, записываемым (если не используется какой-либо из рассмотренных ниже внешних мультисистемных загрузчиков) в MBR (Master Boot Record) — главную загрузочную запись физического диска. Программа в MBR не может превышать по размеру 512 байт, поэтому функции /boot/boot0 очень просты — определить первичные разделы диска (слайсы) и передать управление программам в их загрузочных секторах.

Тем не менее, /boot/boot0 вполне в состоянии выполнять и функции мультисистемного загрузчика. Если на диске имеется более одного первичного раздела с различными операционными системами, он предлагает загружать их на выбор примерно следующим образом:

```
F1 DOS
F2 FreeBSD
F3 Linux
F4 Имя_рек

Default: F2
```

В качестве варианта по умолчанию сразу после установки FreeBSD выступает эта система. В дальнейшем по умолчанию предлагается система, выбранная в предыдущем сеансе работы.

При наличии в системе второго физического диска, /boot/boot0 способен передать управление программе в его загрузочный сектор. В этом случае в меню выбора при загрузке добавляется строка

```
F5 Drive 1
```

И если на втором диске установлена какая-либо операционная система, при соответствующем выборе она будет благополучно загружена. Хотя не со всеми ОС и их загрузчиками это проходит — почему к этому вопросу мы еще вернемся в одной из последних глав *части II*.

При выборе в меню начальной загрузки FreeBSD (или если она — единственная ОС на машине) управление передается второму компоненту загрузчика — /boot/boot1. Он локализован в загрузочном секторе первичного раздела BSD (сиречь слайса) и также по определению имеет размер 512 байт. Его роль — не более, чем определить на слайсе третий компонент загрузчика, /boot/boot2 и передать управление ему.

Выше я говорил, что каждый компонент загрузчика FreeBSD ассоциирован с определенным файлом. Однако это — в определенной мере условность, поскольку собственно к файловой системе FreeBSD они отношения не имеют. Согласно Руководству пользователя FreeBSD, "/boot/boot0, /boot/boot1 и /boot/boot2 ... не являются реальными копиями MBR, загрузочного сектора или метки диска соответственно". Тем не менее, третий компонент загрузчика, /boot/boot2, запускает уже программу, являющуюся составной частью файловой системы FreeBSD, ассоциированную с исполняемым файлом /boot/loader. Именно он в документации FreeBSD и называется собственно загрузчиком. Хотя говорят, что ранее /boot/boot2 загружал уже непосредственно ядро системы.

При запуске программы /boot/loader происходит считывание конфигурационных файлов /boot/loader.rc и /boot/loader.conf. Затем выдерживается 10-секундная пауза в ожидании нажатия клавиши, и если такового не последует, в автоматическом режиме загружается ядро. Если ожидание было прервано, появляется приглашение `ok`, после которого в режиме командного интерпретатора (подобного стандартной командной оболочке shell) становится возможным ручное конфигурирование процесса загрузки.

Набор команд, встроенных в оболочку загрузчика, можно получить, набрав в командной строке символ "?" А дав команду

```
help тема
```

можно получить и справку по интересующему аспекту использования оболочки загрузчика. Список тем помощи получается при указании опции `index`.

К ручному конфигурированию процесса загрузки приходится прибегать не часто, и потому я не буду останавливаться на деталях (тем более, что это подробно описано в Руководстве пользователя). Затрону лишь два момента, важных в аварийной ситуации.

Первый из них — загрузка в однопользовательском режиме, что необходимо, когда запуск системы обычным образом почему-либо невозможен (самый тривиальный случай — забыт пароль суперпользователя). Это достигается вводом после приглашения команды

```
$boot -s
```

При этом без запроса пароля обретаются права root-оператора со всеми вытекающими последствиями, так что злоупотреблять этим режимом не следует. К тому же обычную работу в нем выполнять практически невозможно — доступна только одна системная консоль, не удастся запустить систему X Window, и т. д.

Второй момент — это загрузка ядра, отличного от принятого по умолчанию, т. е. от /kernel. Необходимость в этом возникает при отладке нового ядра, или просто после неудачной его регенерации. Чтобы загрузить какое-либо иное ядро, следует дать команду

```
$unload
```

удаляющую из памяти все загруженные ранее модули, а вслед за этим — команду

```
$boot имя_файла
```

с указанием имени файла образа требуемого ядра (например, kernel.old или kernel.GENERIC). Вместо команды `boot` можно дать и команду

```
$load kernel.old
```

Если имя файла образа ядра забыто, его можно посмотреть командой `ls` — данная без аргументов, она выведет содержимое корневого каталога загрузочного устройства. А просмотреть список разделов загрузочного слайса (и вообще всех доступных дисковых устройств) можно с помощью команды `lsdev`. Если же дать ее в форме

```
$lsdev -v
```

для каждого из устройств будет выведена информация о его размере. Аналогично, команда `lsmod` даст список загруженных на текущий момент модулей, а опция `-v` при ней — адреса областей занимаемой ими памяти.

Для продолжения нормальной загрузки в приглашении нужно дать команду `boot` без опций (или с опцией `-v`, которая обеспечит вывод более подробной информации о ходе загрузки).

Настройка загрузчика

За конфигурирование загрузчика по умолчанию отвечает файл /boot/defaults/loader.conf, а пользовательские настройки его устанавливаются в файле /boot/loader.conf. Первый файл описывает все возможные опции загрузки и не предназначен для редактирования — по умолчанию он закрыт для записи даже суперпользователю (конечно, последний всегда может присвоить себе такое право, но делать этого не стоит).

Файл /boot/defaults/loader.conf разбит на следующие секции:

- Basic configuration options;
- Splash screen configuration;
- Loader settings;
- Kernel settings;
- Kernel tunables;
- Filesystem and related modules;
- Screen saver modules;
- Emulation modules;
- Networking modules;
- Networking drivers;
- Netgraph modules;
- Sound modules;
- USB modules;
- Other modules;
- Module loading syntax example.

Секция Basic configuration options включает основные опции загрузчика — имя файла образа ядра (по умолчанию — `kernel="/kernel"`) и опции команды `boot` (по умолчанию отсутствуют), пользовательские сценарии при загрузке, имена пользовательских конфигурационных файлов загрузчика.

В секции Splash screen configuration указывается, следует ли загружать модули скринсейверов, и если да — то в каком формате, а также имена соответствующих splash-файлов.

В секции Loader settings устанавливается время ожидания действий пользователя перед началом загрузки ядра (как уже говорилось, по умолчанию 10 секунд), устройство для загрузки и устройство, монтируемое в качестве корневой файловой системы, пути к файлам подгружаемых модулей и т. д.

В секциях Kernel settings и Kernel tunables перечислены возможные опции интерактивного конфигурирования и отладки ядра, управления памятью, использования PC-крат и т. д. (по умолчанию все они выключены).

Секция Filesystem and related modules содержит перечень файловых систем, поддержка которых может быть реализована в виде подгружаемых модулей (т. е. которые не обязательно жестко включать при конфигурировании ядра).

Назначение секции Screen saver modules понятно из названия — это, во-первых, разрешение подгружать модули, отвечающие за включение скринсейверов вообще (которые могут быть и встроены в ядро, как было показано выше), и, в положительном случае, название конкретного модуля. Список скринсейверов, доступных в виде модулей, можно посмотреть в каталоге /modules (файлы вида *_saver.ko).

В секции Emulation modules можно включить поднадоевшую опцию эмуляции математического сопроцессора, а также некоторых операционных систем (Linux, System V, Santa Cruz Operation).

Секции Networking modules, Networking drivers и Netgraph modules отвечают за загрузку сетевых устройств и протоколов, секция Sound modules выполняет аналогичную функцию для звуковых карт, секция USB modules — для устройств USB. Секция Other modules позволяет подгружать поддержку устройств типа телетюнеров (на чипах Brooktree Bt848/Bt878).

Наконец, заключительная секция, Module loading syntax example, — это просто примеры синтаксиса опций, определяющих загрузку модулей.

В сущности, настройка загрузчика осуществляется редактированием файла /boot/loader.conf. По умолчанию его содержание имеет вид вроде следующего:

```
# -- sysinstall generated deltas -- #
userconfig_script_load="YES"
```

И задача пользователя сводится к тому, чтобы выбрать из файла /boot/defaults/loader.conf требуемые ему опции, перенести их в файл /boot/loader.conf и придать соответствующие значения.

Стартовые файлы

По завершении загрузки ядра (и его модулей) оно передает управление первому пользовательскому процессу — `init`, запускаемому одноименной программой, расположенной в каталоге /sbin. Именно он порождает все прочие процессы в системе, в том числе — и считывание стартовых файлов или файлов ресурсов (именуемых также rc-файлами — от *resources control*). Эти файлы определяют, какие процессы запускаются к моменту завершения полной загрузки системы (фиксируемой появлением приглашения к авторизации).

Все стартовые файлы сосредоточены в каталоге /etc. Важнейшими из них являются /etc/defaults/rc.conf и etc/rc.conf. Подобно ситуации с конфигурированием загрузчика, первый описывает все возможные стартовые опции по умолчанию и не предназначен для редактирования. Файл же /etc/rc.conf, генерируемый в процессе установки программой /stand/sysinstall, описывает установленные пользователем отклонения от "умолчальной" конфигурации. Изменения в нем могут быть внесены в любой момент или повторным запуском /stand/sysinstall, или, в более общем случае, ручным редактированием в текстовом редакторе.

Рассмотрим структуру файла /etc/defaults/rc.conf. Он состоит из четырех секций:

- Important initial Boot-time options;
- Network configuration sub-section;
- System console options;
- Miscellaneous administrative options.

В первой секции описываются главные опции начальной загрузки, такие как включение файла подкачки (в дополнение к swap-разделу на жестком диске), доступность устройств типа PC-Card (PCMCI), местоположение дополнительных стартовых файлов, имена дополнительных сценариев, исполняемых при загрузке. По умолчанию все эти опции не задействованы в файле /etc/rc.conf и если значения их булевы — отключены.

Вторая секция разделяется, в свою очередь, на многочисленные подсекции, в которых сгруппированы опции по настройке сетевых соединений и систем безопасности.

Секция системной консоли позволяет сконфигурировать клавиатурные раскладки и экранные шрифты, таблицы перекодировки, мышь, а также задать опции экрана и клавиатуры, общие для всех виртуальных консолей. Все эти вопросы будут подробно рассмотрены в соответствующей главе.

В последней секции описываются стартовые сервисы, запускающие демонов печати, поддержки USB-шины, доставки почты и другие, а также опции совместимости с другими операционными системами.

В каталоге /etc содержатся и другие rc-файлы. Из них интерес для пользователя может представлять файл rc.syscons, в котором можно дополнительно настроить параметры отдельных виртуальных терминалов (о чем будет говориться в соответствующей главе).

Кроме этого, некоторое отношение к загрузке системы имеют файлы: /etc/ttys, в котором настраиваются виртуальные консоли (их количество и тип), /etc/fstab, описывающий автоматически монтируемые при загрузке файловые системы, /etc/motd, в котором задается содержание сообщения, выводимого на системную консоль после загрузки системы.

Представляет интерес также файл /etc/syslog.conf, указывающий пути стандартного вывода для всякого рода системных сообщений. В частности, именно его редактированием можно избавиться от надоедливых сообщений, выскакивающих в самый неподходящий момент на консоли root-оператора. Для этого их нужно перенаправить с консоли в какой-либо временный файл (или просто "в ничто" — в файл /dev/null). Что, впрочем, оправдано только на локальной машине.

Останов системы

Работа не может продолжаться вечно — хотя в принципе FreeBSD приспособлена именно для непрерывной работы, без перезагрузки и выключения машины. Тем не менее, время от времени эти процедуры проделывать приходится. Как? Существует один неправильный способ завершения работы FreeBSD и несколько правильных.

Неправильным способом является "холодный" перезапуск компьютера, вне зависимости от того, выполняется через кнопку Reset или выключением питания. В отличие от некоторых других UNIX-подобных систем, например, Linux, в FreeBSD он редко приводит к фатальным последствиям (например, полному разрушению файловой системы). Тем не менее, вероятность такого исхода существует. А при использовании асинхронного режима работы с диском или включении режима Soft Update (о них будет говориться в *главе 9*) вполне возможно повреждение пользовательских файлов, с которыми осуществлялась текущая работа. И потому прибегать к "холодной" перезагрузке системы следует только в том случае, если все остальные, правильные, способы категорически отказываются работать.

Из правильных способов первым является классический метод "трех пальцев" — нажатие комбинации клавиш <Alt>+<Ctrl>+<Delete>. Оно корректно завершает все текущие процессы, после чего происходит перезагрузка машины. В момент появления меню начального загрузчика для выбора ОС (стадия, управляемая /boot/boot0) питание можно безбоязненно отключить.

Перезагрузка методом "трех пальцев" доступна любому пользователю системы. Если, конечно, соответствующая возможность не отключена при сборке ядра с помощью включения опции

```
options     SC_DISABLE_REBOOT
```

в файле его конфигурации, о чем говорилось в *главе 6*. После этого перезагрузка и отключение машины становятся исключительно прерогативой администратора, которому доступны соответствующие команды.

Этими командами являются: `reboot`, `halt` и `shutdown`. Первая из них, данная без опций, отправляет сигналы завершения всем текущим процессам и вызывает немедленную перезагрузку компьютера, аналогично комбинации

клавиш <Alt>+<Ctrl>+<Delete>. Команда `halt` выполнит те же действия по завершению работы, после чего осуществляется выход из системы. Далее ее можно перезагрузить (нажатием любой клавиши) или просто выключить питание.

Команды `halt` и `reboot` имеют несколько общих опций (например, отмены записи в log-файлы по выходе, или запрещения синхронизации кэшированных данных). Опция `-d`, используемая только с командой `reboot`, позволяет создавать дампы памяти перед перезагрузкой.

Команда `shutdown` — наиболее гибка. С ее помощью можно как остановить работу системы (опция `-h`), так и вызвать ее перезагрузку (опция `-r`), причем — как немедленно, так и через любой, явным образом заданный промежуток времени, например, команда

```
$shutdown -h +5
```

приведет к перезагрузке системы через пять минут, а команда

```
$shutdown -r now
```

предписывает немедленную (опция `now`) ее перезагрузку. А прибегнув к опции `-p`, можно (на платах формата ATX с соответствующими настройками BIOS) вызвать и автоматическое выключение питания.

Все перечисленные команды корректно завершают работу системы в том смысле, что синхронизируют дисковые операции (при асинхронном доступе к файловым системам) и производят все требуемые обновления при использовании режима Soft Update, после чего размонтируют используемые файловые системы (в том числе и смонтированные вручную). А команда `shutdown` даже выводит сообщение об оставшемся до перезагрузки или останова времени (а также — произвольное сообщение, указываемое после опций). Однако ни одна из них не избавляет от необходимости сохранять пользовательские данные в прикладных программах — изменения в открытых файлах будут утеряны, если их предварительно не записать.

Использование команды `shutdown` позволяет избежать также перезагрузки системы после внесения практически любых изменений в общесистемные конфигурационные файлы: полный рестарт системы фактически необходим только после перекомпиляции ядра или при изменении дискового раздела (partition), на котором размещена корневая файловая система.

Во всех же остальных случаях достаточна реинициализация системы переводом ее в однопользовательский режим:

```
s$hutdown -r time
```

Запущенная с консоли root-оператора, она немедленно делает недоступными все остальные виртуальные консоли — текущие пользовательские сеансы на них прекращаются и повторная авторизация становится невозможной.

Однако если дать эту команду с пользовательской консоли, получив на ней временно права администратора командой `su`, она остается доступной и после выхода из привилегированного режима.

Перейти в однопользовательский режим можно и командой `kill` в отношении процесса init (об этом будет говориться в *главе 8*). Данная в виде

```
$kill 1
```

она приведет к прекращению всех пользовательских сеансов и активизации однопользовательского режима — в отличие от команды `shutdown`, на первой системной консоли, а не на консоли, с которой она была дана.

Практическая работа в однопользовательском режиме неудобна, небезопасна и нецелесообразна. И потому после этого следует сразу же перейти обратно в многопользовательский режим. Для этого дается команда

```
$shutdown now
```

без всяких опций, после чего следует нажать <Enter> и набрать `exit`. Это вызывает повторное приглашение к авторизации на текущей (привилегированной) консоли и возможность авторизации — на всех прочих.

Следует учесть только, что после такой реинициализации не все стартовые сервисы запускаются автоматически. В частности, вероятнее всего, придется вручную запустить демон консольной мыши. Делается это командой `moused` с указанием типа мыши и соответствующего файла устройства, о чем подробнее будет сказано в *главе 12* о системной консоли FreeBSD.

Дополнительные источники

Процесс загрузки и останова FreeBSD описан в руководстве пользователя (Handbook) — **http://www.FreeBSD.org.ua/handbook/**. Дополнительные сведения можно получить из статьи Джея Ричмонда "Установка и использование FreeBSD совместно с прочими операционными системами" (**http://freebsd.org.ru/tutorials/mdltios.html**).

Глава 8

Процессы

В этом и двух следующих главах будут рассмотрены ключевые понятия, на которых базируются UNIX и все UNIX-подобные системы — процессы, файлы и объединяющие их файловые системы, учетные записи, а также атрибуты всех этих объектов. Можно сказать, что в UNIX пользователь, определяемый своей учетной записью, имеет дело только с двумя объектами — процессами и файлами, а взаимодействие с ними осуществляется в соответствии с их атрибутами и атрибутами его учетной записи.

Не то чтобы эти сведения были жизненно необходимы конечному пользователю. Более того, начинать практическую работу в FreeBSD вполне можно и без них. Однако знание основных понятий, во-первых, поможет подходить более осмысленно к процессу настройки и использования системы, во-вторых, — весьма полезно в нештатных ситуациях. И в-третьих, они просто интересны сами по себе...

Тем не менее, автор не претендует на исчерпывающее изложение устройства UNIX-систем вообще и FreeBSD в частности, и потому изложенные в этих главах сведения носят ограниченный характер. Для более глубокого изучения затронутых здесь вопросов есть смысл обратиться к дополнительным источникам информации.

Вводные замечания

Понятие учетной записи представляется наиболее непривычным пользователю, переходящему с платформы Windows. А поскольку, в соответствии с принятой (и аргументируемой чуть ниже) логикой изложения, именно оно будет рассматриваться последним, замечу пока, что учетная запись (пользовательский бюджет, аккаунт) однозначно характеризует пользователя системы, хотя и не всегда соответствует именно пользователю как физическому лицу. И потому будут употребляться как (почти!) синонимы: надеюсь, что по прочтении *главы 10* в этот вопрос будет внесена ясность. Пока же достаточно запомнить, что пользователь — это не более чем метафора строки учетной записи в соответствующей базе данных.

Понятия процессов, файлов и пользователей с их атрибутами тесно (и причем рекурсивно) связаны между собой:

- пользователь запускает процесс, порождающий файл (файлы);
- атрибуты процесса определяются тем, какими атрибутами был наделен запустивший процесс пользователь;
- файлы наделяются своими атрибутами в силу атрибутов породившего их процесса;
- пользователь получает доступ к файлу в соответствии с атрибутами его и собственной учетной записи.

Так что начинать рассмотрение этих понятий можно с любой точки цикла. В настоящей книге за точку отсчета принято понятие процесса — именно с запуска процесса начинается функционирование FreeBSD после ее загрузки. Следует подчеркнуть только, что по ходу описания процессов будут упоминаться (пока без расшифровки) понятия и атрибутов файлов, и полей учетных записей пользователя, более детально рассмотренные в следующих главах.

Понятие процесса

Понятие процесса принадлежит к тем сущностям, кажущимся интуитивно ясными, которым, тем не менее, нелегко дать строгое (и при этом внятное) определение. Дабы не заниматься отсебятиной, позволю себе просто процитировать книгу Андрея Робачевского о UNIX (тем более, что она рекомендована компетентными в этом органами в качестве учебного пособия): "Мы можем ... представить процесс как программу в стадии ее выполнения, поскольку все выполняющиеся программы представлены в UNIX в виде процессов".[1]

Вместе с тем, далее в этой книге подчеркивается, что процесс и программа — далеко не идентичные понятия. Во-первых, существуют процессы, которым не соответствует никакой программы (т. е. исполняемого файла). Во-вторых, программа в ходе своего выполнения может породить более одного процесса. А в-третьих, и главных, при запуске программы порождение нового процесса (системный вызов fork) предшествует собственно запуску программы (системный вызов exec).

Так что, пожалуй, наиболее строгим определением процесса будет следующее: процесс — это то, что порождается системным вызовом fork. Каковой, в свою очередь, в лучших традициях UNIX-рекурсии можно определить как системный вызов, порождающий процессы.

Впрочем, с точки зрения пользователя, это не очень существенно. Хотя и ему не лишне знать, что существуют, с одной стороны, процессы системные

[1] Робачевский А. Операционная система Unix. — СПб.: БХВ-Петербург, 2000. — с. 39.

и прикладные, с другой — интерактивные и неинтерактивные. Первые порождаются ядром и не имеют соответствующих им программ (исполнимых файлов). Функция их, в первую очередь — управление виртуальной памятью, свопингом и т. д. К системным относят и процесс init. Это вызвано тем, что, хотя он и не является частью ядра, запускаясь из исполняемого файла, он является прародителем всех остальных процессов.

Забегая несколько вперед, замечу, что просмотреть список текущих процессов в системе можно с помощью команды `ps`, выполняющей для процессов ту же функцию, что и команда `ls` — для файлов. Данная в форме

```
$ps aux
```

она выведет на экран список процессов, начало которого будет выглядеть примерно следующим образом:

```
USER  PID  %CPU  %MEM  VSZ  RSS  TT  STAT  STARTED  TIME  COMMAND
Alex  247  0.0   0.0   444  240  v2  R+    5:32     0:00.00  ps aux
Root  1    0.0   0.1   552  316  ??  Ils   8:46     0:00.00  /sbin/init --
Root  2    0.0   0.0   0    0    ??  DL    8:46     0:00.00  (pagedaemon)
Root  3    0.0   0.0   0    0    ??  DL    8:46     0:00.00  (vmdaemon)
Root  4    0.0   0.0   0    0    ??  DL    8:46     0:00.00  (bufdaemon)
```

Первая строка в этом списке — сам процесс ps (благополучно, к слову сказать, завершившийся). Далее, за номером 1, идет процесс init, а системные процессы pagedaemon, vmdaemon, bufdaemon (об их назначении можно догадаться из названий) имеют номера от 2 и выше (что значат эти номера, будет сказано далее).

То, что эти процессы являются именно системными, подчеркивается отсутствием пути к исполнимому файлу в последнем поле соответствующих строк (в отличие от строки процесса init, где в том поле можно наблюдать абсолютный адрес исполнимого файла — /sbin/init). Впрочем, особо недоверчивые могут проверить это, запустив поиск (например, командой `find`) любого из таких файлов: уверяю вас, результат будет отрицательным.

Все прочие протекающие в системе процессы относятся к категории прикладных. Они разделяются на процессы интерактивные или неинтерактивные (очевидно, что все системные процессы — неинтерактивны). Первые, как правило, порождаются в рамках пользовательского сеанса работы, привязаны к определенному терминалу (реальному или виртуальному — тому, с которого они стартовали) и имеют своего хозяина, т. е. запустившего процесс пользователя. Например, во фрагменте списка процессов по команде `ps aux`

```
USER  PID  %CPU  %MEM  VSZ  RSS  TT  STAT  STARTED  TIME     COMMAND
...
```

```
alex  151  0.0   0.4   2484 2240 v0  I     4:53     0:00.21  -zsh (zsh)
alex  173  0.0   0.4   2484 2288 v1  I     4:56     0:00.21  -zsh (zsh)
...
```

приведенные выше процессы (запускающие отдельные экземпляры командной оболочки zsh) являются прикладными и интерактивными, принадлежащими пользователю alex и привязанными к виртуальными консолям первой и второй (v0 и v1 соответственно).

Неинтерактивные процессы именуются обычно демонами (daemon). Термин этот не несет в себе мистического смысла, представляя простую аббревиатуру (Disk And Execution MONitor). Они инициируются самой системой при ее загрузке обычным образом (запуском соответствующих программ) и функционируют в фоновом режиме, активизируясь при требовании соответствующей функции (печати, доставки почты и т. д.). Не будучи при этом привязанными к какому-либо терминалу, реальному или виртуальному. Приведенный ниже фрагмент вывода команды `ps aux` характеризует такие процессы-демоны (интернет-сервер inetd, демон расписания задач cron, демон консольной мыши moused и т. д.):

```
USER PID %CPU %MEM VSZ  RSS  TT  STAT STARTED TIME     COMMAND
...
root 71  0.0  0.1  1048 700  ??  Is   4:46    0:00.00  /usr/sbin/inetd
root 73  0.0  0.1  988  724  ??  Is   4:46    0:00.01  /usr/sbin/cron
root 79  0.0  0.1  912  520  ??  Ss   4:46    0:00.08  /usr/sbin/moused
```

Помимо процесса init, особое положение занимает и процесс getty, активизирующий виртуальные консоли. С одной стороны, это обычный прикладной процесс, т. к. он связан с исполнимым файлом (/usr/libexec/getty) и терминалом, на котором инициируется (что определяется файлом /etc/ttys). С другой стороны, он не запускается пользователем физически, а порождается процессом init. Однако он не является и демоном, поскольку прекращает свою жизнь после авторизации пользователя, как бы замещаясь ответственным за нее процессом login. Это можно проиллюстрировать фрагментами вывода команды `ps aux`. Сразу после загрузки системы в списке процессов можно видеть серию процессов getty:

```
USER PID %CPU %MEM VSZ  RSS  TT  STAT STARTED TIME     COMMAND
...
root 139 0.0  0.1  948  652  v0  Is+  6:17AM  0:00.00  /usr/libexec/getty
root 140 0.0  0.1  948  656  v1  Is+  6:17AM  0:00.00  /usr/libexec/getty
root 141 0.0  0.1  948  656  v2  Is+  6:17AM  0:00.00  /usr/libexec/getty
root 142 0.0  0.1  948  656  v3  Is+  6:17AM  0:00.00  /usr/libexec/getty
root 143 0.0  0.1  948  656  v4  Is+  6:17AM  0:00.00  /usr/libexec/getty
root 144 0.0  0.1  948  656  v5  Is+  6:17AM  0:00.00  /usr/libexec/getty
```

```
root 145 0.0  0.1  948  656  v6  Is+  6:17AM  0:00.00  /usr/libexec/getty
root 146 0.0  0.2  1200  896  v7  Ss  6:17AM  0:00.01  login -p root
...
```

Что же мы видим сразу после авторизации пользователя на первой виртуальной консоли? А вот что:

```
USER  PID  %CPU  %MEM  VSZ  RSS  TT  STAT  STARTED  TIME  COMMAND
...
root  139  0.0  0.2  1200  896  v0  Is  6:17AM  0:00.01  login -p alex
...
```

Из чего можно сделать вывод, что процесс getty на первой виртуальной консоли как бы подменился процессом login, сохранившим идентификатор первого. Вернее говоря, процесс getty вызвал к исполнению программу /usr/bin/login, которая, собственно, и осуществляет авторизацию пользователя (т. е. сверяет его пользовательское имя и пароль с соответствующими полями базы данных, о чем будет говориться в одной из ближайших глав).

Однако на самом деле — это тот же самый процесс, жизненный цикл которого продолжается на протяжении всего данного пользовательского сеанса. После выхода из которого (командой `logout`) процесс getty инициируется снова:

```
USER PID %CPU %MEM VSZ  RSS  TT  STAT  STARTED TIME  COMMAND
...
root 240 0.0  0.1  948  652  v0  Is+  7:15AM  0:00.00 /usr/libexec/getty
...
```

Но, как видно из вывода команды `ps`, уже — с другим идентификатором `PID`. Такая способность к "регенерации" процесса getty определяется в файле /etc/ttys, ответственном за доступность виртуальных консолей.

Атрибуты процесса

Атрибутами процесса, кроме его хозяина (строго говоря, хозяин процесса — это не более чем метафора, но об этом — чуть ниже) и, для прикладного процесса, терминала, о которых уже упоминалось, являются:

- идентификатор процесса и его родителя;
- его приоритет;
- реальный и эффективный идентификаторы пользователя;
- реальный и эффективный идентификаторы группы пользователей.

Для получения информации о протекающих в системе процессах и их атрибутах служит команда `ps`. Запущенная неким пользователем без опций, она

выдает краткую информацию о процессах текущего терминала, владельцем которых данный пользователь является, например:

```
PID  TT  STAT  TIME  COMMAND
216  v0  S  0:00.08  -csh (csh)
669  v0  R+  0:00.00  ps
621  v1  I  0:00.04  -csh (csh)
622  v1  I+  0:00.00  less /etc/ttys
189  v9  I  0:00.04  -csh (csh)
620  v9  S+  2:00.57  mp3blaster
```

В приведенном примере можно видеть, что данным пользователем (в качестве коего выступает ваш покорный слуга) с первой виртуальной консоли (значение поля TT — v0) запущены процессы (COMMAND) — оболочка csh и собственно команда `ps` (с идентификаторами `PID` 216 и 669 соответственно), со второй консоли (v1) — та же csh (`PID` — 621) и команда просмотра `less` (622), с десятой консоли (ve9) — опять же csh (`PID` — 189) и mpeg-проигрыватель mp3blaster (`PID` — 620). Процессы эти имеют различный статус (STAT), о чем я скажу несколько ниже.

Более полную информацию об атрибутах процесса можно получить, прибегнув к различным комбинациям опций команды `ps`, перечень которых — на странице экранной документации

```
$man ps
```

Опции эти весьма многочисленны, и рассматривать их подробно нет смысла. Замечу только, что они могут указываться различным образом — с предваряющим дефисом или без оного. Так, формы команды

```
$ps a
```

и

```
$ps -a
```

эквивалентны, обе они выведут информацию о процессах, запущенных всеми пользователями, а не только тем, кто дал указанную выше команду. Опция `u` позволяет получить дополнительные сведения о процессах, такие как имя запустившего пользователя, использование ресурсов процессора и памяти и т. д.:

```
USER  PID  %CPU  %MEM  VSZ  RSS  TT  STAT  STARTED TIME    COMMAND
Alv   283  0.0   0.1   452  264  v0  R+    8:59    0:00.00 ps aux
Root  1    0.0   0.1   552  316  ??  Ils   10:58   0:00.00 /sbin/init --
...
```

а опция `l` выведет вместо имени пользователя его уникальный идентификатор:

```
UID  PID PPID CPU PRI  NI  VSZ  RSS  WCHAN STAT TT  TIME    COMMAND
0    136 1    0   10   0   1200 964  wait  Is   v0  0:00.01 login -p alv
```

```
1000 166 136  0   18   0   1564 1240 pause I    v0  0:00.11  -csh (csh)
...
```

Рассмотрим подробнее основные атрибуты процесса не в порядке вывода команды `ps`, а в порядке их значимости.

Важнейшим из атрибутов является идентификатор процесса — `PID`, или Process IDentificator). Это — просто его уникальный номер в текущем сеансе работы (системы, не пользователя). Процесс с `PID=1` — это всегда процесс init, прародитель всех прочих процессов в системе. Порожденные им процессы получают следующие по старшинству номера, вплоть до достижения максимального значения, высчитываемого по специальным правилам, исходя из значения опции

```
maxusers 10
```

в файле конфигурации ядра. Номера процессов, прекративших свою деятельность (т. е. завершенных), остаются свободными вплоть до исчерпания лимита номеров процессов. После этого `PID` вновь запущенных процессов начинают принимать освободившиеся значения.

Идентификатор процесса — это именно тот атрибут, метафорой которого является понятие хозяина. При этом из рассмотрения вывода команды `ps` может показаться, что для всех процессов-демонов в качестве хозяина выступает пользователь root. На самом деле это не так — каждый из них имеет своего хозяина-псевдопользователя, имя которого, как правило, совпадает с названием исполняемого файла программы-демона. А имя пользователя root выступает в качестве псевдонима для всех таких квази-пользователей. Список этих псевдонимов приводится в файле /etc/aliases.

В момент зарождения нового процесса (да и потом — тоже) важным его атрибутом, как будет показано ниже, является идентификатор родительского процесса — `PPID` (Parent Process IDentificator), смысл которого интуитивно ясен. Идентификатор `PPID` для системных процессов, порожденных ядром, а также процесса init, всегда равен нулю — ведь процессов-родителей, как таковых, они не имеют. Каждый из прочих процессов имеет только один родительский процесс, но сам может инициировать несколько процессов-потомков.

Далее следуют атрибуты принадлежности процесса, отраженные в поле `USER` вывода команды `ps`. Как уже говорилось, понятие владельца, или хозяина процесса — это не более чем метафора. На самом деле под этим понятием скрываются два численных значения — реальный и эффективный идентификаторы пользователя процесса (`UID` и `EUID` соответственно). Первый совпадает с идентификатором пользователя, процесс запустившего (об этом будет говориться в разделе о пользователях и группах). Именно он в большинстве случаев определяет права доступа процесса к файлам, такие как возможность их открытия, изменения, записи и т. д.

Эффективный идентификатор пользователя обычно также совпадает с идентификатором пользователя, запустившего процесс (и, соответственно, с его реальным идентификатором). Однако в определенных случаях есть возможность, придав исполняемому файлу соответствующий атрибут (о чем — в разделе, посвященном файлам), дать процессу, запустившему такую программу на исполнение, права не пользователя, а владельца файла, каковым, как правило, является администратор системы (пользователь root).

В чем смысл такого действия? Наиболее простой пример этого — смена пользовательского пароля. Очевидно, что любой пользователь должен при необходимости иметь возможность изменить свой пароль доступа в систему, не обращаясь за этим к администратору. Однако сведения об учетных данных пользователя хранятся в определенных файлах (простых базах данных), владельцем которых является суперпользователь. И не менее очевидно, что давать к ним доступ обычным пользователям — мягко говоря, не очень разумно. Именно тут на помощь и приходит атрибут EUID. Благодаря ему пользователь, запустивший команду смены пароля (например, passwd), получает возможность чтения и изменения файла, в котором эти пароли хранятся. Правда, только на время исполнения этой программы и только в отношении своего собственного пароля.

Это легко проиллюстрировать, запустив (от лица обычного пользователя) команду

```
passwd
```

и (с другой виртуальной консоли) — команду

```
ps au
```

В выводе ее можно обнаружить строку

```
USER  PID  %CPU  %MEM  VSZ  RSS  TT  STAT  STARTED  TIME     COMMAND
...
root  451  0.0   0.1   1088 672  v0  I+    11:50    0:00.00  passwd
```

из которой видно, что хотя команда passwd была дана пользователем, хозяином соответствующего ей процесса (с PID 415) является root-оператор. Или, точнее, идентификатор пользователя процесса совпадает с таковым администратора, что можно видеть из вывода команды

```
ps al

UID PID  PPID  CPU  PRI  NI  VSZ   RSS  WCHAN  STAT  TT  TIME     COMMAND
...
0   451  166   0    3    0   1088  672  ttyin  I+    v0  0:00.00  passwd
```

Характерно, что в выводе команды ps без опции a (данной пользователем, запустившим программу passwd) процесса с PID 451 просто не обнаружится (как и команды passwd в поле COMMAND).

Подчеркнем, что возможность получения процессом привилегий, недоступных в обычной ситуации запустившему его пользователю, имеет смысл только в отношении пользовательских процессов, связанных с некими исполнимыми файлами. И зависит не от действий пользователя или атрибутов процесса, а исключительно от атрибутов файла, ответственного за запуск программы. И потому, хотя различение реального и эффективного идентификаторов процесса и может в определенных случаях представлять собой угрозу для безопасности системы, для реализации ее требуется именно некорректное функционирование программы, исполнимому файлу которой приданы соответствующие атрибуты.

Кроме реального и эффективного идентификатора пользователя, для процесса устанавливаются еще два атрибута принадлежности — реальный и эффективный идентификаторы группы (`GUID` и `EGUID` соответственно). Смысл их аналогичен, но они наследуют права доступа процесса для группы, к которой принадлежит запустивший процесс пользователь, о чем подробнее будет говориться в соответствующем разделе.

Следующий атрибут процесса — его относительный приоритет (NICE), обозначаемый по английски словосочетанием nice value (что интерпретируется обычно как степень "дружелюбия" или "тактичности" по отношению к другим процессам). Он варьирует в диапазоне от −20 (минимальное "дружелюбие", т. е. высший приоритет) до +20 (максимальное "дружелюбие", соответствующее низшему приоритету). Прикладные процессы в момент своего рождения получают приоритет 0 (т. е. некую "среднюю" степень "дружелюбия").

Относительный приоритет процесса лишь косвенно связан собственно с приоритетами выполнения процессов, которые перераспределяются системой динамически, в зависимости от ряда факторов (и относительный приоритет — лишь один из них). Относительный приоритет сам по себе остается неизменным на протяжении всего времени существования процесса, хотя, как будет показано ниже, может быть изменен принудительно.

Наконец, последний из упоминавшихся выше атрибутов процесса — терминал, с которым он связан. Атрибут этот имеет смысл только для прикладных процессов, запущенных пользователем с определенной консоли, реальной или виртуальной, от которой он и наследуется (хотя результаты выполнения процесса могут быть перенаправлены на другую консоль).

В результатах вывода команды `ps` с различными опциями можно видеть и еще несколько значений. Смысл большинства из них понятен — это задействованные ресурсы процессора и памяти, время зарождения процесса и продолжительность его исполнения и т. д. Остановимся чуть подробнее лишь на значении STAT, отражающем состояние процесса, т. е. степень его исполнения: процесс может быть исполняемым в данное время, находя-

щимся в режиме ожидания ввода, приостановленным, исполняться в фоновом режиме и т. д.

Кроме этого, существует понятие т. н. процесса-"зомби" — завершившегося дочернего процесса, от которого родительский процесс еще не принял сигнала окончания работы. По прошествии некоторого времени "зомбированные" процессы завершаются окончательно и исчезают из списка. Наконец, процесс может находиться в состоянии непрерываемого ожидания (uninterruptible sleep), обычно — ввода/вывода. Такой процесс не способен завершиться сам по себе, не реагирует на системные запросы (в том числе на команды принудительного завершения, к которым мы обратимся немного позднее) и может быть уничтожен только перезапуском системы.

Все приведенные сведения могут потребоваться при управлении запущенными процессами. Однако сначала рассмотрим цикл существования процесса — это не помешает нам в дальнейшем.

Жизнь процесса

Процесс в своем существовании проходит стадии зарождения, исполнения (часто с порождением новых процессов) и завершения (отмирания). Как все это выглядит в реальности, можно рассмотреть на примере самого первого пользовательского процесса, запускающего его командную оболочку по умолчанию (login shell). Для этого опять обратимся к выводу команды `ps` в форме

```
$ps al
```

для того, чтобы видеть идентификаторы родительских процессов (за их вывод и отвечает опция `l`).

Сразу после запуска этот вывод выглядит примерно следующим образом:

```
UID  PID  PPID  ...  TT   COMMAND
0    139  1     ...  v0   /usr/libexec
0    140  1     ...  v1   /usr/libexec
...
```

Можно видеть, что на каждом из виртуальных терминалов не запущено никакого процесса, кроме getty, имеющих следующие подряд идентификаторы и порожденные процессом init со значением `PPID`=1. После же авторизации пользователя на первой, для определенности, виртуальной консоли картина приобретает следующий вид:

```
UID   PID  PPID  ...  TT   COMMAND
0     139  1     ...  v0   login -p alex
1003  150  139   ...  v0   -zsh (zsh)
...
```

То есть процесс getty посредством команды `login` породил процесс zsh (получивший следующий по порядку `PID` и имеющий `PPID`, равный `PID` родительского процесса — 139).

Далее из командной строки нашей оболочки запускаем какую-либо программу, например, текстовый редактор joe, в котором пишутся эти строки. И картина опять меняется:

```
UID   PID  PPID  ...  TT   COMMAND
0     139  1     ...  v0   login -p alex
1003  150  139   ...  v0   -zsh (zsh)
1003  158  150   ...  v0   joe ch08.htm
...
```

Из чего может создаться впечатление, что процесс zsh непосредственно породил процесс joe, т. е. вызвал на исполнение соответствующую программу. Впечатление, нужно сразу сказать, ошибочное. К сожалению, проиллюстрировать динамику зарождения процесса не представляется возможным (по крайней мере, я не знаю, как), поэтому прошу поверить на слово — не мне даже, а авторам фундаментальных руководств по UNIX (некоторые из них приведены в дополнительных источниках).

На самом деле родительский процесс посредством системного вызова fork порождает свою собственную копию — в данном случае еще один экземпляр командной оболочки zsh (не потому ли первый Shell Борна и получил свое имя, что вызов программы из оболочки напоминает последовательную серию скорлупок?). И различаются в момент зарождения родительский и дочерний процессы фактически только своими `PPID` (ну и собственными идентификаторами, разумеется).

А далее уже дочерний процесс посредством вызова `exec` осуществляет запуск на исполнение собственно программы. Образом которой замещается в оперативной памяти образ родительского процесса.

В свою очередь, новый процесс также может выступать в качестве родительского. Так, редактор joe обладает способностью запуска из своей среды программ оболочки (как — будет рассказано в соответствующей главе). В образовавшемся имитаторе командной строки можно запустить, скажем, поиск файла командой `find`. Если в процессе поиска посмотреть распределение процессов (например) с другой виртуальной консоли), можно видеть, что процесс joe с `PID`=158 породил как бы свою копию — но уже с иными идентификаторами (при этом `PPID` копии, как и следовало ожидать, равен `PID` оригинала):

```
UID   PID  PPID  ...  TT   COMMAND
0     139  1     ...  v0   login -p alex
1003  150  139   ...  v0   -zsh (zsh)
```

```
1003  158  150  ...  v0  joe ch08.htm
1003  213  158  ...  v0  joe ch08.htm
```

По выходе же из редактора отвечающий за него процесс умирает, и в списке процессов первой консоли мы опять видим только указатель на авторизацию пользователя и его командную оболочку:

```
UID   PID  PPID  ...  TT  COMMAND
0     139  1     ...  v0  login -p alex
1003  150  139   ...  v0  -zsh (zsh)
```

как видно из их идентификаторов, те же самые процессы, что и до запуска редактора.

Управление процессами

Может возникнуть резонный вопрос — для чего все сказанное выше приведено в книге, адресованной конечному пользователю. Тем более что, признаюсь по секрету, изложение мое отнюдь не претендует на строгость и всеохватность. Прежде всего, иметь хоть какое-нибудь представление о процессах полезно во многих ситуациях — например, для понимания настройки командных оболочек. А главное, конечному пользователю подчас приходится прибегать к ручному управлению процессами — эта возможность неоценима в трудных ситуациях. Именно благодаря ей в UNIX-системах, в отличие от Windows, можно выйти с минимальными потерями из, казалось бы, безнадежно зависших программ.

Пользователь может управлять только теми процессами, владельцем которых является. Администратор системы же располагает правами на управление всеми запущенными процессами.

Управление процессами включает в себя изменение их приоритета и принудительное завершение. Все пользовательские процессы (и большинство системных) по умолчанию запускаются с равным, как бы промежуточным, относительным приоритетом 0. И, соответственно, при многих запущенных задачах ресурсы компьютера (процессорное время и объем оперативной памяти) распределяются между ними (более или менее) равномерно.

При необходимости перераспределения ресурсов между программами можно изменить приоритет выполнения какой-либо из них. При этом пользователь в состоянии только уменьшить приоритет одного или нескольких процессов, хозяином которых он является, администратор же имеет возможность и повысить приоритет любого процесса.

Делается это двояко. Если требуется запустить программу с приоритетом, отличным от обычного, используется команда `nice` с величиной изменения

nice value в качестве опции (предваряемой дефисом) и именем программы в качестве опции. Так, команда

```
$nice -5 joe
```

запустит редактор joe с приоритетом, уменьшенным на пять единиц. Если же опустить опцию, приоритет уменьшится на десять единиц. Для увеличения приоритета администратор (и только он) может дать команду

```
$nice --7 joe
```

что приведет к росту приоритета (т. е. уменьшению "дружелюбия") на семь единиц.

Кроме того, приоритет может быть изменен для уже запущенных процессов с помощью команды `renice`, параметром которой является новое значение приоритета, а аргументом идентификатор (`PID`) процесса. Например, команда

```
$renice 7 735
```

данная от лица пользователя — владельца процесса с `PID` 735, приведет к установке для него nice value, равного 7 (при условии, что прежний приоритет этого процесса был выше, например, 3 или 0). Администратор же может понизить приоритет того же процесса до значения nice value, равного 2, с помощью команды

```
$renice 2 735
```

или присвоить ему максимальный приоритет:

```
$renice -20 735
```

Необходимость ручного управления приоритетами возникает достаточно редко, а вот принудительное прерывание процесса — задача более обычная. Оно требуется, например, для выхода из безнадежно зависшей программы, не реагирующей на комбинации клавиш <Ctrl>+<C> и т. п. Или при невозможности закрыть окно программы в системе X Window штатными средствами управления.

Для таких случаев предназначена команда `kill`. В общем виде в качестве опции ее используется название сигнала или его номер, а аргументом служит `PID` процесса. Список сигналов команды получить с помощью

```
$kill -l
```

ответом на что будет

```
HUP INT QUIT ILL TRAP ABRT EMT FPE KILL BUS SEGV SYS PIPE ALRM TERM URG
STOP
TSTP CONT CHLD TTIN TTOU IO XCPU XFSZ VTALRM PROF WINCH INFO USR1 USR2
```

а расшифровку значений наиболее употребимых сигналов — получить по команде

```
$man kill
```

в следующем примерно виде:

```
1  HUP   (hang up)
2  INT   (interrupt)
3  QUIT  (quit)
6  ABRT  (abort)
9  KILL  (non-catchable, non-ignorable kill)
14  ALRM (alarm clock)
15  TERM (software termination signal)
```

Таким образом, как команда

```
$kill -TERM PID
```

так и команда

```
$kill -15 PID
```

предписывает завершить работу процесса с идентификатором (PID), указанным в качестве аргумента. Почти тот же смысл имеет и команда

```
$kill -KILL PID
```

или, в численной форме,

```
$kill -9 PID
```

Различие их в том, что по получении сигнала TERM (номер 15) программа по возможности пытается корректно завершить свою работу (с записью всех буферизованных данных), а сигнал KILL (номер 9) означает немедленное и неизбежное завершение процесса. Именно сигнал TERM посылается всем текущим процессам при завершении работы по командам `reboot`, `halt`, `shutdown`.

Значение сигнала команды `kill` по умолчанию — -TERM, и потому на практике для прекращения работы программы часто достаточно дать ее в виде

```
$kill PID
```

где аргумент, как уже говорилось, определяется с помощью команды `ps`. Более того, кроме внешней команды `/bin/kill`, одноименная команда встроена и в наиболее распространенные командные оболочки, например, bash и tcsh (о встроенных командах оболочек будет говориться в соответствующей главе *части II* книги). И потому вместо PID процесса можно использовать номер задания оболочки

```
$kill %#
```

где # (т. е. номер задания оболочки) определяется командой `jobs` (как это будет описано далее).

При уничтожении процессов следует помнить, что команда `kill` в отношении родительского процесса приведет к завершению и всех происходящих от него дочерних. А уничтожение процесса login shell автоматически завершит сеанс работы данного пользователя и выведет приглашение к повторной авторизации. Иногда это бывает единственным способом избавиться от безнадежно зависшего процесса. Однако данные во всех открытых в текущем сеансе программах будут, разумеется, потерянными.

Можно уничтожить и процесс init. Что, как легко догадаться, приведет к прекращению работы всех программ, закрытию всех пользовательских сеансов (в том числе — и root-оператора) и переходу машины в однопользовательский режим.

Еще один, весьма эффективный, способ отслеживания процессов и, при необходимости, управления ими — использование команды `top`. В отличие от команды `ps`, она, выведя на экран информацию о процессах, не завершает свою работу, а продолжает обновлять ее через промежуток времени, значение которого может быть установлено пользователем. Формат вывода информации также настраивается. Кроме того, возможно интерактивное управление процессами, не прекращая работы программы top.

Доступ к справке о возможностях программы осуществляется нажатием клавиши <h> или <?>. Основные из этих возможностей — следующие:

- <Ctrl>+<L> — перерисовка экрана;
- <F> — удаление и добавление полей, выводимых на экран; и то, и другое осуществляется нажатием символьной клавиши-переключателя для соответствующего поля;
- <O> — изменение порядка вывода полей; для этого также используются литерные клавиши, нажатие которых в верхнем регистре приводит к смещению соответствующего поля влево, в нижнем регистре — вправо;
- <C> — показ/скрытие полных путей команд в соответствующем поле;
- <K> — истребление процесса; после нажатия этой клавиши следует предложение сначала ввести `PID` процесса, а затем — номер сигнала (по умолчанию — 15, SIGTERM);
- <R> — изменение приоритета процесса, для чего сначала указывается его `PID`, а затем — новое значение приоритета;
- <U> — показывать процессы только определенного пользователя, имя которого вводится после нажатия этой клавиши; если имени не ввести, будут показаны процессы всех пользователей;
- <Q> — выход из программы.

Кроме того, изменяются такие параметры, как порядок сортировки (по PID, по возрасту, по использованию CPU и памяти), количество выводимых процессов, время (в секундах) обновления информации и т. д. Все сделанные изменения имеют силу в текущем сеансе. Однако их можно сохранить в файле ~/.toprc нажатием клавиши <W> (верхний регистр обязателен).

Дополнительные источники

- Все, что было описано в этой главе — не более чем элементарное (и скорее метафорическое, чем строгое) введение в предмет. Для углубленного понимания процессов следует обратиться к фундаментальным руководствам по UNIX. Из тех, с которыми я знаком, обращаю внимание на книгу: Робачевский А. Операционная система UNIX. — СПб.: БХВ-Петербург, 2000.
- С. Д. Кузнецов. Операционная система UNIX (есть, например, на **http://alone.dubna.ru:8080/history/contents.html**).
- Морис Дж. Бах (пер. А. В. Крюкова). Архитектура операционной системы UNIX (имеется, в частности, на **http://linux.perm.ru/doc/linux/bach/**).
- Интересные сведения о процессах (хотя и применительно к Linux) можно найти в статье Виктора Хименко "Процессы, задачи, потоки и нити" (Мир ПК, 2000, #5—6, **http://www.osp.ru/pcworld/2000/05/042.htm** и **http://www.osp.ru/pcworld/2000/06/054.htm**).
- И, наконец, почти исчерпывающее введение в предмет, и именно для FreeBSD — недавний цикл статей Станислава Лапшанского "Исследуем процессы" — **http://www.softerra.ru/freeos/18189/**, **http://www.softerra.ru/freeos/18735/**, **http://www.softerra.ru/freeos/18852/**.

Глава 9

Файлы и их атрибуты

Понятие файла и его атрибутов — второе, после процесса, ключевое понятие в FreeBSD. Разумеется, файлы есть и в любых других ОС, например, DOS или в Windows. Однако в FreeBSD (и в UNIX вообще) понятие файла используется в качестве универсального интерфейса для доступа не только к данным, но и к ресурсам системы, в том числе и физическим устройствам. Можно сказать, что если процессы — это все, что происходит в системе динамически, то файлы — это все, что существует в ней статически — исполняемые программы, созданные ими данные, дисковые и иные другие накопители, принтеры, терминалы, а также все прочие устройства.

Ввиду многообразия объектов, описываемых в UNIX термином "файл", они разделяются на несколько типов — обычные файла, каталоги, файлы устройств, специальные файлы, а также символические связи. Каждый их представитель имеет набор атрибутов. В DOS/Windows таковых три — скрытый, архивный, системный. В UNIX же важнейшими являются весьма многочисленные атрибуты доступа. Время создания и модификации также может рассматриваться как атрибут файла, причем смысл его тоже отличается от привычного по DOS/Windows.

Большинство этих аспектов — внутреннее устройство файлов, их классификация и атрибутика, — будут последовательно рассмотрены в этой главе.

Файлы и их имена

Каждый файл в FreeBSD (и UNIX-системах вообще) состоит из двух частей — области метаданных и собственно данных. Область метаданных по английски именуется inode; общепринятого перевода этого термина на русский мне не известно (иногда переводится на русский как "узел" или "i-узел"), поэтому ниже я буду использовать его без перевода.

В области inode содержатся такие сведения о файле, как его размер, формат, атрибуты (права доступа, время создания и модификации и т. д.), а также информация о физическом размещении на диске данных, составляющих файл. Следует обратить внимание, что имени файла нет ни в области мета-

данных, ни в данных: имя — атрибут не файла, а файловой системы. Взамен этого каждый узел имеет уникальный цифровой идентификатор, по которому и отыскивается программами.

Имена любых файлов хранятся в файлах специально предназначенного для этого типа — каталогах, которые в русскоязычной литературе по Windows обычно именуются папками. Однако для UNIX-систем первый термин предпочтительнее, и я сейчас постараюсь объяснить, почему.

Принятая в Windows (и идущая исторически от Mac OS X) метафора каталога как папки с документами способна создать впечатление (и подчас создает), что каталог — это некое физическое вместилище включенных в него файлов. Но это — не так. Каталоги — суть не более чем списки имен входящих в них файлов и их числовых идентификаторов (по которым собственно данные и находятся системой). И потому метафора библиотечного каталога (или книжного оглавления) как нельзя лучше отражает физический смысл соответствующего понятия в UNIX-системах.

Таким образом, имя файла в файловой системе UNIX существует только в составе каталога, и никак иначе. Собственно же содержимое файла ставится в соответствие имени через идентификатор его inode, подобно тому, как это осуществляется в реляционных базах данных. Механизм соотнесения имени файла, его метаданных и данных носит название жесткой ссылки (hardlink), или просто ссылки (link).

Из сказанного следует несколько важных свойств имен файлов. Во-первых, имя файла (т. е. атрибут включающего его каталога) может существовать только в той же физической файловой системе (дисковом разделе), что и inode, с которым он связан жесткой ссылкой (и, разумеется, в той же, что и собственно область данных файла).

Во-вторых, на один inode с его данными может быть создано произвольное количество жестких ссылок. То есть файл с одним и тем же физическим содержанием может выступать под рядом имен, и все они будут равноправны между собой. Редактирование же файла, вызванного под любым из этих имен (например, текста в текстовом редакторе) будет приводить к изменению области данных файла; не зависящему от имени, и, следовательно, содержимое всех таких файлов-двойников будет оставаться идентичным. Более того, любое из имен такого файла можно удалить (т. е. изменить содержимое включающего его каталога), что не окажет никакого влияния на остальные имена файла (и, тем более, на его реальное содержимое).

В-третьих, удаление файлов в FreeBSD происходит совершенно иначе, чем в DOS/Windows. А именно, файл считается удаленным, когда уничтожены все имена, ссылающиеся на идентификатор данного inode (т. е. файл исключен из файловой системы), и закрыта последняя программа, к нему обращающаяся (т. е. завершен процесс, загрузивший данные файла в память). Разу-

меется, сами по себе данные, составляющие содержание файла, физически могут продолжать существовать на диске, но для системы они уже недоступны. А поскольку содержание файла оторвано от его имени, восстановление случайно удаленного файла по фрагменту имени (на чем основаны DOS-утилиты типа UNERASE и UNDELETE) оказывается невозможным.

Пока любой файл открыт, т. е. существует ссылающийся на него процесс, он продолжает существовать, даже если имя его на диске стерто, и может быть записан, скопирован, переименован и т. д.

Подчеркнем, что два имени файла, связанных жесткой ссылкой с одним и тем же inode, не являются копией один другого, поскольку при копировании файла создаются новые области метаданных и данных (т. е. новый файл), которые в дальнейшем будут существовать абсолютно независимо друг от друга — т. е. изменение содержимого копии или ее атрибутов никак не затронет исходный файл.

Для просмотра идентификаторов файлов используется команда `ls` с опцией `-i` (от *inode*). С ее помощью можно убедиться, что два файла, расположенные в разных каталогах (или даже в одном), суть не что иное как две различные ссылки на один и тот же набор метаданных и данных. Так, например, в ответ на команду

```
$ls -il /bin/csh ; ls -il /bin/tcsh
```

последует вывод

```
142 -r-xr-xr-x  2 root  wheel  638700 28 янв 16:11 /bin/csh*
142 -r-xr-xr-x  2 root  wheel  638700 28 янв 16:11 /bin/tcsh*
```

из которого следует, что файлы, выглядящие внешне как исполняемые для двух разных командных оболочек — csh и tcsh, — имеют не только одинаковые размер, время создания, владельца, группу и атрибуты доступа, но и ссылаются на inode с одним и тем же идентификатором (142), т. е. являют собой два разных имени для одной и той же программы. К слову сказать, значение третьего поля вывода (цифра 2 после идентификатора и атрибутов доступа) показывает, сколько жестких ссылок связано с данным inode.

Создать жесткую ссылку можно командой `ln` с указанием имени исходного и целевого файла в качестве аргументов. Еще раз подчеркну, что оба эти имени выступают в качестве абсолютно равноправных — т. е. просто меток для некоего набора метаданных и данных. Для примера, в текущем каталоге мы имеем файл ./book.html (представляющий собой текст настоящей книги), атрибуты которого, определенные командой

```
$ls -il ./book*
```

следующие:

```
828807 -rw-r--r--  1 alv  alv  562268 24 мар 18:47 book.html
```

Обратите внимание, что в третьем поле вывода стоит цифра 1 — она показывает, что с данным идентификатором (828807) связана только одна жесткая ссылка: это — имя файла в текущем каталоге. Командой

```
$ln book.html book2.html
```

мы просто создаем второе имя (book2.html) для набора данных, составляющих содержание файла book.html, в чем тут же легко убеждаемся, дав повторно команду просмотра его свойств:

```
$ls -il ./book*
828807 -rw-r--r--  2 alv  alv  562268 24 мар 18:47 book.html
828807 -rw-r--r--  2 alv  alv  562268 24 мар 18:47 book2.html
```

Можно видеть, что идентификаторы и все прочие атрибуты обоих файлов идентичны, а количество ссылок на данный inode увеличилось до двух.

Наконец, особенность файловой системы FFS — то, что на имя файла накладывается весьма мало ограничений. Максимально допустимая его длина — 255 символов. Из них абсолютно запрещенными к использованию в именах файлов являются только символы прямого и обратного слэша (/ и \). Правда, и некоторые другие специальные символы, такие как !, @ и прочие из верхнего ряда клавиатуры, за исключением _, всякого рода скобки и кавычки, также не рекомендуются к использованию в именах файлов, особенно — в начальной позиции, но это, обычно, пожелание (а то и прямое запрещение) оболочки командной строки, а не файловой системы.

Кроме того, в FreeBSD нет расширений имен файлов в том смысле, в котором они используются в DOS/Windows. В общем случае файлу данных любого типа может быть приписано любое расширение, или их может не быть вовсе: на понимание его породившей программой это никак не отразится. Более того, файл может иметь несколько расширений (т. е. групп знаков, разделенных точками): типичный пример — файлы компрессированных архивов вида *.tar.gz.

Правда, некоторые программы (скажем, графические редакторы или офисные пакеты) все же требуют, чтобы файл формата TIFF имел расширение tif, и т. д. Но это вызывается тем, что имя файла неявно передается программе, т. е. запускающей ее команде, в качестве одного из аргументов.

Классификация файлов

Как уже говорилось, в FreeBSD (и в UNIX вообще) выделяются следующие типы файлов:

- обычные (regular) файлы;
- каталоги (directory);

- файлы устройств (devices);
- специальные файлы — сокеты (sockets) и именованные каналы (named pipes);
- символические ссылки (symlinks).

Смысл понятий обычного (regular) файла понятен пользователю, знакомому с файловой системой DOS/Windows. Это откомпилированные бинарные программы и интерпретируемые сценарии оболочки, конфигурационные ASCII-файлы, текстовые файлы, а также файлы данных, создаваемые прикладными программами в их собственных форматах — растровой и векторной графики, текстовых процессоров и т. д. Общее между ними то, что все они могут быть непосредственно просмотрены либо стандартными командами типа `cat`, `less`, `more`, либо специально предназначенными для этого программами.

Каталоги (directory), как уже поминалось, — это файлы, содержанием которых является информация о входящих в их состав файлах любого типа (в том числе и вложенных каталогах). Это не тавтология — такой информацией, в сущности, являются просто списки имен файлов, входящих в данный каталог, которым поставлены в соответствие идентификаторы их inode.

В сущности, единственное назначение каталогов — это хранение списков имен файлов всех остальных типов (впрочем, и имен подкаталогов — тоже). Однако это — очень важная функция: имена файлов, кроме как в составе каталогов, более нигде в системе не фигурируют. И, следовательно, файл, не принадлежащий ни к одному каталогу, для системы просто как бы не существует. Хотя, повторяю, удаление имени файла из состава каталога в общем случае не равнозначно физическому уничтожению информации о нем.

Файлы устройств соответствуют различным присутствующим в системе устройствам. Устройства эти, с одной стороны, могут быть реальными (жесткие диски, принтеры и т. д.) и т. н. псевдоустройствами, с которыми не ассоциировано никакого "железа" (например, пустое устройство /dev/null).

С другой стороны, выделяются файлы символьных устройств и устройств блочных. К первым возможен только последовательный (побитный, т. е. символьный) доступ. Примером их являются последовательные и параллельные порты, а также терминалы, реальные и виртуальные. К блочным устройствам можно осуществлять произвольный доступ, обмен информацией с ними осуществляется блоками фиксированной (реже — переменной) длины. Блочные устройства представлены, в частности, жесткими дисками и другими накопителями.

Файлы устройств идентифицируются своими номерами — основным (major), определяющим класс устройств (например, 12 — старший номер устройств, относимых к классу терминалов, реальных и виртуальных), и до-

полнительным (minor), который обычно является просто порядковым номером данного устройства в своем классе.

Задержим внимание на трех очень важных устройствах, которые некоторым образом занимают промежуточное положение между реальными и виртуальными устройствами — стандартным вводом (stdin), стандартным выводом (stdout) и стандартным выводом сообщений об ошибках (stderr). Они имеют старший номер 22, младшие же их номера — 0, 1 и 2 соответственно. В большинстве случаев для персонального компьютера в качестве таковых выступают просто клавиатура локальной машины и текущая виртуальная консоль. Однако ни один из файлов устройств стандартного ввода/вывода в общем случае не соответствует никакому из указанных устройств. Более того, и стандартный ввод, и стандартный вывод, и сообщения об ошибках могут быть перенаправлены в файл иного физического или виртуального устройства, или даже в произвольный файл обычного (regular) типа, например, в текстовый файл (подробнее об этом будет говориться в главах о системной консоли и, особенно, о командных оболочках).

Специальные виды файлов — именованные каналы (named pipe) и сокеты (socket), — предназначены для обмена данными между процессами. Они важны для разработчиков ПО, пользователь с ними, как правило, напрямую не общается. Однако знать о факте их существования полезно.

На символических ссылках (symlinks) следует остановиться подробнее. Это — отдаленные аналоги (и прародители) ярлыков в Windows или "теней" (shadow) в OS/2. Их следует отличать от упоминавшихся выше обычных, или жестких, ссылок. Последние не являются отдельным типом файла, а представляют собой нормальный способ организации связи между данными (вкупе с метаданными) файлов (причем — любых типов) и их именами.

Символические ссылки могут быть созданы командой

```
$ln -s имя_файла имя_ссылки
```

на файл любого из перечисленных выше типов. Это просто именованный файл, лишенный иного содержания, кроме указания на файл-источник, что можно определить по последнему полю вывода команды

```
$ls -s /usr/local/bin/tcsh
lrwxr-xr-x  1 root  wheel  9  5 апр 15:41 /usr/local/bin/tcsh ->
/bin/tcsh
```

из которого можно видеть, что файл /usr/local/bin/tcsh представляет собой символическую ссылку на файл /bin/tcsh. При явном обращении к символической ссылке действия (исполнение, просмотр и т. д.) осуществляются на самом деле с тем файлом, на который она ссылается. При этом файл-источник может находиться в другом каталоге, в другом разделе диска (partition — в приведенном примере каталоги /bin/ и /usr/local/bin/ располагаются в разных файловых системах, / и /usr соответственно), на другом слайсе или даже на другой машине.

Атрибуты файлов

Как уже говорилось, все файлы в файловой системе FreeBSD характеризуются набором атрибутов. Важнейшие из них — атрибуты принадлежности файлов и атрибуты доступа к ним. Именно их восприятие психологически наиболее сложно для Windows-мигранта, и поэтому им следует уделить особое внимание.

Атрибутов принадлежности файла — три. Во-первых, каждый файл имеет своего владельца (owner). Это, как правило (хотя и не обязательно), — пользователь, создавший его или скопировавший. Во-вторых, файл принадлежит группе пользователей (group) — причем не обязательно к одной из тех, в которые входит его владелец. И, наконец, все прочие пользователи (other), т. е. не являющиеся ни хозяином файла, ни членами группы, к которой файл приписан, также имеют некоторое отношение к данному файлу (и, соответственно, могут иметь некоторые права на него).

Атрибутов доступа — также три: право на чтение (read), право на изменение (write) и право на исполнение (execute). Причем права эти понимаются различно в зависимости от принадлежности файла к одному из типов, выделенных при их классификации.

Наиболее важно различие в атрибутах доступа к обычным (regular) файлам и каталогам (directory). Так, право чтения обычного файла означает возможность просмотра его с помощью команд типа `cat`, `more`, `less`, текстовых редакторов или специализированных прикладных пакетов. Кроме того, обладатель права на чтение может скопировать файл — разумеется, только в тот каталог, где он наделен правами записи (изменения).

Право на изменение позволяет изменить содержание файла, но не удалить, переместить или переименовать его — для этих операций требуется право изменения не файла, а каталога, в который он входит (о чем будет сказано ниже). В то же время отсутствие права на изменение данного файла не мешает его копированию — ведь при этом содержание исходного файла не претерпевает никаких изменений, т. к. создается новый файл, наследующий атрибуты не источника, а пользователя, запустившего процесс копирования.

Право на исполнение имеет смысл только для файлов исполняемых (опять рекурсия или, если угодно, тавтология), т. е. откомпилированных бинарных программ и сценариев оболочки. Бесполезно было бы устанавливать право на исполнение для нарративного текста или растрового графического изображения. В то же время именно это право (и только оно) отличает текстовый файл с листингом пользовательского сценария от собственно исполняемого сценария.

В отношении каталогов смысл атрибутов доступа иной. Право на чтение каталога означает возможность вывода его содержания (например, командой

`ls имя_каталога`), а также копирования каталога (в том числе и со всем его содержимым, если права доступа к последнему тому не противоречат). Однако права чтения для выполнения этих действий мало — необходимо еще право на исполнение (о чем ниже).

Право на запись для каталога — это возможность изменять его содержимое, т. е. записывать в него файлы или удалять их. При этом в последнем случае отнюдь не обязательно иметь права доступа на изменение удаляемого файла — ведь его содержимое не затрагивается, изменяется только содержимое каталога.

Наконец, право на исполнение в отношении каталога означает возможность перехода в него (командой `cd имя_каталога`) и последующего просмотра содержимого командой `ls` (при наличии права на чтение). Так что право исполнения и право чтения для каталога тесно сопряжены друг с другом — возможность перехода в каталог дает мало радости при отсутствии права его просмотреть. И обычно следует предоставлять для каталога или оба права, или ни одного. Тем не менее, права эти — разные, и иногда это может быть использовано для разграничения доступа.

Права доступа к существующим файлам могут быть просмотрены с помощью команды `ls` с параметром `-l` (от long). Рассмотрим их на примере:

```
$ls ~/work
-rw-rw-r--    2 alv      alv         72611 Июл 18 11:36 admin.txt
-rw-rw-r--    1 alv      alv         79326 Июн 24 16:51 install.txt
-rw-rw-r--    1 alv      alv         70578 Июл  4 09:41 qstart.txt
drwxr-x--x    3 alv      alv          4096 Июл 16 13:11 ris_admin
drwxr-x--x    5 alv      alv          4096 Июн 24 09:48 ris_install
drwxr-x--x   11 alv      alv          4096 Июл 13 17:19 ris_user
-rw-rw-r--    1 alv      alv        276559 Июл 14 19:31 user.txt
```

В этом списке за права доступа отвечают значения первого поля, содержащего десять символов. Первый из них определяет тип файла: обычный — `-` (символ дефиса), `d` — каталог, `l` — символическая ссылка и т. д., о чем уже говорилось.

Остальные девять символов разделяются на три равновеликие части. Первая (слева направо) определяет права доступа для владельца (owner), вторая — для группы владельцев (group), к которой файл приписан, и третья — для всех прочих (other). Порядок символов следующий — чтение (`r`, read), запись (`w`, write), исполнение (`x`, от eXecute). Присутствие любого из символов в соответствующей позиции каждой части означает наличие данного права, знак дефиса — отсутствие такового.

Так, в приведенном примере можно видеть, что для всех обычных файлов (опознаваемых по символу дефиса в первой позиции) его хозяин (alv, третье

поле записей) имеет право на чтение и запись, но не исполнение (сочетание символов `rw-`), те же права присвоены и членам группы (alv, четвертое поле записи). Все же остальные имеют только право на чтение файла (сочетание символов `r--`).

Иная картина будет для исполняемых файлов, например, пользовательских сценариев, что можно видеть на следующем примере:

```
$ls -l ~/bin
-rwxr-xr-x  1 alex  alex  123 29 мар 11:25 mapdos
-rwxr-xr-x  1 alex  alex  123 29 мар 11:24 mapwin
```

Здесь владелец файлов обладает всей полнотой прав — чтения, записи и исполнения (`rwx`), члены группы — право на чтение и исполнение, но не изменение (`r-x`), прочие же могут лишь запускать сценарии на исполнение, но ни изменить, ни просмотреть их не в состоянии (`--x`).

Вернемся, однако, к первому примеру и рассмотрим для него атрибуты доступа к каталогам:

```
drwxr-x--x    3 alv      alv          4096 Июл 16 13:11 ris_admin
drwxr-x--x    5 alv      alv          4096 Июн 24 09:48 ris_install
drwxr-x--x   11 alv      alv          4096 Июл 13 17:19 ris_user
```

Как и в случае с файлами, владелец имеет все права в отношении этих каталогов — право просмотра их содержимого (`r`), право удалять или записывать в них файлы (`w`) и право перехода в каталог (`x`). Права членов группы — уже, им разрешается просматривать каталоги и переходить в них (`r-x`). Наконец, за прочими есть только право исполнения, т. е. перехода в каталог: ни изменить его содержание, ни даже просмотреть его они не могут (`--x`).

Может показаться, что такой набор прав для пользователей лишен смысла: чтобы был толк от возможности перейти в каталог, следует иметь и право его просмотра. В данном примере это так и есть. Однако вернемся к примеру с пользовательскими сценариями. Если просмотреть права доступа к содержащему их каталогу, можно увидеть те же атрибуты доступа:

```
drwxr-x-x    2 alv      alv          4096 Июл  1 09:34 bin/
```

т. е. полный набор прав для доступа хозяина, возможность чтения и перехода для членов группы и лишь возможность перехода — для всех остальных. То есть пользователь, не являющийся хозяином каталога и не входящий в его группу, может перейти в этот каталог и запустить на исполнение любой из содержащихся там сценариев — как мы помним, такое право в отношении их ему дано. Правда, при условии знания их точного имени — принцип дополнения команды клавишей <Tab> для него не сработает ввиду запрета на чтение содержимого каталога.

Приведенная форма записи прав доступа называется символьной. Она не является единственной — существует еще т. н. абсолютная, или цифровая,

форма. Она выражается числом вроде 664, каждая цифра которого соответствует группе символов в символьной форме `rw-rw-r--`: правам владельца, группы владельцев и пр.

Образуются эти цифры простым суммированием прав доступа для каждого из уровней принадлежности, поскольку в абсолютной нотации каждому из прав доступа соответствует цифра: т. е. `-rw-rw-r--` в символьной нотации — это 011 011 100 в двоичной системе счисления, что в восьмеричном счислении и дает 664.

Легко подсчитать, что предоставление всех возможных прав доступа для владельца, группы владельцев и всех остальных выразится значением 777 (111 в каждой позиции), а отсутствие любых прав для них — значением 000. Ниже будет показано, что в одних случаях удобнее пользоваться символьной нотацией, в других — абсолютной.

Теперь следует поговорить о том, откуда берутся атрибуты доступа и принадлежности. Они возникают в силу создания файлов пользователем (вернее — запущенным им процессом, но в большинстве случаев, как было сказано в предыдущей главе, эти понятия совпадают). То есть файл, созданный пользователем alv, будет числиться за ним как владельцем, и принадлежать (по умолчанию) к основной группе, в которую тот входит.

Права доступа для файла определяются не правами пользователя, их создавшего, а правами запущенного этим пользователем процесса, породившего данные файлы — в общем случае, как говорилось выше, они могут и не совпадать. По умолчанию каждый создаваемый пользователем файл получает атрибуты доступа, определяемые командой `umask`. Формат ее следующий:

```
$umask 022
```

Аргумент команды и представляет собой маску прав доступа каждого вновь создаваемого файла. Значение цифр подобно таковым абсолютной нотации, но достигается не суммированием права чтения (4), изменения (2) и исполнения (1), а их вычитанием из цифры 7 (максимального значения прав в абсолютной нотации). То есть если считать, что без определения `umask` файл получал бы полные атрибуты доступа для всех атрибутов принадлежности (`777`), аргумент ее указывает, какие из "прирожденных" прав следует отнять.

Аргумент команды `umask` в приведенном примере означает, что для каждого вновь создаваемого файла будут устанавливаться права чтения, записи и исполнения для его владельца ($7 - 4 - 2 - 1 = 0$), и права чтения и исполнения для группы и всех остальных ($7 - 4 - 1 = 2$).

Приведенное значение аргумента `umask` по умолчанию (`022`) в FreeBSD определено глобально в файле /etc/login.conf в описании класса пользователя (default). При необходимости его можно изменить для отдельных пользова-

телей, создав дополнительный класс, или просто внеся соответствующее значение в их файлы конфигурации командной оболочки (для оболочки /bin/sh — в файл ~/.profile, для /bin/csh — в ~/.cshrc). Так, строка

```
umask 027
```

определяет, что по умолчанию любой создаваемый данным пользователем файл будет доступен хозяину для чтения, записи и исполнения, группе — только для чтения и исполнения, прочим же — недоступен вообще.

Впрочем, и атрибуты доступа, и атрибуты принадлежности файла не есть нечто неизменное. Владелец файла может легко сменить все права на доступ файла для самого себя, группы и прочих. Может он и назначить принадлежность файла другой группе, хотя и не любой, а только той, членом которой он сам является. Однако изменить владельца файла (т. е. назначить владельцем файла другого пользователя) он не имеет права. Это — прерогатива исключительно администратора, который располагает полномочиями изменить для файла все атрибуты доступа и принадлежности (как, впрочем, и почти все прочие атрибуты).

Для изменения владельца файла предназначена команда `chown` (от change owner). Она вводится в форме

```
$chown newowner file
```

где `newowner` — имя нового владельца файла `file`. Из сказанного выше ясно, что воспользоваться этой командой может только администратор. Пользователю же доступна команда для смены принадлежности группе chgrp (от change group):

```
$chgrp newgroup file
```

где `newgroup` — имя новой группы, которой будет принадлежать файл `file`. Как уже говорилось, для выполнения этого действия хозяин файла должен сам быть членом группы `newgroup`.

Изменить атрибуты принадлежности можно и одной командой `chown` в следующей форме:

```
$chown newowner.newgroup file
```

Для смены атрибутов доступа служит команда `chmod` (от change mode), где в качестве аргумента используется имя файла, а параметры определяют присвоение (`+`) или отнятие (`-`) соответствующих прав: чтения (`r`), записи (`w`) и исполнения (`x`). Например, команда

```
$chmod +rwx file
```

присвоит всем пользователям права на чтение, изменение и исполнение файла `file`, а команда

```
$chmod -w file
```

отнимет у всех (включая владельца) право изменять этот файл. Однако права можно присваивать или отнимать и избирательно для разных уровней принадлежности: для хозяина (u, от user), группы (g) и всех остальных (o — other). Для этого соответствующие символы (в любом наборе и сочетании) указываются слева от знака присвоения/отнятия: команда

```
$chmod ug+wx file
```

присвоит право изменения и исполнения файла хозяину и его группе, а команда

```
$chmod o-w file
```

отнимет право изменения файла всеми остальными. Кроме того, есть и форма

```
$chmod a+rwx file
```

присваивающая права доступа всем уровням принадлежности, что эквивалентно ранее приведенной форме без указания принадлежности вообще.

Как и большинство команд в FreeBSD, любая из указанных команд для смены атрибутов может использоваться рекурсивно, т. е. применительно к каталогам, всем входящим в них подкаталогам и составляющим их файлам. Так, команда

```
$chown newowner -R dir1/
```

сменит хозяина не только каталога dir1, но и всех входящих в него подкаталогов и файлов. То же справедливо и для команд `chgrp` и `chmod`.

Именно при рекурсивном исполнении проявляется некоторое неудобство символьной нотации атрибутов доступа. Следует подчеркнуть, что команда `chmod` изменяет только те права доступа, и только для тех уровней принадлежности, которые явно указаны в виде ее параметров. То есть команда

```
$chmod g-w -R dir1/
```

только отнимет право изменения каталога dir1 и всех его файлов у членов группы, не затронув прав доступа (которые в общем случае могут быть самыми разными для файлов и подкаталогов) владельца и остальных. Правда, есть и возможность "эксклюзивного" присвоения какого-либо права. Для этого используются параметры `=r`, `=w`, `=x`, которые устанавливают для аргумента (файла или каталога, в том числе и рекурсивно) только указанное право и никакое другое. Так, команда

```
$chmod g=r -R dir1/
```

присвоит группе право чтения каталога dir1 и его составляющих, одновременно отнимая все остальные права — записи и выполнения, а также просмотра данного каталога. Аналогично

```
$chmod a=x -R dir1/
```

для всех пользователей (хозяина, группы и прочих) присвоит исключительно право выполнения для структуры dir1, отнимая право на запись и чтение.

Однако именно в случае рекурсивного выполнения команды `chmod` удобнее может оказаться абсолютная нотация атрибутов доступа, поскольку при ней все они определяются одной командой для всех уровней принадлежности. Особенно эффективно использование абсолютной нотации администратором, т. к. он может в один прием унифицировать политику доступа к файлам всех пользователей вообще.

Файлы имеют и другие атрибуты, кроме атрибутов доступа и принадлежности. Остановимся на атрибутах, имеющих отношение к существованию файла во времени. Это — время доступа (Access Time), время модификации (Modification Time) и время изменения (Change Time), все они имеют разные значения. Характерно, что ни один из них не отражает время создания файла как такового:

- время доступа устанавливается при любом обращении к файлу — например, считыванию его прикладной программой;
- время модификации фиксируется в момент изменения содержания файла;
- время изменения — это время смены атрибутов файла, например, прав доступа.

Именно последний атрибут можно с определенной долей условности считать временем создания файла — в том случае, если за время его существования права доступа не изменялись.

Для изменения атрибутов времени файла применяется команда `touch`. Запущенная без параметров, с именем файла в качестве аргумента, она присваивает этому файлу атрибуты времени текущего момента. Если файл с таким именем не существует — он будет создан (пустым) этой командой.

С помощью параметров команды `touch` можно изменить время доступа (`-a`), модификации (`-m`), причем приписать им любое заданное время вместо текущего (`-d`), или заимствовать временные атрибуты у некоего файла (`-f имя_файла`).

В заключение — несколько слов о дополнительных атрибутах доступа. Один из них именуется "битом суидности" (что к суициду, впрочем, никакого отношения не имеет). В разделе о процессах упоминались их реальные и эффективные идентификаторы, и говорилось, что в большинстве случаев они идентичны. Однако в некоторых случаях процесс наследует идентификатор не пользователя, его запустившего, а исполнимого файла, инициировавшего процесс.

Это происходит в том случае, если соответствующий исполнимый файл имеет атрибут `SUID` (Set User IDentificator). Для такого файла символьной

нотации атрибутов доступа в поле владельца символом исполнения будет не x, как обычно, а s, например:

```
$ls -l /usr/X11R6/bin/XFree86
-rws--x--x  1 root  wheel  1736777 Jan 19 21:37 /usr/X11R6/bin/XFree86
```

Если же файл не является исполнимым, в соответствующей позиции можно было бы наблюдать символ S. Хотя установка такого атрибута для файла, не запускающего никакого процесса, по-видимому, лишена всякого смысла.

Присвоение атрибута "суидности" выполняется той же командой chmod. Например, для возможного случая, рассмотренного в разделе об установке XFree86, это будет выглядеть так:

```
$chmod u+s /usr/X11R6/bin/XFree86
```

Существует еще и атрибут SGID (Set Gropu IDentificator). Смысл его аналогичен атрибуту SUID, но применительно не к пользователю, а к группе, т. е. при его наличии символическая запись атрибутов доступа будет выглядеть как -rwx--s--x, если файл в принципе имеет атрибут исполнения, и как -rw-r-Sr--, если таковой отсутствует.

Последний из дополнительных атрибутов доступа — т. н. бит sticky ("липкий"). В символьной записи он обозначается как ********t при наличии атрибута исполнимости для other (и, теоретически, как ********T, но это, насколько мне известно, реального смысла не имеет).

Такой атрибут присваивается обычно каталогам, и влечет за собой невозможность удаления из него файла кем бы то ни было, за исключением владельца файла. Ведь в обычном случае возможность удаления файла определяется правами доступа не к нему, а к каталогу, в котором он находится. Это имеет смысл для каталогов типа /tmp, права доступа к которым по умолчанию позволяют всем пользователям записывать в них файлы и удалять. Однако удаление (например, случайное) файлов из такого каталога не их владельцем вряд ли желательно. И именно для предотвращения такой ситуации предназначен атрибут sticky.

Кроме того, атрибут sticky может присваиваться и исполнимому файлу. В этом случае он означает, что образ такого файла должен оставаться в памяти после завершения вызванной им программы. Согласно документации, это может быть полезно для ускорения доступа к часто используемым программам общего назначения. Однако, по свидетельству Ивана Паскаля, реальных примеров такого использования атрибута sticky не известно.

Глава 10

Файловые системы

Файлы в UNIX организованы в виде файловых систем. Термин этот понимается здесь в трех различных значениях:

- как часть ядра, которая управляет файлами — в этом смысле говорят о файловой системе (или подсистеме) ядра, наряду с системами (подсистемами) управления процессами и ввода/вывода;
- как физический способ организации хранения информации на диске (или ином накопителе) — при этом речь может идти о файловой системе FFS (Fast File System — файловая система FreeBSD, ext2fs (файловая система Linux), FAT16/FAT32 (файловая система DOS/Windows) и т. д.;
- как логическая иерархия файлов и каталогов, в которую они организованы.

Именно последний аспект и будет предметом настоящей главы — первые два в той или иной мере были затронуты ранее.

Логическая организация файловой системы

Если с именами файлов в FreeBSD допустимо весьма свободное обращение, то логическая организация файловой системы, т. е. структура каталогов, напротив, жестко фиксирована. Конечно, обладая правами суперпользователя, ее можно изменить. Но — делать это крайне не рекомендуется — в результате система может просто утратить работоспособность.

Структура каталогов в FreeBSD (как, впрочем, и в любой UNIX-системе) имеет иерархическую (древовидную) организацию, в основании которой лежит корневой (/, не путать с домашним каталогом администратора — /root) каталог. В качестве подкаталогов его выступают:

- /bin — каталог для исполняемых (иначе называемых двоичными, или бинарными, binary) файлов общего назначения; здесь помещаются оболочки командной строки, общие команды управления файлами и их архивации, традиционные текстовые редакторы типа vi и ее, и т. д.; именно

каталог /bin в первую очередь просматривается на предмет поиска введенной с клавиатуры команды;

- /boot, как явствует из названия, содержит файлы, отвечающие за загрузку системы;
- /cdrom — по умолчанию это точка для монтирования файловой системы соответствующего носителя;
- /compat — каталог, в котором размещаются файлы, обеспечивающие бинарную совместимость FreeBSD с иными операционными системами, например, Linux;
- /dev — каталог для файлов устройств;
- /dist — каталог, в который монтируется дистрибутивный CD-ROM при вызове программы sysinstall (и только в этом случае — обычная точка монтирования CD-ROM, как уже говорилось, — /cdrom);
- /etc — каталог для конфигурационных файлов общего пользования;
- /home — включает в себя домашние каталоги пользователей со всеми их программами, личными конфигурационными файлами (имеющими в сеансе данного пользователя предпочтение перед общими файлами конфигурации) и данными;
- /mnt — каталог для монтирования сменных накопителей (вроде дискет, Zip-дисков и т. д.) или временно подключаемых файловых систем (например, FAT-раздела диска);
- /modules — каталог, в который помещаются загружаемые модули ядра, собираемые при каждой перекомпиляции последнего; при этом модули, собранные при предыдущей компиляции, перемещаются в каталог /modules.old;
- /proc — виртуальная файловая система для чтения информации о процессах;
- /root — аналог каталога $HOME для суперпользователя; по умолчанию закрыт для просмотра всеми, кроме администратора;
- /sbin — содержит бинарные исполняемые файлы, используемые для системного администрирования;
- /stand — близкий по назначению каталог, файлы из которого, однако, доступны для исполнения, только будучи вызваны в явном виде, с указанием полного пути;
- /sys — символическая ссылка на каталог /usr/src/sys, содержащий исходные тексты ядра системы;
- /tmp — включает в себя всякого рода временные файлы — в частности, те, которые образуются при перекомпиляции ядра или полной пересборке системы;

- /usr — каталог для прикладных пользовательских программ со всеми их компонентами — исполняемыми, конфигурационными и разделяемыми файлами (/usr/bin, /usr/etc и /usr/share соответственно), библиотеками (/usr/lib) и т. д.;
- /usr/local — подкаталог, имеющий самостоятельное значение; именно в него устанавливается все дополнительное программное обеспечение из коллекции пакетов или системы портов, включая их исполняемые файлы (/usr/local/bin), документацию (/usr/local/doc, /user/local/info, /usr/local/man), библиотеки (/usr/local/lib), разделяемые файлы (/user/local/share);
- /var — каталог для часто меняющихся файлов: всякого рода системных журналов, почтовых и принтерных спуллингов и т. д.; кроме этого, в нем расположены базы данных устанавливаемых пакетов.

Каталоги /tmp, /usr, /var (а иногда и /usr/local) часто выделяются в самостоятельные файловые системы, расположенные в отдельных дисковых разделах (partitions). Как отдельную файловую систему имеет смысл располагать и каталог /home.

Управление файловыми системами

Как было сказано в *главе 4*, основные файловые системы, необходимые для функционирования системы, создаются при ее установке. Однако они могут быть созданы и позднее, например, при добавлении в машину нового накопителя. Как и обычно, это можно сделать различными способами. Основной из них — все та же универсальная программа /stand/sysinstall. Запустив ее (разумеется, с правами root-оператора, впрочем, от лица обычного пользователя она и не запустится), следует перейти в пункт **Configure** (Настройка) главного меню, а оттуда — в подпункт **Fdisk** (Создание слайсов). На появившейся панели выбора дисков выбирается диск, нуждающийся в создании (или перестройке) файловых систем; добавленный в систему диск появится в этом списке автоматически.

Далее накопитель разбивается на требуемое (в пределах возможных четырех) количество слайсов или, напротив, все его пространство отводится под один слайс. Порядок действий при этом полностью идентичен тому, что имел место при первичной установке системы *(см. главу 4)*.

По умолчанию для вновь созданного слайса устанавливается файловая система FreeBSD (номер 165). Но при желании можно создать и иные разделы, например, FAT (за номером 6) или Ext2fs (Linux, номер 131). Необходимо только помнить, что отформатировать дисковый раздел для иных ОС (т. е. собственно создать на нем соответствующую файловую систему) средствами FreeBSD нельзя, для этого придется использовать "родные" утилиты (FDISK для FAT под DOS/Windows и mkfs под Linux — для Ext2fs).

Закончив разбиение диска, необходимо сохранить сделанные изменения (нажатием клавиши <W>, в данном случае выхода через нажатие клавиши <Q> для этого недостаточно), выйти из программы fdisk и определиться с установкой начального загрузчика — очевидно, что для второго диска необходимости в нем нет, и потому на соответствующей панели можно смело выбрать пункт **None** (Нет).

После этого осуществляется выход в подменю **Configure** (Настройка) и переход к подпункту **Label** (Создание разделов) для создания разделов (partitions) и логических файловых систем на них (разумеется, в пределах физической файловой системы FreeBSD). При этом целесообразно придерживаться общего правила: разделы, к которым планируется более частое обращение, должны располагаться ближе к начальным секторам диска.

Разделы создаются точно как при инсталляции: дается команда создания раздела, задается его размер (в блоках или мегабайтах) и из возможных типов раздела выбирается пункт **File system** (Файловая система). После этого необходимо определить еще и точку монтирования (хотя реально она и не используется). Это, как уже говорилось, заранее созданный (и — пустой) каталог, к которому будут приписаны подкаталоги и файлы раздела.

Кроме того, для созданного раздела очень полезно активизировать опцию Soft Update (о смысле этого я скажу чуть позже). Впрочем, в версии 4.5 это делается по умолчанию для всех вновь создаваемых разделов, в версии 4.4 для этого потребуется просто нажатие одной клавиши (<S>).

Файл устройства, соответствующий вновь созданному разделу, будет иметь имя вроде /dev/ad1s1d, /dev/ad1s1e и т. д., поскольку, как было показано в *главе 2*, литеры a, b и c зарезервированы за корневой файловой системой, разделом подкачки и слайсом целиком соответственно. Обращаться к новому разделу можно сразу же после сохранения результатов разметки (нажатия клавиши <W>) и выхода из программы sysinstall, без перезагрузки, однако — только в текущем сеансе. В дальнейшем новый раздел придется или монтировать вручную (командой `mount`), или внести некоторые изменения в файл /etc/fstab, о чем будет сказано несколько позже.

В ряде случаев может возникнуть желание (а то и необходимость) создать на добавленном диске еще один раздел подкачки — во-первых, при увеличении объема ОЗУ (или, напротив, ярко выраженной его нехватке), во-вторых, для повышения производительности: говорят (хотя количественными измерениями этого никто не доказал), что распараллеливание своппинга на два устройства этому способствует.

По моему глубокому убеждению, сие есть баловство — увеличение объема памяти в любом случае даст эффект больший. Видимо, эту точку зрения разделяли и создатели FreeBSD, поскольку создание swap-раздела на добавленном диске несколько сложнее, чем раздела для данных. Если попытаться сделать это одновременно с созданием обычных разделов, без подготовки, —

при записи изменений последует сообщение об ошибке. В результате чего даже обычные разделы не будут созданы.

Поэтому следует сначала зарезервировать под swap-раздел пустое место — желательно в начале диска, повышения производительности ради. Сделать это можно разными способами — например, создав два раздела с нормальной файловой системой и уничтожив первый перед записью изменений (нажатием клавиши <D>). Затем следует выйти из программы sysinstall и внести изменения в файл /etc/fstab, о котором будет говориться в одном из следующих разделов. Забегая вперед, скажу по секрету, что это будет строка вида

```
/dev/ad1s1b  none  swap  sw  0  0
```

Повторяю, к рассмотрению устройства этого файла я еще вернусь, тогда и попытаюсь объяснить, что все это значит. А пока нужно перезагрузить машину (или реинициализировать систему иным образом, о чем говорилось в *главе 7*), вторично запустить /stand/sysinstall и уже теперь превратить пустое пространство в раздел подкачки, что достигается выбором соответствующего пункта в диалоговом окне **Type Partition** (Тип раздела). Далее — сохраняем изменения, выходим из sysinstall и убеждаемся, что раздел подкачки создан. Делается это командой `swapinfo`, которая при успехе должна вывести сообщение следующего примерно вида:

```
$swapinfo
Device    1K-blocks  Used  Avail     Capacity  Type
/dev/ad0s1b  1033120    0  1033120    0%    Interleaved
/dev/rar0s1b  1048448    0  1048448    0%    Interleaved
Total    2081568    0  2081568    0%
```

Если второго раздела подкачки в этом сообщении не видно — вероятно, его потребуется активизировать вручную:

```
$swapon /dev/file_name
```

Примерно таким образом эта процедура описана в официальном руководстве проекта (см. дополнительные источники в конце главы). Из пиетета перед коим, собственно, я и затратил на ее описание столько места. Однако абсолютно ничто не запрещает заблаговременно внести соответствующую запись в файл /etc/fstab, а уже затем вызывать /stand/sysinstall для создания разделов всех видов. По моему опыту — результат абсолютно тот же, никаких сообщений об ошибках при записи изменений не появляется.

Разумеется, создание разделов, как и вообще любая операция в FreeBSD, может быть выполнено и иным методом — непосредственно в командной строке программами fdisk и disklabel с соответствующими опциями. Однако он а) требует знания оных (весьма многочисленных, доложу я вам), б) при создании более чем одного раздела придется прибегать к не вполне прозрачным вычислениям и, главное, в) я им никогда не пользовался (хотя, вероятно, и обеспе-

чивает бóльшую гибкость). И потому позволю себе пропустить описание этого метода, отсылая читателя к соответствующей экранной документации (man fdisk и man disklabel), а также к дополнительным источникам.

Как уже говорилось, любой вновь созданный раздел становится доступным только после его монтирования. Термин "монтирование" часто упоминался без подробной расшифровки на протяжении предшествующего повествования. Настало время рассмотреть его подробнее.

Под монтированием понимается включение файловой системы дискового раздела (или иного носителя, в том числе и сменного) в структуру каталогов корневой файловой системы FreeBSD. Осуществляется это с помощью команды `mount` и ее разновидностей, и может быть выполнено в различных формах. Наиболее простая —

```
$mount /dev/file_name /mountpoint
```

где `/dev/file_name` — имя устройства с файловой системой FreeBSD (и только с ней), а `/mountpoint` — точка монтирования, т. е. существующий каталог. Каталог, предназначенный для монтирования, должен быть пустым. Впрочем, при монтировании в каталог, содержащий файлы, ничего страшного не произойдет. Просто его содержимое станет недоступным, его невозможно будет увидеть, например, командой `ls`: оно подменится содержимым смонтированной файловой системы. По размонтировании же последней содержимое каталога — точки монтирования снова станет доступным.

Программа mount не распознает тип файловой системы, отличный от FreeBSD. Поэтому при монтировании разделов или носителей с файловой системой других ОС тип ее должен быть указан в явном виде. Это можно сделать с помощью опции `-t`. Например, команда

```
$mount -t msdos /dev/file_name /mountpoint
```

смонтирует файловую систему FAT (VFAT или FAT32 монтируются посредством той же опции), а команда

```
$mount -t ext2fs /dev/file_name /mountpoin
```

смонтирует файловую систему Linux (Ext2fs), в том числе и в ее журналируемом варианте (Ext3fs).

Список доступных для монтирования файловых систем и их обозначения можно посмотреть в соответствующем экранном руководстве:

```
$man mount
```

Собственно говоря, опция `-t` самостоятельного значения не имеет: она просто вызывает специальную команду для монтирования соответствующей файловой системы. Каковая, разумеется, может быть дана и в явном виде, например, `mount_msdos` или `mount_ext2fs` соответственно (аргументы — имя файла устройства и точка монтирования остаются, естественно, теми же).

Та же команда, `mount` (или ее варианты для соответствующих файловых систем) используется и для монтирования сменных накопителей — флоппи-, Zip- и CD-дисков. В отличие от дисковых разделов, для них предусмотрены специальные точки монтирования — /mnt и /cdrom, хотя в большинстве случаев использование их не обязательно. Кроме того, следует помнить, что флоппи- и Zip-диски с большой вероятностью, а CD — наверняка, несут на себе отличную от FreeBSD файловую систему. В первом случае это, скорее всего, FAT, причем пространство Zip-диска фабричным образом размечается как четвертый "раздел", и соответствующий файл устройства выглядит как /dev/afd0s4 (при IDE-интерфейсе). Во втором же случае могут использоваться только диски с файловой системой ISO-9660: основанные на UDF системы, образуемые при пакетной записи дисков CD-R/RW (программами типа PocketCD и DirectCD) для FreeBSD пока, насколько мне известно, недоступны.

Проверить правильность монтирования (а заодно и получить информацию о доступных в текущий момент времени файловых системах) можно командой `mount` без опций и аргументов. До монтирования она выведет сообщение примерно такого вида:

```
/dev/ad0s1a  on  /  (ufs, local, soft-updates)
 /dev/ad0s1h  on  /home  (ufs, local, soft-updates)
/dev/ad0s1f  on  /tmp  (ufs, local, soft-updates)
/dev/ad0s1g  on  /usr  (ufs, local, soft-updates)
/dev/ad0s1e  on  /var  (ufs, local, soft-updates)
procfs    on  /proc  (procfs, local)
```

После же монтирования к этому списку должны добавиться новые строки, например:

```
/dev/acd0c  on  /cdrom  (cd9660, local, read-only)
```

в случае монтирования CD-диска, или

```
/dev/afd0s4  on  /mnt/zip  (msdos, local, noexec)
```

при монтировании Zip-накопителя.

Временно используемую файловую систему по миновании надобности следует размонтировать. Причем для сменных носителей это — обязательная процедура перед извлечением их из привода, отказ от нее чреват потерей данных. Впрочем, смонтированный CD-диск извлечь из привода и не удастся. Однако для флоппи-диска это делается легко — и нередко с печальными последствиями. Что же касается Zip-дисков, существует мнение, что монтирование блокирует возможность их извлечения. В общем случае это не так — проверено лично и многократно. Вполне вероятно, что это определяется особенностями конкретных чипсетов (вернее, их южных мостов), однако очень часто неразмонтированный Zip-диск может быть извлечен из при-

вода обычным способом. Это может повлечь за собой потерю данных, почему и требует внимательного отношения.

Размонтирование файловой системы осуществляется командой `umount` с указанием точки монтирования в качестве аргумента:

```
$umount /mount_point
```

При этом к файловой системе не должно быть обращений, иначе в ответ на попытку ее размонтирования последует сообщение об ошибке типа Devise busy. Под обращением к файловой системе понимается то, что ни один из ее каталогов не должен быть текущим. В то же время, скажем, открытие файла, находящегося на смонтированной файловой системе, из текущего каталога, лежащего за ее пределами (командой типа `less /mount_point file_name`), размонтированию ее не помешает. Ведь открытый файл представляет собой свою собственную копию, выведенную на текущее устройство вывода, и с файлом на диске никак не связан. Именно поэтому в FreeBSD в принципе не возбраняется одновременное редактирование одного и того же файла в различных программах и даже разными пользователями. Хотя некоторые программы блокируют вызов второго своего экземпляра или возможность изменения повторно вызванного файла.

Постоянно используемые файловые системы обычно монтируются при старте системы (как — будет сказано в следующем разделе). Однако и они при необходимости могут быть размонтированы (например, для проверки на целостность или для настройки). Например, если каталоги /usr или /home были при установке системы оформлены в виде самостоятельных файловых систем (что очень рекомендуется, особенно для второго каталога), они могут быть размонтированы командами

```
$umount /usr
```

или

```
$umount /home
```

соответственно. Разумеется, как сказано выше, при отсутствии к ним обращения. Впрочем, размонтирование возможно и в этом случае: придется только задействовать опцию `-f` (force). Однако на принудительно размонтированной файловой системе не исключено появление ошибок. Кроме этого, корневая файловая система не может быть размонтирована даже в принудительном порядке.

Можно размонтировать и все файловые системы, смонтированные в данный момент времени:

```
$umount -A
```

или все файловые системы, перечисленные в файле /etc/fstab (о котором — в следующем разделе):

```
$umount -a
```

Разумеется, в обоих случаях корневая файловая система размонтированию не подвергнется.

Опции `-A` и `-a` могут использоваться совместно с опцией `-t`, которая позволяет размонтировать файловые системы только определенных типов. Например, команда в форме

```
$umount -A -t msdos
```

размонтирует все смонтированные в данный момент FAT-разделы, а в форме

```
$umount -A -t msdos,cd9660
```

еще и диск CD-ROM (значения опции `-t` перечисляются через запятую, без пробела).

Наконец, все файловые системы, в том числе и на сменных носителях, автоматически размонтируются при корректном останове системы (комбинацией клавиш <Alt>+<Ctrl>+<Del>, командами `reboot`, `halt` или `shutdown`) — никакой порчей данных это не грозит. Диски аудио-CD, не содержащие никакой файловой системы, в монтировании и размонтировании не нуждаются.

Как монтирование, так и размонтирование файловых систем требует прав администратора. Однако последний может разрешить выполнение этих процедур и обычным пользователем. Правда, как сказано в официальной документации, это может представлять угрозу для безопасности, т. к. связано с изменением прав доступа к файлам соответствующих устройств. На локальной машине таких проблем нет, однако там нет и проблемы получения root-полномочий. Почему я и не буду останавливаться на этом вопросе, отсылая заинтересованных к сборнику часто задаваемых вопросов по FreeBSD.

Возвращаясь к команде `mount`, замечу, что она имеет большое количество опций, определяющих условия монтирования — синхронный или асинхронный режим записи, возможность (или, напротив, невозможность) размещения в файловой системе исполнимых файлов, средства для восприятия в именах файлов символов, отличных от чистой латиницы, и др. Подробно с ними можно ознакомиться на соответствующей странице экранной документации (`man 8 mount`). Некоторые из этих опций я рассмотрю в следующем разделе.

Автоматическое монтирование файловых систем

Как уже отмечалось выше, файловые системы, используемые постоянно, можно монтировать автоматически при старте системы. А для корневой системы это — обязательное требование, иначе функционирование системы просто невозможно.

Системы, предназначенные для автоматического монтирования, описываются в файле /etc/fstab. Как это в обычае в FreeBSD, он имеет простой текстовый формат и образован серией записей, разделенных на обязательные поля. Просмотреть формат и содержание файла можно командой `less` (или при помощи текстового редактора). Минимум, который в нем будет присутствовать, выглядит примерно следующим образом:

```
# See the fstab(5) manual page for important information on automatic
mounts
#of network filesystems before modifying this file.
#
#Device    Mountpoint  Fstype  Options    Dump  Pass#
/dev/ad0s1b  none     swap  sw     0  0
/dev/ad0s1a  /     ufs   rw     1  1
proc     /proc     procfs  rw    0  0
```

Эти записи соответствуют разделу подкачки, корневой файловой системе и виртуальной файловой системе /proc, служащей для взаимодействия с ядром. Однако, скорее всего, в файле /etc/fstab будут записи, соответствующие и другим файловым системам — например, /usr, /var, /home. Кроме того, в него целесообразно внести и записи для файловых систем сменных носителей — это очень упрощает процесс их монтирования. Например, в системе, в которой пишутся эти строки, файл /etc/fstab имеет следующий вид:

```
# Device      Mountpoint    Fstype    Options      Dump   Pass#
/dev/ad0s1b   none          swap      sw           0      0
/dev/ad0s1a   /             ufs       rw           1      1
/dev/ad0s1h   /home         ufs       rw           2      2
/dev/ad0s1f   /tmp          ufs       rw           2      2
/dev/ad0s1g   /usr          ufs       rw           2      2
/dev/ad0s1e   /var          ufs       rw           2      2
/dev/acd0c    /cdrom        cd9660    ro,noauto    0      0
/dev/afd0s4   /mnt/zip      msdos     rw,noauto    0      0
/dev/fd0      /mnt/floppy   msdos     rw,noauto    0      0
/dev/ar0s4    /mnt/disk     ext2fs    rw,noauto    0      0
proc          /proc         procfs    rw           0      0
```

На его примере и рассмотрим значение каждого поля и смысл отдельных записей.

Первое поле каждой записи — имя файла соответствующего устройства, второе — точка его монтирования. О них говорилось уже вдоволь, и возвращаться к этому не будем. Замечу только, что в данном примере точки монтирования для систем на сменных носителях и используемых время от времени, представляют собой подкаталоги каталога /mnt — это делает

возможным их одновременное использование (следует только озаботиться заблаговременным их созданием).

А вот третье — тип файловой системы. Для разделов FreeBSD он обозначается как ufs (UNIX File System), значение поля для сменных носителей понятно без комментариев. Кроме того, в этом поле могут фигурировать такие значения, как mfs (Memory File System — файловая система в памяти, нечто вроде RAM-диска в DOS), NFS (Network File System — сетевая файловая система фирмы Sun), ext2fs (файловая система Linux, см. предпоследнюю строку примера) и другие (большинство их требует включения соответствующих опций при компиляции ядра).

Четвертое поле записи — те самые условия монтирования файловых систем, которые могут задаваться и опциями команды `mount`. Значения `rw` в записях для разделов FreeBSD указывают, что монтируемая файловая система доступна для чтения и записи. Кроме того, по умолчанию подразумевается синхронный доступ к файловой системе (т. е. результаты файловых операций немедленно фиксируются на диске) и ее автоматическое (при загрузке) монтирование. Для изменений этого положения в соответствующие значения должны быть указаны в этом поле в явном виде.

Так, внесение значения `ro` (Read Only) приводит к блокированию возможности записи в файловую систему. Очевидно, что для дисков CD-ROM — это единственно возможный режим, что и отмечено в соответствующей записи.

Далее, значение `async` включает асинхронный режим доступа к файловой системе. При этом результаты файловых операций (в первую очередь копирования) не переносятся на диск немедленно, а некоторое время сохраняются в оперативной памяти. Этот режим дает прирост быстродействия при файловых операциях, и именно он принят по умолчанию, например, в ОС Linux. Однако он чреват потерей данных (и даже полным разрушением файловой системы). Кроме того, в FreeBSD есть иное средство для повышения производительности при работе с файлами, как будет показано в следующем разделе. И потому включение асинхронного режима нецелесообразно. По крайней мере, без веских к тому оснований.

Наконец, очевидно, что автоматическое монтирование бессмысленно для сменных носителей, и потому в соответствующих записях присутствует опция `noauto`. Которую, впрочем, можно указать и для систем на жестких дисках, используемых лишь время от времени (в примере это сделано для Linux-раздела физического диска, подключенного к контроллеру IDE-RAID).

Внесение в файл /etc/fstab записей для сменных носителей (или вообще временно используемых файловых систем) упрощает процесс их монтирования: после этого нет необходимости указывать ни полного имени файла устройства, ни типа файловой системы — достаточно задать только заранее определенную точку монтирования. Так, при записях, аналогичных приве-

денным в примере, для монтирования CD-ROM к желаемому результату приведет команда

```
$mount /cdrom
```

Выше говорилось, что после добавления в машину нового жесткого диска и создания на нем файловых систем FreeBSD последние должны быть смонтированы в корневую систему. Если нет желания после каждой перезагрузки выполнять эту процедуру вручную, следует внести изменения в файл /etc/fstab. Они сводятся к добавлению в него строки вида

```
/dev/ad1s1e  /mountpoint  ufs  rw  2  2
```

где `/dev/ad1s1e` — имя файла соответствующего устройства; `/mountpoint` — желательная точка монтирования раздела; `ufs` — тип файловой системы, а `rw` означает, что раздел доступен для чтения и записи.

Также следует поступить и при добавлении второго раздела подкачки. С той только разницей, что строка вроде

```
/dev/ad1s1b     none swap sw    0     0
```

должна быть добавлена до, а не после того, как соответствующий раздел будет определен (например, через `sysinstall`) как Swap Partition.

Настройка файловых систем

Файловая система FreeBSD замечательна своей устойчивостью и надежностью. Достаточно сказать, что при установках по умолчанию она почти не подвержена разрушению вследствие аварийного завершения работы (например, сбоев электропитания), что сплошь и рядом имело место быть с файловой системой Linux (по крайней мере, до появления журналируемых ReiserFS и ext3fs). Однако она существенно проигрывает последней в быстродействии, особенно при массовом копировании большого количества мелких файлов.

К счастью, в FreeBSD предусмотрено и средство для исправления положения. Имя ему — Soft Updates. Адекватный перевод его представляется затруднительным, виденные мной варианты типа "Мягкие обновления" трудно назвать удачными как с точки зрения русского языка, так и понимания существа дела. Так что пусть уж оно и выступает далее под своим английским именем...

В последних версиях FreeBSD режим Soft Updates можно включить еще при установке (на этапе создания дисковых разделов), а начиная с версии 4.5, он *включается по умолчанию для всех файловых систем, кроме корневой* (о причинах этого скажу в заключение раздела). Однако он может быть включен (или выключен — к последнему в некоторых случаях тоже есть резоны) и в любой иной момент времени. Кроме того, активизация режима может

потребоваться для новых файловых систем. Единственное, что для этого требуется — знать, как. И потому я остановлюсь на этом вопросе подробнее.

Итак, режим Soft Updates — это модификация кода файловой системы FreeBSD, UFS, ускоряющая ее работу и одновременно делающая файловую систему более устойчивой при аварийных сбоях. Он требует поддержки со стороны ядра системы, для чего в конфигурационном файле должна присутствовать опция

```
options SOFTUPDATES
```

Начиная по крайней мере с версии 4.2, она присутствует в файле конфигурации ядра (/usr/src/sys/i386/conf/GENERIC) по умолчанию. Однако, если режим этот не был включен при создании файловых систем средствами sysinstall (или вообще файловые системы создавались вручную), может потребоваться его активизация.

Активизировать режим Soft Updates можно только для размонтированных файловых систем. И тут самое время вспомнить, что на протяжении всего своего повествования я предупреждал о целесообразности создания отдельного корневого раздела (/) и самостоятельных разделов /usr и /home (и, возможно, других, типа /var и /tmp). Ведь если все каталоги расположены в единой корневой файловой системе, размонтировать их не удастся. Впрочем, даже если бы корневой раздел и удалось бы размонтировать (хотя я, честно говоря, не представляю, как), система, очевидно, полностью утратит работоспособность, и никакие дальнейшие манипуляции с ней будут невозможны.

А так — можно спокойно дать описанную выше команду

```
umount -Af
```

После этого начинается собственно процесс активизации. Для этого дается команда

```
tunefs -n enable /dev/ad0s1e
```

где `/dev/ad0s1e` — имя устройства, для которого требуется включить режим Soft Updates, например, /var. Далее команда эта повторяется для систем /usr, /home и т. д. После чего выполняем обратное монтирование всех файловых систем, определенных в файле /etc/fstab

```
mount -a
```

Наконец, снова запускаем команду `mount` без параметров и имеем удовольствие наблюдать, что на всех системах, кроме корневой, режим Soft Updates благополучно включен:

```
$mount
/dev/ad0s1a  on  /      (ufs, local)
/dev/ad0s1g  on  /home  (ufs, local, soft-updates)
```

```
/dev/ad0s1f  on  /usr   (ufs, local, soft-updates)
/dev/ad0s1e  on  /var   (ufs, local, soft-updates)
procfs on        /proc  (procfs, local)
```

Обращаю внимание на то, что для вступления изменений в силу не требуется перезагрузки системы. И, что характерно, после выполнения таковой изменения сохраняются. Так что в одном сеансе работы можно сколько угодно включать и выключать Soft Updates для любой файловой системы, кроме корневой.

Разумеется, режим Soft Updates можно активизировать и для корневой (/) файловой системы, или для единственного раздела (partition) в пределах слайса, буде таковым мы ограничились при инсталляции системы. Однако для этого нужно загрузиться с инсталляционного CD и в меню программы sysinstall выбрать опцию `fixit` (перед чем не худо озаботиться созданием соответствующей дискеты или наличием соответствующего CD из дистрибутивного комплекта). Поскольку при этом все имеющиеся на винчестере разделы оказываются размонтированными, команду

```
tunefs -n enable /dev/ad0s1*
```

можно применить к любому из них. В том числе и к единственному (или просто корневому) разделу. Так что, казалось бы, дробление диска не обязательно. Но все не так просто.

Конечно, сама по себе процедура перезагрузки не сложна. Однако при этом необходимо именно создание специальной дискеты fixit (или иметь диск fixit-CD): стандартный вариант резервной дискеты программу tunefs не содержит, и разместить ее там простым способом не удается. А главное, в этом случае исчезает возможность включать/выключать Soft Updates по желанию в течении одного сеанса работы. Что может понадобиться, например, для освобождения ресурсов системы или, напротив, для увеличения быстродействия файловых операций, в зависимости от сиюминутной задачи.

Что реально дает активизация Soft Updates? Это легко проиллюстрировать на примере копирования каталога, включающего вложенные подкаталоги с большим количеством различных по размеру (от пары килобайтов до 50—100 Мбайт), и последующего уничтожения копии. Результаты такого эксперимента для каталога с содержимым, суммарный объем которого превышает 4 Гбайт, приведены в табл. 10.1.

***Таблица 10.1.** Быстродействие операций над очень большим количеством различных по размеру файлов*

Операция	Soft Updates disable, мин	Soft Updates enable, мин
Копирование	15	9
Уничтожение	3	1

Можно видеть, что в результате нехитрого действа производительность FreeBSD при выполнении типичных файловых операций возрастает в полтора-три раза.

Еще более впечатляющая картина вырисовывается при действиях с большим количеством мелких (от первых килобайтов до 100—120 Кбайт суммарным объемом около 15 Мбайт. В этом случае (табл. 10.2) превосходство в быстродействии при включении Soft Updates — более чем четырехкратное при копировании и более чем на порядок — при удалении.

Таблица 10.2. *Сравнительное быстродействие файловых операций под FreeBSD и Linux*

Система	FreeBSD с SU	FreeBSD без SU	Linux
Копирование, с	8,5	30	5,7
Удаление, с	Менее 1	11	Менее 1

Легко видеть, что при включении режима Soft Updates быстродействие файловых операций в FreeBSD становится сопоставимым с таковым в Linux. Для полноты картины следует отметить, что результаты табл. 10.2 были получены на машинах, идентичных по всем компонентам, за исключением материнской платы. При измерениях для Linux в качестве таковой выступала Abit SE6 (чипсет i815e с поддержкой ATA/100). В системе на FreeBSD использовалась MSI 6326 (чипсет 815, поддерживающий только ATA/66). А учитывая традиционную ориентированность плат Abit на оптимизацию по быстродействию, различие в скорости копирования менее чем в 3 секунды нельзя считать значимым.

Таким образом, настройка должным образом файловых систем дает существенный выигрыш в быстродействии при операциях с данными. По эффективности я сравнил бы эту процедуру с использованием аппаратного кэширования дисков — помните, были некогда такие кэш-контроллеры, дававшие на некоторых задачах просто фантастическую скорость при обращениях к данным? Разница лишь в том, что тут такой выигрыш достигается а) абсолютно бесплатно и б) не требует никаких сверхъестественных действий (даже, скорее всего, регенерации ядра). Так что пренебрегать Soft Updates ни в коем случае не следует.

Осталось рассмотреть вопрос, почему же этот режим не включается по умолчанию для корневого раздела (и не включался автоматически в предыдущих версиях). Это обосновывается а) некоторой вероятностью повреждения данных в аварийных ситуациях и б) возможностью временного переполнения файловой системы.

Режим Soft Updates работает внешне подобно программному или аппаратному кэшу, при нем реальная запись на диск выполненных изменений может задерживаться ядром на промежуток времени до 30 секунд. Теоретически рассуждая, если в это время произойдет сбой питания, находящиеся в памяти данные будут потеряны или нарушена их целостность. Кроме того, возможно временное переполнение файловой системы, вследствие того, что файл, удаляемый для освобождения пространства на диске, еще не уничтожен физически, тогда как на его место уже претендует другой файл.

Обе ситуации представляются мне не очень жизненными именно применительно к корневому разделу. Если в качестве самостоятельных разделов обособить такие ветви файловой системы, как /var, /tmp, /usr (не говоря уже о /home), корневая файловая система будет подвергаться изменению только при перекомпиляции ядра (собственно, даже только в момент установки нового скомпилированного ядра), смене пароля, добавлении нового пользователя — все эти процедуры на пользовательской машине случаются достаточно редко.

Так что при наличии источника бесперебойного питания риском потери данных можно пренебречь, а эффект переполнения файловой системы для корневого раздела просто теряет силу. И потому режим Soft Updatess можно безбоязненно включать на любых разделах. Другое дело, что его использование в сочетании с асинхронным доступом к файловой системе несколько повышает риск потери (или повреждения) данных. И потому целесообразно ограничиться только им — не случайно создатели FreeBSD остановились на синхронном доступе по умолчанию.

В FreeBSD есть и другие, более хитрые, способы повышения производительности файловых операций. Один из них — создание файловой системы в памяти (MFS — Memory File System), другой — использование программного RAID-массива. Однако о них я говорить не буду за отсутствием собственного опыта — интересующиеся кое-какую информацию смогут найти в дополнительных источниках.

Дополнительные источники

- В качестве основных источников для более глубокого ознакомления с файловой системой UNIX можно рекомендовать те же руководства (Робачевского, Кузнецова, Баха), которые были указаны в предыдущих главах. Конкретно о файловой системе FreeBSD можно почитать в руководстве пользователя (Handbook) и сборнике часто задаваемых вопросов, на которые я уже неоднократно ссылался. А отдельным частным вопросам, затронутым в этой и предыдущей главах, посвящены следующие материалы:
- Маршал Кирк Маккьюзик, Грегори Р. Ганджер. Мягкие обновления. Методика, позволяющая избежать большей части синхронных операций за-

писи в быстрой файловой системе. Открытые системы, #07-08/1999 — **http://www.osp.ru/os/1999/07-08/13.htm**;

- Гилберт Гонг (пер. Станислава Лапшанского). Эффекты тонкой настройки. Софтерра On-line, 16.11.2001 — **http://www.softerra.ru/review/oses/linux/14083/**);
- Дуг Вайт. Форматирование носителей для использования с FreeBSD — **http://www.FreeBSD.org.ua/articles/formatting-media/**;
- Майкл Лукас (пер. Станислава Лапшанского). Трюки BSD — Memory File System — **http://www.softerra.ru/freeos/16111/**;
- Виктор Хименко. Файлы, файлы, файлы. Мир ПК, 2000, #2–3 — **http://www.osp.ru/pcworld/2000/02/064.htm** и **http://www.osp.ru/pcworld/2000/03/050.htm**.

Глава 11

Пользователи и группы

Понятие пользователей и их групп — еще одно из ключевых понятий в FreeBSD, предназначенное для разграничения доступа к файлам. А поскольку доступ к данным, устройствам и прочим ресурсам машины реализуется через универсальный интерфейс файловой системы, именно атрибуты пользователя определяют его возможности.

Пользователи

Следует сразу подчеркнуть, что понятие пользователя системы отнюдь не совпадает с пользователями ее в физическом смысле слова. Учетные записи могут существовать и для т. н. виртуальных пользователей, не соотносящихся ни с какими реальными лицами. В сущности, пользователь — столь же абстрактное понятие, которое может быть определено как запись в базе данных пользователей — учетная запись пользователя, именуемая также пользовательским аккаунтом или бюджетом.

База данных пользователей размещена в каталоге /etc, где присутствует как бы в четырех ипостасях, т. е. соотнесена с четырьмя отдельными файлами — /etc/spwd.db, /etc/pwd.db, /etc/master.passwd, /etc/passwd. Первые два имеют двоичный формат (т. е. недоступны для просмотра командами типа `less` или для изменения в текстовом редакторе), два последние — обычные текстовые файлы.

При этом содержимое пар файлов /etc/spwd.db—/etc/master.passwd и /etc/pwd.db—/etc/passwd идентично (за исключением формата). Файл /etc/spwd.db — это собственно полная база учетных записей со всеми их атрибутами, откуда последние и извлекаются при необходимости соответствующими программами. Файл /etc/master.passwd — просто ее представление в текстовом виде, он может быть просмотрен и модифицирован различными средствами, в том числе и просто текстовым редактором.

Содержание файла /etc/pwd.db аналогично таковому /etc/spwd.db, однако среди записанных в нем атрибутов пользователя отсутствует его пароль. Ну, а /etc/passwd — его текстовое представление, также не содержащее пароля.

Может возникнуть вопрос — почему так много файлов содержат сведения об атрибутах пользователя? С одной стороны, для удобства управления учетными записями. Разумеется, на сей предмет существуют специализированные утилиты, которые вносят изменения непосредственно в бинарные файлы /etc/*db. Однако в ряде случаев проще выполнить непосредственное изменение какого-либо атрибута просто в текстовом редакторе — именно для этого и служат /etc/*passwd. Например, при утрате пароля это может быть чуть ли не единственным способом получить доступ в систему.

Следует только помнить, что фактически-то база данных одна, и потому для сохранения ее целостности после редактирования одного из текстовых файлов требуется опять-таки запуск специальной утилиты, соответствующим образом модифицирующей и содержание всех остальных (в том числе и бинарных).

С другой стороны, наличие двух пар идентификационных файлов способствует повышению безопасности. Если просмотреть атрибуты этих файлов командой типа

```
$ls -l /etc/*wd*
```

можно увидеть примерно следующий вывод:

```
-rw-------  1 root  wheel   1424 Mar 30 19:35 /etc/master.passwd
-rw-r--r--  1 root  wheel   1210 Mar 30 19:35 /etc/passwd
-rw-r--r--  1 root  wheel  40960 Mar 30 19:35 /etc/pwd.db
-rw-------  1 root  wheel  40960 Mar 30 19:35 /etc/spwd.db
```

Из чего следует, что файлы /etc/spwd.db и /etc/master.passwd доступны даже для просмотра (не говоря уж об изменении) только root-оператору, тогда как два других файла (не содержащие паролей) могут быть просмотрены (хотя и не изменены) не только членами группы, но и кем угодно (символы `r` в полях `group` и `other`).

Пароли хранятся (даже в файле /etc/master.passwd) в односторонне-зашифрованном виде (т. е. дешифровать пароль в обратном направлении теоретически невозможно). Однако знание вида зашифрованного пароля может помочь при его подборе (методом грубой физической силы, т. е. перебором вариантов, например). И потому выставлять его на всеобщее обозрение было бы неосторожно.

Однако информация об атрибутах пользователя часто требуется программам, запущенным обычным пользователем. И потому полная база пользовательских аккаунтов дублируется своим урезанным вариантом.

Итак, воспользуемся этой возможностью и обратимся к файлу /etc/passwd на предмет понимания структуры учетной записи пользователя. Просмотр его содержимого показывает, насколько мало учетные записи могут иметь отношение к физическим лицам — пользователям системы:

```
root:*:0:0:Charlie &:/root:/usr/local/bin/zsh
toor:*:0:0:Bourne-again Superuser:/root
daemon:*:1:1:Owner of many system processes:/root:/sbin/nologin
operator:*:2:5:System &:/:/sbin/nologin
bin:*:3:7:Binaries Commands and Source,,,:/:/sbin/nologin
tty:*:4:65533:Tty Sandbox:/:/sbin/nologin
kmem:*:5:65533:KMem Sandbox:/:/sbin/nologin
games:*:7:13:Games pseudo-user:/usr/games:/sbin/nologin
news:*:8:8:News Subsystem:/:/sbin/nologin
man:*:9:9:Mister Man Pages:/usr/share/man:/sbin/nologin
smmsp:*:25:25:Sendmail Submission User:/
var/spool/clientmqueue:/sbin/nologin
mailnull:*:26:26:Sendmail Default User:/var/spool/mqueue:/sbin/nologin
bind:*:53:53:Bind Sandbox:/:/sbin/nologin
uucp:*:66:66:UUCP pseudo-user:/
var/spool/uucppublic:/usr/libexec/uucp/uucico
xten:*:67:67:X-10 daemon:/usr/local/xten:/sbin/nologin
pop:*:68:6:Post Office Owner:/nonexistent:/sbin/nologin
nobody:*:65534:65534:Unprivileged user:/nonexistent:/sbin/nologin
...
```

Мы видим здесь и администратора системы (root), и разнообразные системные сервисы, и многое другое. Однако для примера рассмотрим именно запись пользователя реального:

```
alv:*:1000:1000:russian:0:0:Alex Fedorchuk:/home/alv:/bin/csh
```

Каждая такая запись состоит из десяти полей, разделенных символами двоеточия без пробелов. Фактически не все эти поля (особенно для пользователей виртуальных) могут быть заполнены. Содержание этих полей следующее (слева направо):

- имя пользователя (name или username, не вполне точно называемое так же login);
- пароль (password); в файле /etc/passwd он заменен звездочкой, реальный же пароль (в виде неудобопонятного набора алфавитно-цифровых символов) находится в том же поле файла /etc/master.passwd;
- идентификатор пользователя (UID, User IDentificator) — уникальное число, по которому он и опознается при входе в систему (имя пользова-

теля предназначено исключительно для удобства — т. е. учитывает человеческий фактор);

- идентификатор группы (GID, Group IDentificator) — столь же уникальное число, определяющее принадлежность пользователя к основной группе; дополнительные группы, к которым пользователь может принадлежать, определяются в файле /etc/group (о чем будет сказано ниже);
- класс (class) — очень полезный атрибут пользователя, имеющийся только в BSD-системах; он позволяет определить (в файле /etc/login.conf) переменные среды, автоматически устанавливаемые для всех принадлежащих к данному классу пользователей при их авторизации; в приведенном примере принадлежность к классу `russian` приводит к установке русской локали;
- время изменения пароля (Password change time, в приведенном примере не заполнено) — дата, после которой пользователю выдается предупреждение о необходимости, пущей безопасности ради, изменить пароль;
- время прекращения пользовательского аккаунта (Account expiration time) — также не определенное в примере поле, где указывается дата, после которой пользователю не позволяется авторизоваться в системе; хотя сам по себе пользовательский аккаунт (включая его данные) сохраняется;
- общая информация (General information) — поле, в котором указывается реальное его имя и, при желании, иная произвольная о нем информация (обычно адрес, служебный и домашний телефоны);
- домашний каталог (Home dir) — подкаталог пользователя в каталоге /home, обычно совпадающий с его username;
- Shell — обычно полный путь к исполнимому файлу командной оболочки, запускаемой при регистрации пользователя в системе (т.н. login shell); однако в общем случае это может быть почти любая программа (интерпретатор языка программирования, файловый менеджер, даже текстовый процессор); единственное к ней требование — быть в явном виде указанной в файле /etc/shells.

Не все из этих полей обязательны к заполнению: в общем случае таковыми являются только username, UID и GID. Для реального пользователя практически необходимы также указание на пользовательскую оболочку (login shell) — без этого он, авторизовавшись в системе, не сможет выполнить никаких действий, и на домашний каталог — иначе ему некуда будет помещать свои данные. Отсутствие пароля, в принципе, допустимо для реального пользователя (и даже администратора), хотя очень не рекомендуется из соображений безопасности. Разного рода виртуальные пользователи могут не иметь никаких атрибутов, кроме имени и цифровых идентификаторов (UID и GID).

Группы

Понятие группы пользователей дополняет понятие пользователя. Каждый из них в обязательном порядке входит как минимум в одну, основную, группу — именно ее идентификатор является непременным атрибутом пользовательского аккаунта. Однако он может быть членом одной или нескольких других, дополнительных групп.

Данные о группах хранятся в файле /etc/group, имеющем простой текстовый формат и открытом для чтения всем (а для изменения — только администратору):

```
$ls -l /etc/group
-rw-r--r--  1 root  wheel  430 Mar 30 21:46 /etc/group
```

Если рассмотреть его, можно видеть, что, как и в случае с пользователями, большинство групп отнюдь не определение реальных объединений пользователей:

```
$less /etc/group
# $FreeBSD: src/etc/group,v 1.19.2.1 2001/11/24 17:22:24 gshapiro Exp $
wheel:*:0:root,alex
daemon:*:1:daemon
kmem:*:2:root
sys:*:3:root
tty:*:4:root
operator:*:5:root
mail:*:6:
...
```

Однако для рассмотрения структуры записей групп обратимся опять к реальному пользователю:

```
alv:*:1000:
```

Структурно файл /etc/group, подобно файлу /etc/passwd, представляет собой очень простую базу данных, образованную серией записей, разделенных символом : (двоеточие, без пробела) на поля. Первое поле — имя группы, однозначно связанное с ее идентификатором GID (Group IDentificator), занимающим третье поле (аналогично имени пользователя и его UID). Между ними, во втором поле, — пароль группы, который обычно не используется, почему поле это занято неким разрешенным символом — конкретно, *.

Четвертое поле записи о группе — список входящих в нее пользователей, для которых она не является основной. Как уже говорилось, каждый из них входит по крайней мере в одну группу — именно ту, GID которой стоит у него в соответствующем поле учетной записи и имя которой по умолчанию совпадает с именем пользователя. В этом случае четвертое поле записи о

группе будет пустым, как в последнем приведенном примера. Если же в данную группу входят еще какие-либо пользователи, они должны быть перечислены здесь явным образом:

```
alex:*:1003:alv
```

То есть членом группы alex является пользователь alv. Причем допускается использование как их имен (username), так и идентификаторов (UID).

Очень важна группа wheel — только ее члены имеют возможность временно получать права администратора командой `su`. И, соответственно, их имена (или идентификаторы) должны быть внесены в соответствующее поле:

```
wheel:*:0:root,alex
```

Если дополнительных членов группы более одного, их имена даются в строку, разделяясь запятой (без пробела).

Группы предназначены для дополнительного разграничения доступа к файлам — как в плане его расширения, так и ограничения. Первое используется чаше. Типичный пример — доступ к пользовательским файлам. По умолчанию некий user1 не только не имеет доступа к файлам user2, но даже не может просмотреть его домашний каталог (/home/user2) или зайти в него, и — наоборот. Так что обмен данными между ними невозможен.

Однако при необходимости такого обмена (а она возникает, например, при работе над единым проектом) выход есть. Для этого user1 достаточно сделать пользователя user2 членом своей группы (по умолчанию — user1) и присвоить своим файлам требуемые права доступа для ее членов (например, чтения и исполнения). Правда, для обратной связи user2 должен сделать user1 членом своей группы. Кроме того, по умолчанию в этом случае оба пользователя получают доступ (по крайней мере, для чтения и исполнения) ко всем файлам друг друга.

Поэтому может использоваться и другой способ, требующий вмешательства администратора. Он создает отдельную группу, не ассоциированную ни с одним из упомянутых пользователей, и делает их обоих ее членами. Те же должны каждый самостоятельно установить принадлежность к этой группе тех файлов и каталогов, которые требуются им для совместной работы, сохранив для прочих принадлежность к группе исходной.

Ограничение доступа на уровне группы обычно используется администратором и применяется в отношении не реальных, а виртуальных пользователей, таких как HTTP-, FTP- и почтовые клиенты, которые тоже должны иметь свои учетные записи. Типичный пример — создание т. н. "теневых" каталогов, доступ к которым закрывается некоей группе пользователей для чтения и исполнения, хотя доступ на чтение и (или) исполнение содержащихся в нем файлов сохраняется. В результате просмотреть файл или запустить программу в таком каталоге можно только при условии знания точного имени требуемого файла.

Управление пользователями и группами

Создание учетных записей пользователей и групп осуществляется администратором. В FreeBSD для этого существует множество средств. Начнем с создания пользователей, причем реальных — на проблеме виртуальных пользователей я останавливаться не буду.

Самое простое на сей предмет средство (и к тому же общее для большинства UNIX-систем, по крайней мере открытых — команда `adduser`, о которой достаточно подробно говорилось в разделе о начальных настройках *главы 4*. К сказанному остается добавить только, что она автоматически сама проверяет имя и идентификаторы пользователя на предмет уникальности, предлагая для последних подходящие значения (обычно — следующие по порядку за ID предыдущего зарегистрированного пользователя), создает домашний каталог пользователя и копирует туда "скелеты" конфигурационных файлов, после чего делает соответствующие записи в базах данных пользователей.

Использование `adduser` — наиболее автоматизированный, но и наименее гибкий способ. Основное его назначение — создание многочисленных учетных записей со сходными атрибутами и профильными файлами пользователей, "скелеты" которых заблаговременно создаются в каталоге /etc/skel.

Другая крайность — внесение всех данных пользователя в файл /etc/master.passwd вручную (в текстовом редакторе) и дальнейший запуск команды `pwd_mkdb`, которая в форме

```
$pwd_mkdb -p /etc/master.passwd
```

внесет соответствующие изменения в остальные три файла. При этом поле пароля следует оставить пустым, поскольку любой набор символов в нем развертыванию в реальный пароль не поддается. И в дальнейшем не забыть задать его специальной командой `passwd`. Кроме того, в реально многопользовательской (т. е. не настольной) системе следует быть уверенным, что никто другой в этот же момент времени не занимается изменением учетных записей — иначе трудно предсказать, чьи модификации возымеют силу. А заниматься этим может не только другой администратор системы, но и пользователь, изменяющий свой пароль или командную оболочку.

Далее, существует программа vipw, совмещающая оба процесса: вызов текстового редактора для внесения вручную изменений в файл /etc/master.passwd, а по выходе из него — модификацию баз пользовательских записей командой `pwd_mkdb`. В качестве редактора используется значение переменной `EDITOR` в профильном файле администратора. Если такового нет — вызывается классический редактор всех UNIX-систем — vi (откуда и название программы).

Наконец, самое мощное средство для управления учетными записями пользователей вообще — команда `pw`. Это — специфичная для BSD-систем, уни-

версальная утилита администрирования учетных записей пользователей и групп — их создания, изменения и удаления юзеров и групп. Для создания нового пользователя ее проще всего запустить в форме

```
$pw useradd username
```

после чего она сама подберет подходящие (т. е. ближайшие по порядку свободные) идентификаторы пользователя и группы, поля общих сведений, домашнего каталога и командной оболочки заполнит некими значениями по умолчанию (User &, /home/username, /bin/sh соответственно), прочие же оставит пустыми. Сам домашний каталог она тоже не создаст — для этого она потребует формы

```
$pw useradd username -m
```

во исполнение которой заодно будут скопированы и "скелетные" файлы.

Кроме этого, команда `pw` может создавать учетную запись пользователя с заранее предопределенными атрибутами, описанными в файле /etc/pw.conf. Правда, для этого такой файл необходимо предварительно создать — командой

```
$pw useradd -D
```

В первозданном виде он будет содержать все те же сведения по умолчанию. Однако его легко отредактировать в текстовом редакторе для внесения всех необходимых данных, которые отныне и будут выступать в качестве "умолчальных" для команды `pw`. К слову сказать — учетные записи ранее созданных пользователей могут быть приведены в соответствие с новой их схемой с помощью команды

```
$pw useradd -D username
```

Ничто не мешает и задать несколько таких конфигурационных файлов (например, /etc/pw1.conf, /etc/pw2.conf и т. д.) для создания учетных записей пользователей разных категорий (например реальных или виртуальных, или пользователей с различными значениями атрибута `Class`). Приведение конкретной учетной записи к какой-либо из принятых схем после этого можно выполнить командой

```
$pw adduser username -C /etc/pw1.conf
```

Она создает учетную запись нового пользователя с именем `newuser` и его домашним каталогом /home/newuser, в который копируются конфигурационные файлы командной оболочки (по умолчанию — /bin/sh). Файлы, подлежащие копированию, определяются содержимым каталога /etc/skel и могут быть отредактированы администратором в текстовом редакторе.

Многие атрибуты учетной записи пользователя, будучи раз созданными, могут быть изменены в дальнейшем, причем некоторые — и самим пользователем (не только root-оператором). Наиболее часто такая необходимость возникает в отношении пароля. Во-первых, как уже было сказано, не всегда

он задается при создании учетной записи. Во-вторых, менять пароль время от времени рекомендуется и из соображений безопасности.

Для смены пароля предназначена команда `passwd`. Запущенная от лица обычного пользователя без параметров и аргументов, она запросит для начала старый его пароль (если такового не было, этот шаг, естественно, пропускается), затем — новый пароль и его повторение (во избежание ошибки ввода).

Администратор системы с помощью команды

```
$passwd username
```

может поменять пароль любого пользователя. Старый пароль его при этом не запрашивается — да, в общем случае, администратор его и не знает (и восстановить из базы пользовательских записей никаким образом не может).

Далее, пользователь может изменить еще два атрибута — общие сведения о себе и командную оболочку (login shell). Для этого он вправе использовать команду `chpass`, которая вызовет в текстовом редакторе (определенном переменной `EDITOR` профильного файла пользователя, за отсутствием таковой — в редакторе vi) следующие поддающиеся изменению сведения:

```
#Changing user database information for alv.
Shell: /bin/csh
Full Name: Alex Fedorchuk
Office Location:
Office Phone:
Home Phone:
Other information:
```

По завершении их модификации и выходе из редактора внесенные изменения будут зафиксированы в базах данных автоматически. Нетрудно догадаться, что для этого команда `chpass` должна иметь упоминавшийся выше атрибут "суидности", в чем можно убедиться посредством:

```
$ls -l /usr/bin/chpass
-r-sr-xr-x  6 root  wheel  32324 Jan 28 16:12 /usr/bin/chpass
```

Другие изменения учетной записи требуют вмешательства администратора, который может обратиться к той же команде `chpass` с указанием имени пользователя как аргумента:

```
$chpass username
Login: alv
Password: lhBESi9NHbsrg
Uid [#]: 1000
Gid [# or name]: 1000
Change [month day year]:
```

```
Expire [month day year]:
Class: Russian
Home directory: /home/alv
Shell: /bin/csh
Full Name: Alex Fedorchuk
Office Location:
Office Phone:
Home Phone:
Other information:
```

Можно видеть, что root-оператору предоставлено гораздо больше возможностей для изменения. Что, прочем, не значит, что он может использовать их все. Так, изменение идентификатора пользователя эквивалентно созданию пользователя нового, а изменение имени просто поставит ему в соответствие тот же идентификатор. Ни то, ни другое не вызовет ничего, кроме путаницы. Идентификатор основной группы, вероятно, изменить можно, хотя трудно представить необходимость этого. Ну, а о пароле уже говорилось достаточно.

Так что фактически из недоступных пользователю полей изменению подлежат только класс и временные атрибуты аккаунта. Почему последние подведомственны администратору — понято. Отношение же к классу выведено из сферы компетенции пользователя, т. к. через его определение возможен очень широкий круг настроек, в том числе и связанных с разделением доступа, степенью безопасности и т. д.

Теоретически для изменения учетных записей в разных UNIX-системах применяются также команды `chfn` и `chsh`, но в FreeBSD эти файлы — лишь жесткие ссылки на тот же `inode`, что и `chpass`, и потому вызов любой из них автоматически влечет запуск последней.

Внести изменение в учетную запись можно и командой `adduser` с соответствующими опциями, например:

```
$adduser -shell shell_name
```

для изменения командной оболочки,

```
$adduser -class login_class
```

для изменения класса пользователя и т. д. — список возможностей можно получить по команде

```
$adduser --help
```

Кроме этого, для изменения учетных записей может применяться и универсальная команда `pw` с параметром `usermod`, именем пользователя и соответствующими опциями, с которыми можно ознакомиться посредством

```
$man 8 pw
```

Удаление учетной записи пользователя, на первый взгляд, задача очень простая: достаточно удалить соответствующую строку из файла /etc/master.passwd и запустить команду `pwd_mkdb` для синхронизации изменений, точно так же, как это делалось при создании аккаунта. Или прибегнуть к программе vipw, которая сделает это сама. Однако и в том, и в другом случае будут удалены только сами учетные записи, а пользовательские каталоги, почтовые ящики, временные файлы, результаты всякого рода протоколирования действий (log-файлы) благополучно сохранятся, причем будут рассеяны по ряду каталогов (кроме /home, в основном, также по /tmp и /var). Кроме того, упоминания имени удаленного пользователя останутся в списках принадлежности к группам, конфигурационных файлах авторизации, систем доставки почты, почтовых и FTP-клиентов (и в иных местах). Что может вызвать проблемы, если в дальнейшем будет создан пользователь с тем же идентификатором и (или) именем.

Потому для бесследного истребления следов жизнедеятельности удаленного пользователя их следует сначала отыскать. Все принадлежащие ему файлы могут быть найдены командой `find`:

```
$find / -user username
```

А добавив в конец строки опцию `-delete` (или перенаправив вывод по конвейеру команде `rm`) — и немедленно уничтожить их. А для поиска упоминаний имени в конфигурационных файлах можно прибегнуть к команде `grep`:

```
grep -e 'username' /*
```

Однако ради чистоты исполнения в обоих случаях поиск должен проводиться, начиная с корневого каталога (о чем сигнализирует символ / в аргументах обоих команд). И может занять немало времени.

Поэтому может оказаться целесообразным прибегнуть к команде `rmuser` для автоматизации процесса. В форме

```
$rmuser username
```

она удалит не только учетную запись из базы данных, но также и домашний каталог, почтовый ящик, имя в списках принадлежности групп (/etc/group) и еще некоторых конфигурационных файлах.

И конечно, ту же работу в состоянии сделать и палочка-выручалочка администратора в форме

```
$pw userdel username -r
```

где следует обратить внимание на опцию `-r` — без нее `pw` удалит только пользовательскую запись из базы аккаунтов.

Разумеется, существуют и средства для управления записями групп. Как уже говорилось, новая группа автоматически создается при заведении нового пользовательского аккаунта. По умолчанию ее имя и численное значение идентификатора совпадают с именем и идентификатором пользователя.

И для последнего эта группа в дальнейшем остается основной — т. е. только о ней содержится информация в учетной записи пользователя.

Однако пользователь может принадлежать и к иным, дополнительным, группам: для этого его имя или идентификатор должен быть приписан к списку таких групп в файле /etc/group. Это может сделать администратор в любом текстовом редакторе (никаких дальнейших действий по синхронизации изменений не потребуется).

В полномочиях root-оператора также и создание новых дополнительных групп — просто добавив строку по образу и подобию любой существующей, записав в последнее ее поле имена тех пользователей, которые должны быть ее членами. Тем же способом группа может быть и удалена.

И конечно, для управления группами можно применить универсальное средство — `pw`. В форме

```
$pw group add groupname
```

она создаст новую группу,

```
$pw group del groupname
```

удалит существующую, а посредством

```
$pw group mod groupname -m username1 username2
```

или

```
$pw group mod groupname -M username1 username2
```

дополнит или полностью перепишет списки членов группы соответственно. Наконец, с помощью

```
$pw usermod  username -G groupname
```

можно приписать пользователя к указанной группе в индивидуальном порядке.

В прежних версиях FreeBSD имелись еще специальные утилиты для создания (`addgroup`) и удаления (`rmgroup`) групп. Однако в текущих (4.xx) версиях я не нашел даже их следов.

Дополнительные источники

Лучшее описание всех вопросов, связанных с пользователями, группами и управлением ими, можно обнаружить на сайте Ивана Паскаля (**http://www.tsu.ru/~pascal/**) и в соответствующих разделах руководства по FreeBSD Андрея Федорова (**http://www.anriintern.com/computer/freebsd/** или **http://www.citforum.ru/operating_systems/freebsdc/index.shtml**), в значительной мере на нем базирующемся. Очень полезные сведения о пользовательских аккаунтах — на сайте Андрея Лаврентьева (**http://unix1.jinr.ru/~lavr/**).

Глава 12

Системная консоль FreeBSD

Системная консоль — это то, что в первую очередь предстает перед глазами пользователя после загрузки FreeBSD. Она являет собой зримое воплощение консольного драйвера, управляющего стандартными устройствами ввода и вывода, специфичного как для данного "железа", так и для операционной системы.

Таким специфичным драйвером для PC-версии FreeBSD является программа под именем syscons. Хотя, в принципе, в FreeBSD можно использовать и другой драйвер, pcvt (до недавнего времени по умолчанию применявшийся также в OpenBSD). Однако, как правило, каждая из UNIX-подобных систем наиболее эффективно работает с собственной системой управления консолью. И потому в этой главе речь пойдет исключительно о syscons.

Возможности консольного режима

Итак, консоль, как уже говорилось во вводной главе, — это совокупность стандартного устройства ввода и вывода, что в случае персонального компьютера подразумевает клавиатуру и монитор (мышь тоже может использоваться в консольном режиме и даже некоторым образом управляется через syscons, хотя собственно к консольному драйверу она отношения не имеет). А ведь бóльшую часть рабочего времени любой пользователь проводит, вперив глаза в экран и барабаня пальцами по клавишам. И потому естественное его желание — наилучшим образом приспособить то и другое к своим потребностям. Ну, а syscons — он способен удовлетворить эти потребности если и в неограниченном объеме, то — очень к тому близко.

Перво-наперво, в распоряжении пользователя оказывается не одна консоль, а изрядное их количество — т. н. виртуальные консоли, каждая из которых, с некоторыми оговорками, может рассматриваться как самостоятельный экран со своей клавиатурой. На каждой из виртуальных консолей может быть запущена своя задача, между ними возможно не только практически мгновенное переключение, но и обмен данными. Именно это — один из наибо-

лее эффективных способов использования пресловутой многозадачности FreeBSD, ничуть не уступающий в ряде случаев тем, что реализуются через оконные графически интерфейсы.

Впрочем, институт (если так можно выразиться) виртуальных консолей — отнюдь не привилегия именно FreeBSD: аналогичный механизм предлагает любая уважающая себя UNIX-система. Однако реализация этого института в каждой из них имеет свою специфику.

В FreeBSD максимально возможное число виртуальных консолей определяется при конфигурировании ядра и по умолчанию составляет 16. Что, конечно, уступает теоретически доступным консолям в Linux (где их может быть 63), но мне представляется вполне достаточным.

В отличие, опять же, от Linux, где новые консоли создаются по мере потребности, в FreeBSD для этого требуются некоторые действия. Ограничение на максимально возможное их число закладывается в файле конфигурации ядра строкой

```
Options     MAXCONS=##
```

в секции Hardware device Configuration. Я не нашел указаний, можно ли увеличить их число свыше 16 (как говорилось выше, мне и так хватает), но вот уменьшить его ради экономии ресурсов — труда не составит (впрочем, стоит ли овчинка выделки?).

Тем паче, что сразу по установке системы использовать даже 16 консолей не удастся. И по двум причинам. Первая из них в том, что каждой консоли, как и любому другому устройству, реальному или виртуальному, в FreeBSD (и в UNIX вообще) должен соответствовать файл устройства в каталоге /dev. Для консоли таковыми являются файлы вида /dev/ttyv#, где # — номер консоли, как почти всегда в UNIX — начиная с нуля; консоли с номером выше 9 получают значения латинских букв в алфавитном порядке.

Так вот, таких устройств поначалу обнаруживается (всего-то навсего!) двенадцать — от /dev/ttyv0 до /dev/ttyvb. Это — в версиях 4.x, в прежних версиях их было еще меньше. И если этого оказывается недостаточно, дополнительные консоли следует создать командой `MAKEDEV`.

Не исключено при этом, что этой командой устройство создаваться откажется, сославшись на ошибку в определении номера устройства. Ничего страшного в этом нет — следует только обратиться непосредственно к команде создания устройства — `mknod` (`MAKEDEV` — это всего лишь сценарий, несколько автоматизирующий этот процесс).

Для использования `mknod` следует знать такие параметры, как тип устройства — символьный (в нашем случае — именно он) или блочный, а также старший (major) и младший (minor) номера создаваемого устройства. Первый из них — это идентификатор семейства устройств (в данном случае —

виртуальных консолей), младший — идентификатор конкретного экземпляра. Определить их достаточно легко командой вида

```
$ls -l /dev/ttyv*
```

ответом на что будет сообщение вроде

```
crw-------  1 alv   tty     12,   0 23 янв 11:15 /dev/ttyv0
crw-------  1 root  wheel   12,   1 23 янв 07:54 /dev/ttyv1
crw-------  1 root  wheel   12,   2 23 янв 07:54 /dev/ttyv2
...
crw-------  1 alv   tty     12,  11 23 янв 09:40 /dev/ttyvb
```

где 12 в пятой колонке — и есть старший номер устройства, характеризующий виртуальные консоли вообще, а цифры шестой колонки — номера младшие, суть просто порядковые их номера. Соответственно, для создания файлов устройств консолей с тринадцатой по шестнадцатую дается команда

```
$mknod /dev/ttyvc c 12 12;...; mknod /dev/ttyvf c 12 15
```

где c — указание на символьную природу устройства, первое число, 12 — старший номер устройства, а второе — младший номер, просто следующий номер по порядку.

Кроме того, консоль доступна пользователю только в том случае, если она описана в соответствующем конфигурационном файле (/etc/ttys), во-первых, на ней запущен какой-либо процесс — во-вторых, и она активизирована — в третьих. Описание консоли — это всего-навсего строка вида

```
ttyv#  "/usr/libexec/getty Pc"  cons25r    on  secure
```

где первое поле — имя файла соответствующего устройства, второе — тот самый запущенный на ней процесс (а именно — процесс getty, вызывающий приглашение пользователю авторизоваться на консоли), а четвертое — ее активизация (о значении полей третьего и пятого — чуть ниже). Так вот, по умолчанию в файле /etc/ttys строк такого вида всего восемь, так что при потребности в большем количестве консолей их следует добавить вручную.

Нужно помнить только, что по умолчанию девятая (т. е. tty8) консоль зарезервирована для запуска системы X Window, почему она и неактивизирована (значение четвертого поля — off). Кроме того, по умолчанию на ней запущен процесс xdm (авторизация в графическом режиме), хотя без этого можно и обойтись:

```
ttyve    "/usr/X11R6/bin/xdm -nodaemon"      xterm   off insecure
```

Так вот, для запуска системы X Window следует иметь минимум одну такую неактивизированную консоль. Если же есть желание запускать несколько сеансов системы X Window (теоретически это несложно) или использовать графические программы, основанные на библиотеке SVGAlib, следует озаботиться соответствующим количеством неактивизированных консолей — т. е.

должным количеством записей в файле /etc/ttys, наличием соответствующих устройств /dev/ttyv# и, наконец, чтобы это число не превышало максимально разрешенное при конфигурировании ядра. И еще — консоль, с которой запущена оконная система X, становится недоступной в текстовом режиме.

Однако я отвлекся — полагаю, с созданием достаточного количества консолей мы справились. Что это дает в реальной работе? Ну, во-первых, возможность запуска многочисленных приложений. Конечно, этого можно добиться и иными средствами — обращением к фоновому режиму или приостановке задачи (это — уже компетенция командной оболочки, о чем будет разговор в соответствующей главе). Однако виртуальные консоли дают возможности запускать приложения от лица различных пользователей, включая и root-оператора. Кроме того, на мой взгляд, навигация по консолям более удобна, более прозрачна для пользователя и, главное, более универсальна, чем любые иные способы реализации многозадачности в текстовом режиме.

Немного о навигации. Переключение на другую виртуальную консоль осуществляется комбинацией клавиш <Alt>+<F#>, где <F#> — функциональная клавиша, соответствующая номеру консоли. Кроме того, на консоль, следующую по порядку за текущей, можно переключиться клавишей <PrintScreen>. Переключение осуществляется циклически, т. е. после последней доступной — снова на первую. Если, конечно, не запущена система X Window — в ней комбинация <Alt>+<F#> зарезервирована для собственных целей (и используется во многих оконных менеджерах). Поэтому для выхода из графической консоли используется комбинация <Alt>+<Ctrl>+<F#>. И вообще, следует четко уяснить (и всегда помнить), что любые консольные настройки (скажем, переопределение клавиш, о котором будет говориться ниже), никакого влияния на систему X Window не окажут — там все будет по-своему.

Все сказанное понятно для консолей с первой по двенадцатую. А как быть, если консолей больше, чем функциональных клавиш? Ведь перебирать их последовательно клавишей <PrintScrn> не всегда удобно. На сей предмет существует комбинация <Alt>+<Shift>+<F#>, позволяющая перемещаться по консолям 13—16.

Кроме того, все это касается назначения клавиш по умолчанию, в разделе о клавиатурной части консоли будет показано, что оно может быть переопределено. В частности, для переключения на старшие консоли можно задействовать т. н. Windows-клавиши (не зря же они занимают место на клавиатуре?).

Следующее замечательное свойство виртуальных консолей — возможность обмена данными между запущенными в них приложениями. Как известно, такая возможность (через буфер обмена, скажем) считается одним из величайших достижений режима графического (например, в Windows). Однако и

в текстовой консоли это — элементарная процедура. Выполняется она мышью — выделенный обычным образом, при нажатой левой кнопке (или последовательными щелчками левой и правой кнопками — в начале и конце области выделения) в одной из консолей экранный блок, вставляется в любую другую (в том числе и ту же самую) консоль щелчком средней кнопки.

Следует только уяснить, что выделение/вставка мышью в консоли FreeBSD осуществляется принципиально иначе, чем в Windows или в большинстве приложений X Window (отступившихся от первозданного назначения UNIX-мыши). Выделяется с помещением в буфер памяти (и, соответственно, потом вставляется) не объект типа текстового фрагмента или чего другого, а именно блок экрана. То есть если строка уходит за его границу — при выделении мышью она как бы и не существует. И потому вставлена не будет. Чтобы выделить/вставить фрагмент текстовой строки, лежащей за пределами экрана, последний следует пролистать в горизонтальном или вертикальном направлении.

Кроме того, мышь в консоли FreeBSD — ни в коей мере не устройство указательное и позиционирующее. То есть она не оказывает никакого влияния на перемещение текстового курсора — графический курсор консольной мыши ведет абсолютно самостоятельную жизнь, не оказывая ни малейшего воздействия и на меню программ типа Midnight Commander. В этом отличие консольной мыши FreeBSD от таковой в Linux, где посредством программы gpm мышь выполняет и указательные функции. Правда, только в некоторых программах (том же mc, к примеру), в отместку за что в них не работает функция выделения и вставки. В FreeBSD же, повторяю, эта функция в консоли абсолютно универсальна.

Впрочем, разговор о службе консольной мыши у нас еще впереди. А пока, надеюсь, мне удалось убедить читателя в том, что виртуальная консоль — штука исключительно полезная. И что чем больше консолей (в разумных, конечно, пределах) — тем лучше. Осталось только определить, как же не запутаться в этом изобилии — пользователям Windows отлично знакома ситуация, когда за деревьями (то бишь открытыми окнами) леса не видно...

Для этого резонно все доступные консоли упорядочить и систематизировать. Принцип систематизации каждый должен придумать для себя сам. Могу только в качестве примера описать, как действую я.

Обычно у меня доступно 10—15 консолей. Одна из них (последняя по счету) зарезервирована для системы X Window, которая всегда запускается с консоли, идущей непосредственно перед ней (становящейся, таким образом, недоступной в текстовом режиме).

За каждой из прочих консолей закреплен свой круг задач: мониторинг процессов посредством команды `top`, запуск mpeg-проигрывателя, просмотр документации.

Непосредственно для текущей работы отведены консоли с 1-й по 3-ю. Для переключения на них, кроме штатных комбинаций <Alt>+<F1> (<F2>, <F3>), задействованы также Windows-кнопки — левое и правое **Окно** и **Menu**. К слову сказать, клавиша <PrintScreen> служит у меня не для последовательного просмотра консолей, а для переключения в сеанс X Window.

Наконец, отдельная (и единственная) консоль выделена в распоряжение root-оператора. Причем авторизоваться с других консолей в этом качестве я не могу. Это сделано (вполне сознательно) заменой во всех (кроме одной) строках описания консолей файла /etc/ttys значения последнего, пятого, поля — с `secure` (безопасный) на `insecure`. Что указывает на небезопасность этих консолей и запрещает авторизацию на них в качестве администратора (впрочем, не мешая временно получить его права командой `su`). Единственная строка, где сохранилось исходное `secure`, соответствует той самой консоли, на которой разрешено местопребывание root-оператора.

Интересный вопрос — как зримо и весомо различать консоли разного назначения? Возможно и это — каждая из них (в определенных пределах) может быть настроена самостоятельно. О чем мы поговорим в следующих разделах.

Видеорежимы консоли

Вспомним для начала, что консоль — это (в контексте данного труда) диалектическое единство экрана монитора и клавиатуры. Каждый из этих компонентов, как легко догадаться, настраивается независимо друг от друга. Для чего в составе syscons имеются две самостоятельные программы — vidcontrol и kbdcontrol соответственно. Любой пользователь сначала смотрит на экран, а потому уж начинает молотить по клавишам (хотя, вероятно, есть и исключения — вроде чукчи-писателя, но отнюдь не читателя). Поэтому и мы начнем, благословясь, с экрана. То есть — с программы vidcontrol. И сначала — с ее возможностей по расширению объема выводимой информации.

Первая ассоциация, возникающая при слове "экран", — это его разрешение (или, как часто говорят применительно к текстовой консоли, режим отображения). Поскольку экран консоли — явление текстового режима, то и разрешение его (за одним исключением, о котором — позднее) оценивается в количестве символов по горизонтали (т. н. колонках) и по вертикали (строках). Так, стандартное разрешение консоли FreeBSD по умолчанию — 80×25 (80 колонок при 25 строках). Что, разумеется, возможно исключительно при моноширинных шрифтах, каковые только в консоли и используются (впрочем, к этому мы тоже еще вернемся).

Говоря о консольной мыши, я вскользь упоминал, что, каков бы ни был размер текстового массива по обеим координатам, в каждый момент времени можно манипулировать только с тем его блоком, который виден на эк-

ране. Поэтому возникает естественное желание уместить на нем как можно больше информации. На что подвигают пользователя и все увеличивающиеся диагонали мониторов. Ведь согласитесь, смешно видеть на крутой "семнашке", а то и "девятнашке", ровно столько же, сколько и на утильном 14-дюймовом VGA десятилетней давности. К счастью, стандартное разрешение не есть нечто непреходящее, как общечеловеческие ценности. И с помощью команды `vidcontrol` с указанием нового видеорежима в качестве аргумента легко может быть изменено.

Список доступных режимов можно посмотреть на соответствующей странице документации (`man vidcontrol`):

```
mode
Select a new video mode.  The modes currently recognized are:
80x25, 80x30, 80x43, 80x50, 80x60, 132x25, 132x30, 132x43,
132x50, 132x60, VGA_40x25, VGA_80x25, VGA_80x30, VGA_80x50,
VGA_80x60, VGA_90x25, VGA_90x30, VGA_90x43, VGA_90x50, VGA_90x60,
EGA_80x25, EGA_80x43, VESA_132x25, VESA_132x43, VESA_132x50,
VESA_132x60.  The raster text mode VESA_800x600 can also be cho-
sen.  See Video Mode Support below.
```

Из чего следует вывод, что легко изменить плотность строк — до 30, 43, 50 или 60). Для чего дается команда

```
#vidcontrol 80x30
```

где в качестве аргумента можно указать любой из видеорежимов, от 80×25 до 80×60. Правда, не на каждой видеокарте все эти режимы обязаны быть доступными. Кроме того, при очень высокой плотности строк буквы (особенно русские) могут приобрести крайне уродливый вид. Но это — уже проблема шрифтов, о которых пока не говорим.

Однако таким образом мы увеличиваем лишь количество строк, ничуть не увеличивая рабочую область по горизонтали. Тогда как manual толсто намекает нам, что есть режимы с 90 и даже 132 колонками. Не говоря уже о загадочном разрешении 800×600, кажущемся чуждым текстовому режиму. Однако немедленная попытка опробовать эту экзотику успехом не увенчается — последует сообщение об отсутствии поддержки таких режимов.

Чтобы воспользоваться предлагаемым `vidcontrol` богатством режимов, потребуется пересборка ядра. Если обратиться к общему файлу его конфигурации (/usr/src/sys/i386/conf/LINT), то в его секции Hardware device Configuration можно при должном внимании обнаружить строки

```
#options  VGA_WIDTH90
#options  VESA
```

первая из которых отвечает за 90-колоночный, а вторая — за 132-колоночный режим. Каковые и будут доступны, если перетащить их в свой

файл конфигурации ядра, снять комментарии, а само ядро — перекомпилировать. Ну и перезагрузить машину, естественно...

Для получения же режима квази-SVGA (800x600) ищем в том же LINT строку

```
#options  SC_PIXEL_MODE
```

Далее — тем же порядком. Результат, как сказал бы поручик Ржевский, — бесподобен. Правда, на моем мониторе растянуть изображение в этом режиме на весь экран так и не удалось, так что радости от него оказалось мало. Но теоретически — интересно.

Остается добавить, что таким образом (через `vidcontrol`) разрешение экрана устанавливается только в текущей консоли. Вернее, в консоли, на которую направлен или перенаправлен стандартный вывод команды, но обычно это одно и то же. И, разумеется, изменение видеорежима имеет силу только в текущем сеансе работы, до перезагрузки машины. Если есть желание пользоваться плодами своего труда постоянно, да еще и на всех консолях — следует подредактировать файл /etc/rc.conf, а именно — вписать в него строку, например,

```
allscreens_flags="80x30"
```

или что еще душе угодно. В принципе никто не мешает установить разные разрешения для отдельных виртуальных консолей, но это потребует несколько иной формы перенаправления вывода (и внесения соответствующих строк не в /etc/rc.conf, а, скорее, куда-нибудь вроде /etc/rc.syscons). Я в этом смысла не вижу и предоставляю изучение такой возможности заинтересованным лицам.

После смены разрешения экрана по умолчанию в консоли могут начаться мелкие (говорят, что и не очень мелкие) безобразия, исправление коих потребует еще некоторых деяний. Каких? — спросите вы меня. Попробую ответить — настало время вспомнить о файле /etc/ttys, а именно том третьем поле в описании консолей, толкующем о значении cons25r. Это — имя типа терминала, приписанного данной виртуальной консоли. Обсуждение вопроса, что такое тип терминала, далеко выходит за пределы моих задач (да, пожалуй, что и компетенции). И потому, в меру своего разумения, замечу, что слово это понимается, как минимум, в двух смыслах — как синоним (более или менее) консоли, и как программа, эмулирующая некий конкретный (т. е. "железный") терминал (что в режиме графическом, что в текстовом).

Впрочем, простой пользователь (вроде меня, скажем) понимать это и не обязан. Достаточно знать, что существует некий файл /etc/termcap, представляющий собой описание всех возможных и невозможных (относительно невозможных, вернее, вышедших из употребления, — это точно, а вот что касается возможных — тайна сия велика есть) терминалов. В том числе и тех, что не застали ни деды, ни отцы всемирной (чуть не написал — всесоюзной) информатизации. Правда, это — не сам файл, а лишь символическая

ссылка на /usr/share/misc/termcap. Однако объем последнего (а это просто перечисление в обычном текстовом формате, или, как задумчиво говорят, база данных) — впечатляет: почти 200 Кбайт.

Так вот, в нем среди всяких Sun, DEC, pcvtXXXX и протчая (правда, почти все они претендуют на воспроизведение некоего легендарного ANSI-терминала, являющегося стандартным), нашлось место и для терминала consXX, и его вариаций на тему consXXr. Легко догадаться, что цифры в названии указывают на число строк его видеорежима, а буква r — на русификацию терминала (конкретно — поддержку кодировки KOI8-R):

```
cons25w|ansiw|ansi80x25-raw:\
:al=\E[L:am:bs:NP:cd=\E[J:ce=\E[K:cl=\E[H\E[J:cm=\E[%i%d;%dH:co#80:\
:dc=\E[P:dl=\E[M:do=\E[B:bt=\E[Z:ho=\E[H:ic=\E[@:li#25:cb=\E[1K:\
:ms:nd=\E[C:pt:rs=\Ec:so=\E[7m:se=\E[27m:up=\E[A:\
:pa#64:Co#8:AF=\E[3%dm:AB=\E[4%dm:op=\E[39;49m:AX:sc=\E7:rc=\E8:\
:k1=\E[M:k2=\E[N:k3=\E[O:k4=\E[P:k5=\E[Q:k6=\E[R:k7=\E[S:k8=\E[T:\
:k9=\E[U:k;=\E[V:F1=\E[W:F2=\E[X:K2=\E[E:nw=\E[E:ec=\E[%dX:\
:kb=^H:kh=\E[H:ku=\E[A:kd=\E[B:kl=\E[D:kr=\E[C:le=^H:eo:sf=\E[S:sr=\E[T:\
:kN=\E[G:kP=\E[I:@7=\E[F:kI=\E[L:kD=\177:kB=\E[Z:\
:IC=\E[%d@:DC=\E[%dP:SF=\E[%dS:SR=\E[%dT:AL=\E[%dL:DL=\E[%dM:\
:DO=\E[%dB:LE=\E[%dD:RI=\E[%dC:UP=\E[%dA:cv=\E[%i%dd:ch=\E[%i%d:bw:\
:mb=\E[5m:md=\E[1m:mr=\E[7m:me=\E[m:bl=^G:ut:it#8:km:
cons25|ansis|ansi80x25:\
:ac=l\332m\300k\277j\331u\264t\303v\301w\302q\304x\263n\305-Da\260f\370g\
361~\371.^Y-^Xh\261i^U0\333y\363z\362:\
:tc=cons25w:...
cons60-m|ansi80x60-mono:\
:li#60:tc=cons25-m: # KOI8-R FreeBSD console with ACS support
cons25r|pc3r|ibmpc3r|cons25-koi8-r:\
:ac=q\200x\201m\204v\211j\205t\206n\212u\207l\202w\210k\203y\230z\231f\23
4~\225a\2 20h\221-D.^Y-^Xi^U0\215:\
:tc=cons25w:
```

Вообще-то в termcap описаны все свойства терминала, начиная от разрешения и кончая кодами функциональных клавиш. Если это не так — в конце описания имеется поле :tc=имя_рек, указывающее, что отсутствующие свойства берутся из описания указанного терминала. Однако на данном этапе нас интересует только два свойства — co (количество колонок) и li (количество линий). И если есть желание постоянно работать с нестандартным разрешением, нужно подобрать описание терминала с соответствующим их количеством. И во всех полях /etc/ttys заменить cons25r на, скажем, cons30r (при тридцатистрочной плотности символов по вертикали).

В отношении 80-колоночных режимов это сделать легко — таковые среди терминалов consXX представлены в полном наборе. Однако попытки найти в семействе consXX типы терминала для 90- или 132-колоночного режима будут безуспешны. Хотя такие режимы обнаруживаются в иных семействах терминалов, например, в pcvt. Однако не думаю, что использовать их — здоровая идея. А от режима 800×600 вообще нет никаких следов. Что делать?

Честно говоря, для меня это осталось не совсем ясным. Иван Паскаль (см. дополнительные источники) рекомендует создать описание собственного терминала в /etc/termcap. Которое в данном случае потребует только двух полей — нового числа колонок (co) и ссылки на исходный тип (tc=имя_рек). Не забыв ограничить их разделителем полей — двоеточием: все же это база данных, а не что иное. Однако по умолчанию файл /etc/termcap (вернее, то, на что он ссылается) закрыт для записи даже для root-оператора. Каковой, конечно, может присвоить себе такое право. Однако, видимо, разработчики полагали сие нежелательным.

В итоге, я, работая постоянно в режиме VGA_90×30, ограничиваюсь тем, что указываю в /etc/ttys свой тип терминала как cons30r — и пока ни с какими сложностями не столкнулся. Не знаю, насколько это идеологически правильно. Но ведь нам, бывшим советским людям, всякие там идеологии по... (сами знаете, по чему).

И еще одно замечание, также имеющее отношение к идеологии. Считается правильным явным образом задать значение переменной TERM в стартовых файлах своей командной оболочки, т. н. dot-файлах. В Shell-совместимых средах (например, в ~/.profile для /bin/sh или в ~/.bash_profile для bash) это будет иметь вид

```
TERM=${TERM:-cons30r}
export TERM
```

а в средах семейства C-Shell —

```
setenv    TERM    cons30r
```

Кроме того, при изменении видеорежима "на лету" (командой vidcontrol) соответствующую переменную также, возможно, потребуется переопределить вручную, командой set. Иначе некоторые программы могут вести себя неправильно.

Теперь — еще об одном способе увеличения объема выводимой на экран информации, экранном буфере. Возможность пролистывания текстовых экранов — это вторая особенность, которая не перестает восхищать меня в виртуальной консоли (после самого факта ее существования). Правда, в FreeBSD это делается клавишами <PageUp>/<PageDown> при включенном <ScrollLock>, а не комбинацией <Shift>+<PageUp>/<PageDown>, как в Linux-консоли. Что лучше — дело вкуса: мне удобнее первый метод, т. к. переключатель <ScrollLock> — фиксирующий.

Предельный размер экранного буфера (его не следует путать с длиной истории команд, которая входит в сферу контроля командной оболочки) устанавливается при сборке ядра опцией

```
Options     SC_HISTORY_SIZE=###
```

и по умолчанию равен, если мне не изменяет память, 200 строкам. Однако в этом пределе его можно варьировать как угодно командой `vidcontrol -h ###`, причем — для каждой виртуальной консоли отдельно. Правда, действие ее распространяется только на время после исполнения команды.

В прежних версиях FreeBSD размер экранного буфера определялся программой kbdcontrol, как это описано у Ивана Паскаля (и по сию пору это как реликт сохранено за ради совместимости). Однако ныне логика восторжествовала и управление экранным буфером отдано по принадлежности — команде `vidcontrol`.

Наконец, переключение между виртуальными консолями — также некоторым образом расширение объема доступной информации. И это, помимо прочих способов, можно делать командой

```
vidcontrol -s #
```

где `#` — номер требуемой консоли. Вопреки обыкновению UNIX, нумерация в данном случае осуществляется не с нуля, а с единицы.

На этом тему расширения кругозора можно считать исчерпанной. В следующем разделе речь пойдет о том, как придать количеству выводимой информации качественный (по крайней мере внешне) вид.

Украшение консоли

Я всегда не уставал декларировать, что текстовая консоль по умолчанию столь прекрасна сама по себе, что в украшениях не нуждается. Однако и здесь есть горизонты для совершенствования. И охватывают они, в первую голову, шрифты.

Действительно, шрифты, входящие в комплект FreeBSD и устанавливаемые по умолчанию в процессе инсталляции, весьма далеки от эстетического идеала (да и от элементарной эргономики), особенно в кириллической своей части. И поменять их — дело святое. Однако сначала посмотрим, какие они бывают, консольные шрифты.

Все штатные шрифты для консоли FreeBSD хранятся в каталоге /usr/share/syscons/fonts и имеют имена вида cp850-8x14.fnt, cp850-8x16.fnt, cp850-8x8.fnt. Первая часть имени (в данном случае cp850) соответствует кодировке (вернее, набору символов) шрифта и определяет его семейство — гарнитуру, если это слово здесь уместно. Вторая же часть (8×14 и т. д.) — указание на размер шрифта, его с позволения сказать, "кегль", хотя вот

здесь-то смысл явно другой. Действительно, под размером шрифта в данном случае понимается матрица из точек по горизонтали и вертикали, а не его измерение в каких-либо типографских единицах. Тем не менее, дальше я буду использовать как бы типографские термины, не забывая, правда, про кавычки.

Можно видеть, что каждая "гарнитура" экранного шрифта представлена тремя "кеглями". И в файле /etc/rc.conf одновременно загружаются все три шрифтовых файла. Зачем? Каждый из них соответствует группе экранных разрешений, вернее — плотности символов по вертикали. При видеорежимах с 25 или 30 строками используется шрифт с матрицей 8×16, с 43—60 строками — с матрицей 8×8 (матрица 8×14 предназначена для EGA-режимов и ныне, судя по всему, сохраняется просто как реликт).

Интересно (вернее, как раз *не интересно*), что плотность символов по горизонтали с "кеглем" шрифта никак не связана — и при 80-, и при 90-, и при 132-колоночных режимах горизонтальная составляющая матрицы — 8 точек. Именно в этом причина крайне уродливого вида экрана в режимах 80×50 и 80×60.

Нас, разумеется, в первую очередь интересуют шрифты с кириллической составляющей. Они представлены в комплекте для трех кодировок русского языка — DOS (семейство cp866*), Windows (cp1251*) и UNIX (koi8*); для первой и последней, кроме шрифтов русских, имеются и украинские наборы символов.

Далее, DOS- и UNIX-наборы символов имеют в старшем, предназначенным для стандартного разрешения, "кегле" подварианты, именуемые cp866-8×16, cp866b-8×16, cp866c-8×16 и koi8-r-8×16, koi8-rb-8×16, koi8-rc-8×16, соответственно. Вот они-то, собственно, и представляют собой различные гарнитуры. Шрифт cp866-8×16 — почти беззасечечный (Sans Serife), сходный со шрифтами семейства fixed, cp866b-8×16 — курьерообразный, с засечками, cp866c-8×16 — весьма причудливая смесь того и другого (аналогичный облик имеют и шрифты для koi набора символов). Кроме того, шрифт cp866c-8×16 имеет несколько другой набор символов псевдографики.

Как уже неоднократно говорилось, для вывода на экран кириллических текстов в FreeBSD традиционно используются шрифты в кодировке DOS, хотя для ввода с клавиатуры установлена кодировка KOI8. Настало время объяснить причины этого явления. Требующего, помимо всего прочего, еще и загрузки таблицы перекодировки.

Причин этих — две. Первая кроется в аппаратных особенностях видеосистемы PC, которые при использовании шрифтов в кодировке KOI8 обуславливают не очень импозантный вид программ, задействующих символы псевдографики, таких, как Midnight Commander, текстовый редактор le и ряд других.

Конечно, с пунктирным видом меню и панелей mc можно примириться. Однако есть и более веская причина: коды, используемые syscons для отрисовки курсора мыши (D0—D3), совпадают с кодами нескольких русских букв — п, я, р, с, — в кодировке KOI8 (в DOS-кодировке на этом месте расположены редко используемые символы псевдографики). В результате при активизированной консольной мыши (а без нее, как уже говорилось, работать в консоли было бы скучно) просмотр русских текстов попросту невозможен: указанные буквы пропадают с экрана при любом мышином движении (не для этого ли в vidcontrol придумали возможность отключения курсора мыши?).

В документации проекта FreeBSD для исправления ситуации с мышиным курсором предлагается пересборка ядра с опцией

```
Options     SC_MOUSE_CHAR=0x3
```

Это помогает, хотя, как говорят, тоже может вызывать отрицательные побочные эффекты.

Так что, вероятно, использование DOS-кодировки экранных шрифтов в FreeBSD (вернее, собственно в syscons, для pcvt проблемы с мышиным курсором нет) — принудительная сила реальности. И соответствующая процедура исполняется на стадии установки (или постинсталляционного конфигурирования) с помощью программы sysinstall. Но установка при этом по умолчанию (и — безальтрнативно) экранного шрифта cp866b-8×16 — на мой взгляд, не оправдана ничем.

Мало того, что шрифты с засечками — на мой близоруко-астигматичный взгляд, вообще не лучший выбор для экрана, для растровых шрифтов эти гарнитуры просто медицински противопоказаны. Так шрифт cp866b-8×16 отличается и еще одной особенностью — отчетливым различием размера символов латинской и кириллической составляющей. Что, конечно, полезно для тех, кто не вполне уверенно отличает русские буквы от английских, но зрительно производит странное впечатление.

Все мои столь пространные рассуждения призваны были обосновать нехитрый тезис — с экранными шрифтами можно и нужно экспериментировать. Возможности для чего предоставляет все та же программа vidcontrol. Для загрузки некоего интересующего нас шрифта ее следует вызвать в форме

```
$vidcontrol -f типоразмер_шрифта имя_шрифтового_файла
```

Например, командой

```
$vidcontrol -f 8x16 /usr/share/syscons/fonts/cp866-8x16.fnt
```

будет загружен шрифт cp866-8×16 с соответствующей матрицей — на мой взгляд, оптимальный выбор в рамках штатного комплекта, как с точки зрения эстетики, так и эргономики. При этом необходимости в полном пути к

шрифтовому файлу нет, если он хранится в стандартном каталоге (именно том, который указан в примере).

Если результат экспериментов устроил — его можно увековечить для постоянного использования редактированием файла /etc/rc.conf, заменив строку

```
font8x16="cp866b-8x16"
```

на

```
font8x16="cp866-8x16"
```

Строки для остальных двух шрифтов, разумеется, трогать ни к чему — их можно просто закрыть комментариями, если не предполагается использование разрешений выше 80×30.

К сожалению, выбор консольных шрифтов в FreeBSD небогат. Так, он не идет ни в какое сравнение с наборами кириллических консольных шрифтов в современных отечественных дистрибутивах Linux (таких, как Altlinux или ASPLinux). Не удалось мне отыскать дополнительных шрифтов и в Сети. Однако и тут можно найти выход — конвертировать консольный шрифт для Linux в формат FreeBSD. Для этого первый следует сначала разархивировать (в Linux шрифты хранятся в виде gzip-архивов)

```
$gzip font.psf.gz
```

а затем преобразовать следующей командой:

```
$dd if=linux_font.psf of=bsd_font.fnt bs=1 skip=4 count=4096
```

где значение `count` приведено для шрифта стандартного (80×25) разрешения с матрицей 8×16. Однако теоретически можно сделать и шрифт для высокого разрешения — 8×8. А если исходный шрифт — Unicode (а в дистрибутивах Linux есть и такие), то, вычислив смещение, можно изготовить шрифты в любой кириллической кодировке.

Ведь в консоли FreeBSD при желании и необходимости есть возможность подгружать и шрифты в Windows-кодировке (благо, таковые в комплекте поставки имеются). Однако тут следует иметь в виду два обстоятельства. Во-первых, внешний вид программ, использующих псевдографику, будет просто никакой. А во-вторых (и главных), смена экранного шрифта средствами vidcontrol действует одновременно (и имеет обратную силу) для всех виртуальных консолей. И если на них ранее были выведены какие-либо кириллические тексты, они тут же утратят читабельность. Которая, впрочем, восстановится после повторной загрузки шрифта в DOS-кодировке, и потому в качестве временной меры (необходимости срочно прочитать письмо в cp1251, например) такое действие приемлемо. Если же необходимость работать с документами с cp1251 возникает постоянно, потребуется также соответствующая клавиатурная раскладка (о чем — ниже). и, вероятно, таблица перекодировки — т. н. screenmap, — для трансляции Windows-кодов, вводи-

мых с клавиатуры, в DOS-коды экранных шрифтов. Последнее все же более простое решение, чем тотальная замена шрифтов.

С слову сказать, загрузка таблицы перекодировки — также прерогатива программы vidcontrol. Делается это таким образом:

```
$vidcontrol -l имя_файла_screenmap
```

Файлы таблиц перекодировок можно найти в каталоге /usr/share/syscons/scrnmaps/. Из интересующих нас там имеется только koi8-r2cp866.scm (и ее украинский вариант — koi8-u2cp866u.scm) для трансляции клавиатурной раскладки KOI8 в шрифты DOS-кодировки. Так что при необходимости ввода Windows-кодов такую таблицу придется изготовить самостоятельно. Ну, а чтобы отказаться от какой-либо перекодировки, в том числе и подгруженной через /etc/rc.conf, потребуется команда

```
$vidcontrol -L
```

Впрочем, существует и другое решение проблемы одновременной работы с разными кодировками, о котором я расскажу в свое время. А пока — еще об одном резерве для украшения консоли — ее расцветке. Изменение которой тоже осуществляется программой vidcontrol в форме

```
$vidcontrol цвет_текста цвет_фона
```

Ранее по умолчанию всегда устанавливался темный фон при светлом тексте, для получения же обратной картины требовалась опция `-r` (reverce). Ныне в этом нет необходимости (хотя использовать эту опцию никто не запрещает).

Список доступных цветов определяется командой

```
$vidcontrol show
```

и включает пары: `black—grey`, `blue—lightblue`, `green—lightgreen` и т. д.

В отличие от загрузки шрифтов, изменение расцветки текста и фона не распространяется на все виртуальные консоли, подобно изменению видеорежима. Однако, отличие от последнего, смена расцветки, во-первых, не имеет обратной силы (т. е. для ранее введенного текста она сохраняется той же, что и была). А во-вторых, вывод команды смены расцветки направляется на стандартный вывод сообщения об ошибках. По умолчанию таковым является текущая консоль, но вывод этот может быть перенаправлен определенным образом, если есть желание закрепить определенные сочетания цветов за тем или иным терминалом. Ведь это — один из тех самых способов зримого различения консолей, о которых я упоминал ранее.

Я не нашел в документации указаний, в какой из стартовых файлов следует вносить изменения для смены расцветки. Однако методом ползучего эмпиризма пришел к выводу, что самое подходящее для этого место — файл /etc/rc.syscons. Куда можно вписать строки вида, например:

```
$vidcontrol white blue > /dev/ttyv# 2>&1
```

что придаст консоли вид, привычный многим по Norton Commander с его Editor (и по их многочисленным клонам). Последний оператор (`2>&1`) как раз и означает перенаправление вывода сообщения об ошибках (цифра 2) на стандартный вывод терминала /dev/ttyv#, где # — его номер, а 1 — цифра, зарезервированная за стандартным его выводом (напомню для справки, что число 0 соответствует стандартному вводу).

При переопределении расцветки консоли следует иметь в виду, что это, возможно, потребует изменения настроек некоторых программ. Одной из них, почти наверняка, окажется команда `ls`, которая, будучи запущена с определенными опциями, выводит информацию о файлах и каталогах в цветах (об этом будет говориться в главе о командных оболочках). И вполне возможно, что эти цвета потеряются на новом фоне — именно так и будет, если использовать пример с Norton-подобной расцветкой консоли.

Это — одна из причин, почему изменение цвета фона и текста — не самое удобное средство для различения консолей (вторая причина — мне просто не нравятся любые их цвета, кроме "умолчальных"). Но с этой целью можно использовать еще одну особенность syscons — возможность изменения цвета рамки монитора — т. н. бордюра (border). Пользователи Windows на современных мониторах, возможно, даже не знают, что это такое. Но заставшие старые поганые четырнадцатидюймовики помнят, что рамка эта подчас отхватывала по пол-пальца с каждой стороны экрана. Ныне, конечно, соответствующими настройками ее можно свести к нулю. А можно — просто уменьшить и заставить играть полезную роль. Что делается командой

```
$vidcontrol -b цвет
```

где опция `-b`, как легко догадаться, происходит от слова border, а цвет берется из того же `vidcontrol show`. Ну а увековечивание изменений — опять же через редактирование /etc/rc.syscons, куда можно вписать что-либо вроде

```
vidcontrol -b red > /dev/ttyv# 2>&1
```

как и для изменения расцветки консоли. Я, например, сделал красный бордюр (на моем мониторе он имеет устрашающе-багряный оттенок) для той единственной консоли, с которой позволяю себе входить в качестве root-оператора, консолям для мониторинга и запуска mpeg-файлов придал, напротив, рамки жизнерадостных цветов, а основным рабочим — спокойно-нейтральные. Получается очень наглядно.

Теоретически рассуждая, изменению цвета рамки можно придать и иные функции, например, индицирования раскладки клавиатуры (латинской/русской), как это делалось в старых DOS-русификаторах. Хотя включение индикатора CapsLock по умолчанию в FreeBSD делает это не очень актуальным.

Далее, к понятию украшения можно отнести и изменение вида текстового курсора, который не следует путать с графическим курсором мыши (о нем речь пойдет под занавес главы).

Делается это командой

```
$vidcontrol -c тип_курсора
```

Опция `-c` может принимать значения `normal` (это вид курсора по умолчанию, в виде прямоугольника) или `blink` (тот же прямоугольник, но мерцающий); есть и вариант, почему-то именуемый `destructive` — в виде помигивающей черточки. Но на самом деле именно он кажется самым удобным — позиция курсора хорошо видна на фоне выделения мышью, и он не столь раздражает, как `blink`.

И в заключение — еще об одной замечательной функции `vidcontrol` — возможности создания экранных дампов, т. е. копий буфера экрана посредством

```
$vidcontrol -p
```

в бинарном формате, или

```
$vidcontrol -P
```

в текстовом, с игнорированием непечатных символов и текстовых атрибутов. В обоих случаях вывод команды может быть перенаправлен в файл, однако само по себе радости это доставит мало — нужно обладать изрядной фантазией, чтобы принять такой dump за изображение консоли. Однако в главе о консольной мультимедиа я расскажу, как экранные дампы воплотить в истинно графические иллюстрации.

Настройка клавиатуры

Клавиатура — второй компонент системной консоли. Причем, строго говоря, более важный, чем экран: ведь чтобы увидеть на последнем хоть что-то, отличное от картины начальной загрузки, нужно дать какую-никакую клавиатурную команду (хотя бы авторизоваться в системе). И теоретически без экрана работать можно — получается крайний вариант чукчи-писателя, который не читает даже написанное собой, — а вот без клавиатуры...

Примечание

Отступление из мемуаров эникейщика. Помнится, когда в конце 80-х на моей тогдашней службе появилась первая персоналка (86-й Amstrad), она почиталась столь великой ценностью, что по приказу руководства на ночь клавиатуру запирали в опечатанный сейф, наравне с грифованными материалами, — для сохранности. И когда утром машину включали (о клавиатуре, разумеется, забыв), она на хорошем немецком языке ехидно спрашивала: "Интересно, ребята, *а как вы собираетесь работать без клавиатуры?*"

Так что одна из задач драйвера syscons — обеспечить ввод символов, из которых слагаются команды и тексты. Для этого он преобразует коды нажатия

клавиш (т. н. скан-коды), генерируемые контроллером клавиатуры (т. е. "железной составляющей"), в коды символов таблицы ASCII. Преобразование это происходит в два этапа — сначала "железный" скан-код нажатия клавиши, именуемый также "сырым" (вообще говоря, отпускание клавиши тоже генерирует "сырой" скан-код, но для нас это почти не существенно), превращается во внутренний код syscons. Как, зачем, почему — опять же неважно, эта процедура запрятана в недра драйвера и влиять на нее мы не можем. Далее же внутренний код (также называемый скан-кодом) преобразуется в ASCII-код по специальной таблице, именуемой таблицей раскладки клавиатуры (keymap).

Доступные для syscons раскладки клавиатуры собраны в каталоге /usr/share/syscons/keymaps. Это — простые текстовые файлы вида "страна.тип.kbd" (строго говоря, не совсем страна, но об этом позже), которые можно просмотреть, например, командой `less имя_файла`. Что мы и проделаем на примере стандартной американской клавиатуры us.iso.kbd — насколько я понимаю, именно она используется по умолчанию, если не загружена тем или иным способом какая-либо другая раскладка. К чести создателей FreeBSD нужно заметить, что это одна из немногих американских по происхождению систем, которая догадывается о существовании разных языков и алфавитов, кроме американского.

Итак, в ответ на

```
$less /usr/share/syscons/keymaps/us.iso.kbd
```

на экран выводится примерно следующая картина:

```
 # $FreeBSD: src/share/syscons/keymaps/us.iso.kbd,v 1.14.2.1 2000/07/18
00:47:53 ache Exp $
#                alt
#scan        cntrl    alt  alt  cntrl  lock
#code  base  shift  cntrl  shift  alt  shift  cntrl  shift  state
#---------------------------------------------------------------------
000  nop   nop   nop   nop   nop   nop   nop   nop   0
001  esc   esc   esc   esc   esc   esc   debug esc   0
002  '1'   '!'   nop   nop   '1'   '!'   nop   nop   0
003  '2'   '@'   nul   nul   '2'   '@'   nul   nul   0
004  '3'   '#'   nop   nop   '3'   '#'   nop   nop   0
...
020  't'   'T'   dc4   dc4   't'   'T'   dc4   dc4   C
021  'y'   'Y'   em    em    'y'   'Y'   em    em    C
022  'u'   'U'   nak   nak   'u'   'U'   nak   nak   C
023  'i'   'I'   ht    ht    'i'   'I'   ht    ht    C
```

```
024  'o'  'O'  si  si  'o'  'O'  si  si  C
025  'p'  'P'  dle dle 'p'  'P'  dle dle C
...
104  slock  saver  slock  saver  susp  nop   susp  nop  O
105  fkey62 fkey62 fkey62 fkey62 fkey62 fkey62 fkey62 fkey62 O
106  fkey63 fkey63 fkey63 fkey63 fkey63 fkey63 fkey63 fkey63 O
107  fkey64 fkey64 fkey64 fkey64 fkey64 fkey64 fkey64 fkey64 O
108  nop  nop  nop  nop  nop  nop  nop  nop  O
```

Первая колонка этой таблицы — номера скан-кодов (разумеется, внутренних, а не "железных"), вторая — соответствующее этому коду базовое значение. Таковым может быть:

- символ (цифра, буква);
- функциональная клавиша (<F1>—<F12>);
- клавиша специального назначения (переключения виртуальных консолей, включения отладочного режима, перезагрузки машины);
- модификатор, фиксируемый (<Caps Lock> и <NumLock>) или нефиксируемый (<Shift>, <Ctrl>, <Alt>).

Номер скан-кода жестко привязан к своей физической клавише (например, скан-код 001 — это левая верхняя клавиша стандартной современной клавиатуры), а его символьное значение по умолчанию совпадает, как правило, с маркировкой этой клавиши (в данном примере — <Esc>). Подчеркну, что здесь и далее (если явно не оговорено иное) речь обычно будет идти именно о символьных значениях скан-кодов по умолчанию, поскольку все они могут быть переопределены (почти) произвольным образом. Правда, сомнительно, что кому-нибудь доставит удовольствие вспоминать, к какому скан-коду он привязал цифру "1" и как маркирована соответствующая клавиша. Однако в ряде специальных случаев к такому переопределению прибегать целесообразно.

Следующие семь колонок — значения тех же скан-кодов при нажатых нефиксируемых клавишах-модификаторах — <Shift>, <Ctrl>, <Alt> и их сочетаниях. Наконец, последняя колонка (lock state) отмечает, подвержено ли значение данного скан-кода действию фиксируемого модификатора (CapsLock).

Роль перечисленных групп клавиш в общем понятна. Символьные клавиши обеспечивают (вот странно-то!) ввод символов — алфавитно-цифровых, знаков препинания или специальных (например, всеми любимого символа $).

Функциональные клавиши обладают способностью генерировать не один код, как символьные, а их последовательность. Они обычно отдаются на откуп создателям программного обеспечения, в котором они выполняют, опять-таки, различные функции (подобно клавише <F1>, за которой DOS-

Я не знаю, можно ли переопределить положение `slock` — пролистывателя экранного буфера (исходя из общих соображений — почему бы и нет?). Да

традиция закрепила псевдоним "Help"). Это верно и для клавиш управления

только видится это ненужным — осуществление данной процедуры при включенном <ScrollLock> кажется мне более удобным, чем принятый в Linux способ через <Shift>+<PageUp>/<PageDown>.

Возвращаясь к таблице раскладки американской клавиатуры, мы видим в ней 107 задействованных строк (скан-коды с номерами от 001 до 108), чего с лихвой хватает для почти любой существующей стандартной клавиатуры (хотя, помнится, Gateway некогда комплектовал свои машины "досками" клавиш на 120). Но только — для ввода букв латинского (именно — чисто латинского) алфавита плюс минимум специальных символов (скажем, символов фунта и марки, не менее любимых народом, чем $, мы в ней не найдем).

Решение этой проблемы, встающее уже перед жителем любой западноевропейской страны (не говоря уж о нашей отчизне и братских незалежных государствах) — во введении т. н. альтернативной раскладки. В ней требуемые алфавитные символы генерируются нажатиями тех же физических клавиш, но с привязанными к ним скан-кодами, полученными из основных добавлением постоянного смещения на число 128. Что требует, естественно, дополнительного модификатора — **AltGroup** (`alock`, т. е переключателя альтернативных групп, не обязательно имеющего отношение к модификатору <Alt> или одноименной клавише). Это — фиксирующий модификатор, имеющий, теоретически, и нефиксирующий аналог — **ShiftGroup** (`ashift`), хотя практически мне трудно представить ситуацию, в которой он необходим.

В комплект FreeBSD входит большое количество раскладок с альтернативными группами — для большинства европейских стран, Бразилии, испаноязычной Америки, Японии, Украины и даже России. Собственно, как я уже обмолвился, номенклатура раскладок соответствует не странам как таковым, а локалям, но последние будут предметом отдельного разговора. Нынче же мы рассмотрим строение альтернативной группы на примере наиболее интересных раскладок — русских.

Пригодных для Святой Руси раскладок в комплекте обнаруживается аж три: ru.cp866.kbd, ru.koi8-r.kbd и ru.koi8-r.shift.kbd. Первая соответствует кодировке DOS, две другие — традиционной кодировке KOI8, различаясь переключателем латиницы/кириллицы. Из них по умолчанию, через sysinstall, устанавливается ru.koi8-r.kbd, которая и послужит нам основой для дальнейших упражнений. Просмотр ее через

```
$less /usr/share/syscons/keymaps/ru.koi8-r.kbd
```

показывает, что первая ее половина (скан-коды от 000 до 107) идентична американской раскладке. Скан-коды с номерами от 108 до 128 не задействованы, а, начиная с номера 129 (001+128), собственно и начинается альтернативная группа:

```
128  nop  nop  nop  nop  nop  nop  nop  nop  0
129  esc  esc  nop  nop  155  155  debugnop  0
```

```
130  '!'  '1'  nop  nop  177  161  nop  nop  O
131  '"'  '2'  nul  nul  178  192  128  128  O
132  '#'  '3'  nop  nop  179  163  nop  nop  O
133  '*'  '4'  nop  nop  180  164  nop  nop  O
134  ':'  '5'  nop  nop  181  165  nop  nop  O
135  ','  '6'  rs   rs   182  222  158  158  O
136  '.'  '7'  nop  nop  183  166  nop  nop  O
137  ';'  '8'  nop  nop  184  170  nop  nop  O
138  '('  '9'  nop  nop  185  168  nop  nop  O
139  ')'  '0'  nop  nop  176  169  nop  nop  O
144  202  234  dc1  dc1  241  209  145  145  C
145  195  227  etb  etb  247  215  151  151  C
146  213  245  enq  enq  229  197  133  133  C
147  203  235  dc2  dc2  242  210  146  146  C
148  197  229  dc4  dc4  244  212  148  148  C
149  206  238  em   em   249  217  153  153  C
...
```

Как и положено, она позволяет вводить русские буквы (их коды для KOI8 можно видеть во второй и третьей колонках) в обоих регистрах, знаки препинания и специальные символы. В качестве переключателя групп (`alock`) используется клавиша <Caps Lock> (скан-код 058 в основной части раскладки и 186 — в альтернативной). Собственно же функция модификатора `clock` закреплена за комбинацией <Caps Lock>+<Shift>.

Мне такое положение представляется оптимальным — по прямому назначению клавиша <Caps Lock> используется редко, с одной стороны — легкодоступна, с другой — вероятность ее случайного нажатия мала. И расположена слева — а правая рука ведь обычно лежит на мыши. Ну и переключение по одной клавише — всегда лучше, чем по любой комбинации. Однако при наличии привычки к каким-либо Windows-переключателям (типа <Alt>+<Shift> или <Ctrl>+<Shift>) закрепить за ними переключатель групп труда не составит. Можно задействовать под это и любую из Windows-клавиш. Как — оставляю на рассмотрение заинтересованным.

Более существенным мне представляется нестандартное расположение цифр и знаков препинания в верхнем ряду алфавитно-цифровой части клавиатуры. Мало того, что оно не соответствует маркировке ни старых DOS-клавиатур, ни новых, Windows, так еще и инвертировано: цифры находятся на верхнем регистре (т. е. вводятся при включенном модификаторе <Shift>), а знаки препинания — на нижнем. Само по себе это удобно — ведь точки и запятые приходится вставлять чаще, чем числа. Однако привычки к такому,

думаю, нет ни у кого. И потому я первым делом изменил бы эту часть альтернативной группы. Приведя заодно расположение знаков препинания в соответствие с маркировкой Windows-клавиатур.

О консольной мыши

В заключение рассказа о текстовой консоли — о службе консольной мыши. За работу мыши в консольном режиме отвечает программа-демон /usr/sbin/moused. Обычно поддержка мыши включается в стартовом файле /etc/rc.conf, как это было описано в *главе 4*. Однако это можно (а иногда и нужно) сделать и вручную, например, после реинициализации системы переходом в однопользовательский режим. Ведь, как уже говорилось, по возврате в режим многопользовательский не все стартовые сервисы активизируются автоматически, и демон консольной мыши — в их числе.

Для "реанимации" мыши требуется дать в командной строке команду `moused` с указанием опций — ее типа (`-t`) и порта, к которому мышь подключена (`-p`). Например, для USB-мыши эта команда примет форму

```
$moused -t auto -p /dev/ums0
```

В качестве значения опции `-t` могут выступать различные имена сериальных мышей (`microsoft`, `intellimouse`, `mousesystems` и т. д.), `busmouse` (для шинных моделей) или `ps/2` (для одноименного протокола). Для USB-мышей приемлемое значение — исключительно `auto`.

Тип мыши можно определить той же командой `moused` с опцией `-i` (от info):

```
$moused -i all -p /dev/ums0
```

в ответ на что последует сообщение вида:

```
/dev/ums0 usb sysmouse generic
```

Впрочем, от явного указания значения опции `-p` это не избавляет. Кроме того, команда эта сработает только при отключенной поддержке мыши.

Как уже говорилось, сама по себе поддержка консольной мыши к драйверу системной консоли отношения не имеет. Однако последний обеспечивает воспроизведение графического курсора мыши, которое может быть включено (или выключено) командой `vidcontrol`:

```
vidcontrol -m on
```

и

```
vidcontrol -m off]
```

соответственно. Правда, не очень понимаю, зачем так издеваться над несчастным грызуном? Разве что при использовании по умолчанию кириллических шрифтов KOI8 в момент вывода на экран русского текста?

Следует подчеркнуть, что речь идет именно о включении или отключении только воспроизведения курсора мыши — на работе демона мыши как такового это никак не сказывается.

Дополнительные источники

- Описанию текстовой консоли UNIX-систем уделяется обычно до обидного мало внимания. Пожалуй, единственный дополнительный (зато — прекрасный) источник информации на эту тему — работа Ивана Паскаля "Драйвер системной консоли — syscons" (**http://www.tsu.ru/~pascal/unix/syscons/syscons.html**).
- Кое-какие сведения о консольных шрифтах FreeBSD можно почерпнуть из заметки Дейва Боднестаба "Шрифты и FreeBSD" (**http://www.FreeBSD.org.ua/articles/fonts/**).
- Ну и для сравнения — интересные наблюдения о Linux-консоли можно обнаружить на сайте **http://linux-way.org/**.

ЧАСТЬ II

Использование

Глава 13

Принципы командного интерфейса

Начиная разговор об использовании FreeBSD, в первую очередь следует обратиться к знаменитому интерфейсу командной строки UNIX-систем (CLI — Command Line Interface). Несмотря на широкое распространение графических оконных менеджеров и интегрированных сред, именно при его посредстве решается огромный круг задач, от элементарного управления файлами до весьма сложных операций с текстами, цифровыми последовательностями и даже изображениями. Интерфейс командной строки лежит в основе подавляющего большинства утилит и приложений FreeBSD. Более того, множество интерактивных или визуальных программ этой системы (и любой другой UNIX-подобной) — не более чем интерфейсные надстройки над прямыми командными директивами. И, наконец, именно приглашение командной строки — это первое, что видит пользователь сразу же после запуска системы и авторизации.

Интерфейс командной строки в век графических режимов и интегрированных объектных сред может показаться неким атавизмом. Почему же в последнее время тема эта снова и снова привлекает к себе внимание? Причина — в явлении, которое войдет в историю ушедшего столетия под названием бума открытых исходников. Результатом его был беспрецедентный рост числа пользователей открытых UNIX-подобных ОС, в первую — Linux. Конечно, рост этот — относительный, и в абсолютном исчислении не столь уж велик. Однако ныне большинство пользователей UNIX-подобных систем — это недавние мигранты с платформы Windows. А ведь за время ее засилья выросло не одно поколение, не заставшее уже даже командной строки DOS.

Естественное стремление Windows-мигранта — и в UNIX-подобных системах действовать в привычной среде и привычными методами. Благо, стремление это удовлетворяется интегрированными объектными средами графического режима, такими, как KDE и GNOME. Да и современные т. н. user-ориентированные дистрибутивы Linux очень к тому располагают.

Однако новообращенный пользователь любой UNIX-подобной системы довольно быстро выясняет, что, как сказал бы Страшила Мудрый, — "UNIX это не Windows, а Windows — это не UNIX". И что приемы работы, заимствованные из Windows, в UNIX оказываются менее эффективными, чем традиционные инструменты последней. А в их основе, как уже было сказано, лежит CLI.

Предполагается, что со многими понятиями командного интерфейса читатель уже знаком на эмпирическом уровне — в объеме конспекта *главы 2*. Однако в настоящей главе они будут рассмотрены более последовательно и подробно.

С точки зрения пользователя, командный интерфейс складывается из четырех основных компонентов:

- командных директив;
- регулярных выражений;
- командных конструкций;
- командных сценариев.

Рассмотрим их последовательно, но пока в несколько обобщенном виде. Последнее связано с тем, что детали использования и командных директив, и конструкций, и, особенно, сценариев несколько различаются в зависимости от используемой командной оболочки. Однако все, что будет говориться ниже, применимо ко всем оболочкам (за исключением некоторых специально оговоренных случаев).

Команды, опции, аргументы

Основой командного интерфейса, как нетрудно догадаться из названия, является команда. Это — директива, посредством которой пользователь взаимодействует с командной оболочкой (она же — командный интерпретатор, о чем пойдет речь в *главе 15*). Она образуется по определенным правилам, именуемым синтаксисом. Синтаксис команд определяется, в первую очередь, языком, принятым в данной командной оболочке. Кроме того, некоторые команды (не очень многочисленные, но весьма употребимые) имеют собственный, нестандартный синтаксис.

Однако в целом базовые правила построения команд имеют много общего. И именно эти базовые правила будут предметом данного раздела. Синтаксические особенности отдельных команд будут оговариваться по ходу изложения. Различия же в синтаксисе команд различных оболочек будут рассмотрены в следующей главе.

Итак, командная директива образуется:

- именем команды, однозначно определяющим ее функции;
- опциями, определяющими условия ее выполнения;
- аргументами — обычно объектами, над которыми осуществляются действия.

Очевидно, что имя команды является обязательным компонентом, тогда как опции и аргументы могут и отсутствовать (или подразумеваться в неявном виде по умолчанию).

В подавляющем большинстве случаев опции (или их последовательности) следуют непосредственно за именем команды, а аргумент (или группа аргументов) команду завершает, хотя это правило имеет исключения. Вне зависимости от порядка опций и аргументов, принятых для данной команды, интерпретация их осуществляется слева направо.

Имена, опции и аргументы обязательно разделяются между собой пробелами. Кроме того, опции обычно предваряются (без пробела) символом дефиса или двойного дефиса. Впрочем, немногочисленные (но весьма употребимые) команды могут использоваться с опциями без всяких предваряющих символов.

Имена команды, ее опции и аргументы образуют т. н. командные "слова". В качестве словоразделителей выступают пробелы. Кроме того, в некоторых командных оболочках как разделители "слов" интерпретируются ряд специальных символов — прямой слэш (/ — элемент пути к файлу), обратный слэш (\), служащий для экранирования специальных символов, операторы командных конструкций, о которых будет сказано ниже. Подчеркнем, что командное "слово" прямо не соотносится ни с опциями, ни с аргументами команды. Введение этого понятия призвано просто облегчить навигацию в командной строке и ее редактирование, о чем будет рассказано в главе о командных оболочках.

Как уже говорилось, имя команды определяет выполняемые ею функции. Существуют команды, встроенные в оболочку, т. е. не имеющие соответствующих им исполняемых файлов, и команды внешние. В последнем случае имя команды — однозначно указывает на имя исполнимого файла программы, выполняемой при отдаче соответствующей директивы. Для хранения имен таких файлов предназначены специальные каталоги $PREFIX/bin — для программ более или менее общего назначения и $PREFIX/sbin — для команд, отнесенных к категории системных (т. е. выполняющих некие административные функции). Впрочем, разделение это не строгое, и одни и те же команды могут в разных системах выступать в качестве пользовательских, в иных же — удостоиться титула системных.

В качестве префикса выступают корневой каталог (/bin, /sbin), каталог /usr (/usr/bin, /usr/sbin), /usr/local (/usr/local/bin, /usr/local/sbin), /usr/X11R6 (/usr/X11R6/bin, /usr/X11R6/sbin). Префиксом может быть также домашний каталог пользователя ($HOME/bin, системных команд здесь, теоретически рассуждая, быть не должно). Обычно все (или большинство) этих каталогов вносятся в переменную PATH командной оболочки пользователя, определяющую, где именно в системе следует искать исполнимые файлы. Если имя файла команды расположено в каком-либо ином каталоге, для ее за-

пуска потребуется указание полного пути в абсолютной или относительной форме.

Как уже говорилось, указания имени достаточно для выполнения некоторых команд. Типичный пример — команда `ls`. Данная без аргументов, она выводит список имен файлов, составляющих текущий каталог, в некоторой форме по умолчанию.

Исполнение многих команд невозможно без указания опций и (или) аргументов. Для них в ответ на ввод одного ее имени часто следует не сообщение об ошибке (или не только оно), но краткая справка по использованию команды. Например, в ответ на ввод

```
$mkdir
```

последует следующий вывод:

```
usage: mkdir [-pv] [-m mode] directory...
```

Некоторые команды могут выступать под несколькими именами. Это связано с тем, что исторически в различных UNIX-системах команды, исполнявшие одинаковые функции, могли получать разные имена. Однако в каждой конкретной системе обычно используется только одна из таких команд-дублеров. Но при этом имена дублирующих команд также могут присутствовать в системе — для совместимости. Не следует думать, что это две различные программы одного назначения: как правило, такая синонимичность команд реализуется посредством механизма жестких или символических ссылок, реже — т. н. псевдонимов (alias).

Иногда команда, вызванная через имя своего синонима, может отличаться по своей функциональности от самой же себя, вызванной под родным именем. В этом случае говорят о эмуляции команды `cmd1` командой `cmd2`. Типичный пример — командные оболочки bash или zsh, которые могут быть вызваны для эмуляции /bin/sh и в этом случае воспроизводят функциональность оболочки Борна.

Опции команды именуются также флагами или параметрами. Правда, можно встретить определение параметра как совокупности опций и аргументов, но я везде буду использовать все три эти термина как синонимы.

Опции определяют условия выполнения команды. Выше был приведен пример команды `ls` без опций и аргументов. Однако на самом деле отсутствием опций определяется вид выводимого списка по умолчанию — как состоящего из имен файлов без учета т. н. скрытых, или dot-файлов, без каких-либо их атрибутов и без различения обычных (регулярных) файлов и каталогов.

Различные же опции команды `ls` определяют состав и формат выводимого списка файлов:

```
$ls -a
```

вывод имен всех файлов, включенных в текущий каталог, включая скрытые файлы вида .*, а также символы текущего (./) и родительского (../) каталогов;

```
$ls -l
```

вывод списка имен файлов в "длинном" (отсюда название опции `l` — long) формате, т. е. с указанием атрибутов доступа, принадлежности, времени модификации, а также размера и количества жестких ссылок;

```
$ls -i
```

вывод списка имен файлов вместе с идентификаторами их inode;

```
$ ls -F
```

вывод с символьным различием каталогов и исполнимых файлов (dir/ и file* соответственно).

Приведенная форма указания опций называется краткой, или односимвольной. Большинство команд допускают также указание опций в длинной, или многосимвольной, форме. А в некоторых случаях многосимвольная форма является единственно возможной — в частности, для команд, количество опций в которых невозможно уместить в символы латинского алфавита. Многосимвольные опции предваряются двойным символом дефиса.

В приведенном примере с командой `ls` достаточно было самого факта указания на опции. Однако для многих команд одна и та же опция может принимать различные значения. Последние в этом случае отделяются от имени опции пробелом или символом равенства. Иногда значения опций необходимо заключать в одинарные или двойные кавычки.

Одна команда может сопровождаться несколькими опциями. В этом случае при односимвольной форме они даются после символа дефиса единым блоком, без пробелов, при многосимвольной — каждая опция отделяется от предыдущей пробелом и предваряется двойным символом дефиса. Символ двойного дефиса с последующим пробелом обычно указывает на окончание списка опций — все последующие компоненты командной строки интерпретируются как аргументы.

Аргументами определяется, как правило, объект (или объекты) действия команды. В большинстве случаев, в соответствии с универсальным методом доступа через файловую систему, в качестве аргументов команд выступают имена файлов и (или) пути к ним. Выше говорилось, что при отсутствии аргументов команда `ls` выводит список имен файлов текущего каталога. Это значит, что текущий каталог выступает как заданный неявным образом (по умолчанию) аргумент команды `ls`. Если же требуется вывести список имен файлов каталога, отличного от текущего, путь к нему должен быть указан в качестве аргумента команды явно, например:

```
$ls /usr/local/bin
```

в абсолютной форме, или

```
$ls../bin
```

в относительной.

Итак, типичная форма UNIX-команды в обобщенном виде выглядит следующим образом:

```
$cmd_name -[otions] [arguments]
```

Из этого правила выбиваются немногочисленные, но весьма полезные и часто используемые команды. Однако и для таких команд с нестандартным синтаксисом устанавливаются те же компоненты — имя, опции, аргументы, хотя по ряду причин (в том числе исторических) порядок их может меняться.

Это можно проиллюстрировать на примере команды `find`, предназначенной для поиска файлов. Подробный разговор о ней впереди, однако пока можно заметить, что в типичной своей форме она выглядит примерно следующим образом:

```
$find dir -option1 value -option2
```

Здесь `dir` — каталог, в котором выполняется поиск, — может рассматриваться в качестве аргумента команды. Опция `-option1` и ее значение `value` определяют критерий поиска, например, `-name file_name` — поиск файла с указанным именем, а опция `-option2` предписывает, что же делать с найденным файлом (файлами), например, `-print` — вывод его имени на экран.

Регулярные выражения

На протяжении предыдущего повествования я неоднократно упоминал, что в качестве аргументов команд (и, в соответствующих случаях, значений опций) могут использоваться не имена файлов, а их маски, или шаблоны имен, подобно знакомым по DOS символами `*`, `?` и т. п. Однако в UNIX-системах маски имен файлов не более чем частный случай более общего явления, именуемого регулярными выражениями. Которые, во-первых, могут использоваться для подстановки не только в именах файлов, но и во многих иных случаях, а во-вторых, много более разнообразны.

Регулярные выражения, образуемые наборами т.н. метасимволов, очень широко применяются для поиска файлов и фрагментов в текстовых файлах. Наиболее распространенные из них — метасимволы `*` и `?`. Первый соответствует произвольному количеству любых символов, в том числе и отсутствию символов вообще (т. е. символу NULL), а также (в отличие от шаблона * в DOS/Windows) точке. Метасимвол же `?` маскирует любой (в том числе и нулевой) одиночный символ.

Регулярное выражение вида `[...]` подменяет любой из символов, заключенных в квадратные скобки. Символы эти могут даваться списком без пробелов (например, выражение `[12345]` соответствует любому символу от 1 до 5) или определяться в диапазоне, крайние значения которого разделяются дефисом без пробелов (эквивалентное первому выражение — `[1-5]`). Кроме того, символ `^`, предваряющий список или диапазон, означает отрицание: выражение `[^abc]` подменяет любой символ, исключая символы a, b и c.

Регулярные выражения в виде списков или диапазонов могут объединяться логическим оператором "ИЛИ", обозначаемым символом `|`, например выражение `[a-c]|[x-z]` подменяет один из символов из диапазонов a—c или x—z. В этом случае внутри аргумента или значения опции команды регулярные выражения должны быть сгруппированы, что обозначается как `(...)`. Например, команда

```
$ls f([aei]|[ou])le
```

выведет список файлов текущего каталога вида fale, fele, file или fole, fule, но не fyle или каких-либо иных.

Приведенные примеры регулярных выражений могут показаться надуманными. Однако это — мощное средство для работы не только с файлами, но и их содержимым, что я попытаюсь показать при обзоре использования команд различного назначения.

Регулярные выражения, перечисленные выше, более или менее универсальны и имеют одинаковый смысл в аргументах и значениях опций различных команд. Однако существует ряд выражений, применимых только в шаблонах поиска файлов и (или) текстовых фрагментов в них. Они обычно несколько различаются в разных программах, и потому я буду говорить о них при рассмотрении конкретных примеров.

Метасимволы, образующие регулярные выражения, суть обычные символы кодовой таблицы ASCII, и их специальное значение определяется тем, как они интерпретируются данной программой. При этом метасимволы могут совпадать с обычными символами в именах файлов или в обрабатываемом тексте. Конечно, использование символов, относимых к категории специальных, в именах файлов очень не рекомендуется (а иногда и прямо запрещается). Однако символы типа круглых или квадратных скобок — совершенно обычная вещь в текстовых фрагментах, выступающих как значения опций или аргументы команд. В этом случае они должны быть экранированы, для чего служит символ обратного слэша \ (не путать с символом / — в UNIX он не является элементом пути к файлу): ввод его указывает команде, что следующий символ не имеет специального значения и должен интерпретироваться буквально (например, как открывающая или закрывающая скобка в тексте). Действие "экрана" распространяется только на непосредственно следующий за ним символ. Если символы, которые могут быть вос-

приняты как специальные, идут подряд, каждый из них должен предваряться обратным слэшем.

В некоторых командах экранирование требуется, напротив, для того, чтобы следующему символу придавался специальный смысл. Примером этому является все та же нестандартная команда `find`. Данная в форме

```
$find. -name *.tar.gz
```

она вместо поиска файлов компрессированных архивов выведет только сообщение об ошибке. Чтобы значение опции было интерпретировано ею как шаблон, она должна быть отдана в виде

```
$find. -name \*.tar.gz
```

на что последует вывод списка всех архивных файлов текущего каталога. Впрочем, для экранирования аргумента или значения опции, содержащего специальный символ, они могут заключаться в одинарные или двойные кавычки.

Следует заметить, что символ обратного слэша экранирует не только явные (т. е. видимые на экране) специальные символы, но и символы неявные. Например, если по вводе команды предварить им нажатие клавиши <Enter>, последнее будет интерпретировано не как сигнал к исполнению команды, а как символ перехода на новую строку, аналогично любому текстовому редактору. И после вновь появившегося приглашения (т. н. вторичного приглашения командной строки) можно продолжить ввод опций и аргументов, но не новой команды. Последнее допускается только в том случае, если новая команда является составным элементом командной конструкции.

Командные конструкции

Командные конструкции — очень важный компонент интерфейса командной строки. Они позволяют объединять несколько команд воедино и выполнять различные команды последовательно или параллельно. Для этого служат специальные символы — операторы: фонового режима, объединения, перенаправления.

Простейшая командная конструкция — это выполнение команды в фоновом режиме, что вызывается вводом символа амперсанда (через пробел) после списка опций и (или аргументов):

```
$cmd [options] [arguments] &
```

После такой команды возвращается приглашение командной строки и возможен ввод любых других команд (в том числе и фоновых). Команды для последующего исполнения можно задать и в той же строке:

```
$cmd1 & cmd2 &... & cmd#
```

В результате все команды, кроме указанной последней, будут выполняться в фоновом режиме.

Далее, существуют конструкции для последовательного выполнения команд. Так, если ряд команд разделен в строке символом ; с последующим пробелом

```
$cmd1; cmd2; cmd3;...; cmd#
```

то сначала будет выполнена команда cmd1, затем — cmd2 и т. д. (молчаливо предполагается, что каждая из этих команд может иметь любое количество опций и аргументов). Сами по себе команды не обязаны быть связанными между собой каким-либо образом — в сущности, это просто эквивалент последовательного их ввода в командной строке:

```
$cmd1
```

выполнение и возвращение приглашения командной строки

```
$cmd2
```

и т. д. При этом первая команда может, например, копировать файлы, вторая — выполнять сортировку и т. д. И естественно, что выполнение последующей команды не зависит от выполнения предшествующей. Исключение — случай, когда результаты предыдущей команды из конструкции

```
$cmd1; cmd2;,,,; cmd#
```

используются в качестве аргументов последующей — в этом случае ошибка исполнения любого компонента конструкции, кроме последнего, делает невозможной работу последующей команды.

В качестве примера можно вспомнить команды, даваемые для компиляции ядра системы, о чем говорилось в *главе 6*:

```
$make depend; make all; make install
```

Очевидно, что ошибка компиляции ядра вызовет невозможность его установки. Именно на случай последовательного выполнения взаимосвязанных команд существует специальная конструкция:

```
$make depend && make all && make install
```

где следует обратить внимание на то, что символы двойного амперсанда обособляются пробелами с обеих сторон. В этом случае компиляция ядра начинает выполняться только после успешного построения файла зависимостей, а установка ядра — только после успешно завершившейся (т. е. не вызвавшей ошибок) его компиляции.

Практически в приведенном примере обе конструкции приведут к одному и тому же результату: очевидно, что в случае ошибки сборки ядра файла, который подлежит установке в качестве /kernel, просто не будет существовать. Однако в ряде иных случаев различие между этими конструкциями может быть существенным.

Условная конструкция, подобная приведенной, может использоваться для вывода сообщения об успешном завершении некоего действия. Например, если создать конструкцию типа

```
$cp file1 file2 && echo "OK"
```

то по успешном завершении копирования команда `echo` выведет подтверждение этому (текст произвольной формы, заключенный в двойные или одинарные кавычки). При ошибке же копирования (например, при отсутствии копируемого файла) никаких сообщений не последует. Вернее, оно будет определяться системой сообщений об ошибках, установленных для данной команды, в данном случае — нечто вроде

```
cp: file1: No such file or directory
```

Впрочем, предусмотрена и командная конструкция, в которой последующей команде предписано исполняться в том и только в том случае, если предыдущая команда завершилась неудачно. Она имеет вид

```
$cmd1 || cmd2
```

и может служить, в частности, для вывода сообщений об ошибках. Например, если модифицировать приведенный выше пример следующим образом:

```
$cp file1 file2 || echo "Копирование не выполнено"
```

то при неудачном завершении операции на экране появится сообщение, заключенное в кавычки (в дополнение к обычному сообщению об ошибках команды `cp`).

Следующая командная конструкция — это так называемое перенаправление ввода/вывода. Любая команда получает данные для своей работы (например, список опций и аргументов) со стандартного устройства ввода (stdin, идентификатор 0), а результаты своей работы представляет на стандартном устройстве вывода (stdout, идентификатор 1). Из главы о системной консоли мы знаем, что в большинстве случаев в качестве первого выступает клавиатура, а в качестве второго — экран монитора.

Обращаясь же далее вглубь первой части, можно вспомнить, что вследствие универсальной модели доступа к устройствам любое из них — не более чем имя специального файла, именуемого файлом устройства. И, таким образом, ничто не запрещает нам подменить файл устройства ввода или устройства вывода любым иным файлом, как специальным файлом устройства, так и регулярным.

Перенаправление ввода обозначается следующим образом:

```
cmd > file
```

или

```
cmd >> file
```

перенаправление вывода —

```
cmd < file
```

или

```
cmd << file
```

В обоих случаях первая форма (одиночный символ > или <) отвечает режиму замещения вводного или выводного потока данными указанного файла, вторая (двойной символ >> или <<) — режиму присоединения к существующему файлу ввода или вывода.

Простейший случай перенаправления — вывод результата исполнения команды не на экран, а в обычный текстовый файл. Например, конструкция

```
$ls > list
```

выведет список файлов текущего каталога не на экран, а в файл list. Если файл с таким именем уже существует, его содержание будет заменено новым (и, соответственно, старое содержание окажется утраченным безвозвратно). Вторая же форма перенаправления осуществляет его в режиме присоединения. Результатом выполнения конструкции

```
$ls >> list
```

будет добавление списка файлов текущего каталога в конец существующего файла.

При перенаправлении ввода команда получает данные для своей работы из входящего в командную конструкцию файла. Например, конструкция

```
$sort < list
```

выведет на экран строки файла list (а это, как мы помним, список файлов текущего каталога), отсортированных в порядке возрастания значения ASCII-кода первого символа, а конструкция

```
$sort -r < list
```

осуществит сортировку тех же строк в обратном порядке.

В одной конструкции могут сочетаться перенаправления ввода и вывода, оба — как в режиме замещения, так и в режиме присоединения. Так, конструкция

```
$sort -r < list > list_r
```

не только выполнит обратную сортировку строк файла list, но и запишет ее результаты в новый файл list_r, а конструкция

```
$sort -r < list >> list
```

добавит по-новому отсортированный список в конец существующего файла list.

Приведенные примеры относились к обычным (регулярным) файлам. Однако те же манипуляции перенаправления могут относиться и к специальным файлам устройств. Так, вывод команды (например, все той же `ls`) может быть направлен не на текущий виртуальный терминал, а на любой другой:

```
$ls > /dev/ttyv#
```

Вывод команды может быть также перенаправлен на устройство печати (/dev/lp) или просто "в ничто" — на устройство, именуемое /dev/null. Последнее — очень удобный способ избавляться от ненужных файлов и сообщений.

Среди сообщений, которые часто могут относиться к категории ненужных — разного рода сообщения об ошибках. По умолчанию они направляются на специальное устройство, весьма неуклюже именуемое устройством вывода сообщений об ошибках (stder, идентификатор `2`). А таковым выступает все тот же экран текущего терминала, что далеко не всегда удобно. Например, при использовании команд `find` (поиск файлов) или `grep` (поиск текстовых фрагментов или шаблонов) за пределами домашнего каталога пользователя, экран мгновенно загромождается сообщениями об отсутствии доступа к многим каталогам.

И потому часто удобно перенаправить все сообщения об ошибках на иной виртуальный терминал, в специально предназначенный для этого файл или просто в небытие (/dev/null). Делается это следующим образом:

```
$cmd [options] [arguments] 2> /dev/ttyv#
```

или

```
$cmd [options] [arguments] 2> /dev/null
```

Обращаю внимание, что, в отличие от обычного перенаправления, при использовании идентификаторов стандартных устройств отсутствует пробел — ввод его вызовет в этом случае сообщение об ошибке типа

```
cmd: 2: unknown option
```

Однако бывают и обратные случаи, когда требуется увековечить все сообщения программы — и ее вывод, и сведения об ошибках. Для этого выводной поток и поток ошибок можно слить воедино и перенаправить в некий файл. Для этого оператор перенаправления используется в следующем виде: `n>$m`, т. е. файл с идентификатором `n` перенаправляется в тот же файл, что и файл с идентификатором `m`. Например, для требуемого нам случая это примет форму:

```
$cmd [options] [arguments] > messages 2>&1
```

которая предписывает направить поток сообщений об ошибках в тот же файл messages, что и стандартный вывод команды `cmd`.

Возможности построения командных конструкций не ограничиваются перенаправлением ввода/вывода: результаты работы одной команды могут быть переданы для обработки другой команде. Это достигается благодаря механизму программных каналов (pipe) или конвейеров — последний термин лучше отражает существо дела.

При конвейеризации команд стандартный вывод первой команды передается не на устройство вывода или в файл, а на стандартный ввод следующей команды. Простой пример такой операции — просмотр списка файлов:

```
$ls -l | less
```

Перенаправление вывода команды `ls`, т. е. списка файлов, который при использовании полного формата записи (опция `-l`) может занимать многие экраны, на ввод команды `less` позволяет просматривать результат с ее помощью постранично или построчно в обоих направлениях.

Конвейеризация команд может быть сколь угодно длинной. Так, конструкция

```
$ls -lt | sort -r | lpr
```

обеспечит не только сортировку списка файлов каталога в полном формате, выведенного командой `ls` по времени изменения (опция `-t`), в обратном относительно исходного порядке (опция `-r` команды `sort`), но и выведет его на печатающее устройство командой `lpr`.

Перенаправление ввода/вывода и конвейеризация команд могут объединяться в единой конструкции. Например, с помощью

```
$cat file1... file# > allfile | less
```

можно не только слить несколько файлов воедино, но и просмотреть результат на экране (впрочем, того же результата можно добиться командой `tee`, как будет показано в одной из следующих глав.

Кроме того, команды в конструкции могут быть сгруппированы с тем, чтобы они выполнялись как единое целое. Для этого группа команд разделяется символами `;` и пробелами, как при последовательном выполнении команд, и заключается в фигурные скобки. Результатом работы группы команд будет результат исполнения последней ее составляющей.

Для группировки команд могут использоваться и круглые скобки. В этом случае для исполнения группы команд будет вызываться новый (порожденный) экземпляр командной оболочки, возможно, как будет показано в следующей главе, с отличными от текущего экземпляра свойствами.

Группировка команд может использоваться совместно с их перенаправлением. Так, чтобы собрать вывод нескольких команд в одном и том же файле, вместо конструкции

```
$cmd1 > file; cmd2 >> file;...; cmd# >> file
```

можно прибегнуть к конструкции

```
${ cmd1 ; cmd2 ;... ; cmd# } > file
```

Надо заметить (хотя это и немного не из той оперы), что группировка может распространяться и на аргументы команд (хотя и не во всех командных оболочках). Так, посредством конструкции

```
$mkdir dir1 dir1/{dir11,dir12,dir12/{dir121,dir122},dir13}
```

можно в один присест создать внутреннюю структуру подкаталогов, начиная с корневого каталога dir1, т. е. каталоги dir1, dir1/dir11, и т. д. В чем можно убедиться посредством:

```
$ls dir1
dir11/ dir12/ dir13/
$ls dir1/dir12
dir121/ dir122/
```

Командные конструкции могут быть сколь угодно длинными, и работа с ними в единой строке окажется неудобной. Поэтому можно прибегнуть к многострочному вводу конструкций. Для этого каждая из вводимых строк должна завершаться символом обратного слэша, предваряемым пробелом:

```
$cmd1 [options] [arguments] [оператор_конструкции] \
@cmd2 [options] [arguments] [оператор_конструкции] \
...
@cmd# [options] [arguments] [оператор_конструкции]
```

Здесь символ @ объявляет вторичное приглашение командной строки, которому в большинстве оболочек можно придать иной вид, нежели первичному, о чем будет рассказано в соответствующем месте. Ну и очевидно, что в последней строке многострочной команды символ обратного слэша не ставится.

Понятие о пользовательских сценариях

Ввод многострочных команд и командных конструкций, да еще с многочисленными опциями и сложными аргументами может показаться занятием долгим и утомительным. Но это не так. Во-первых, как будет показано в следующей главе, все современные развитые командные оболочки обладают мощными средствами автоматизации и упрощения интерактивной работы с командной строкой. А во-вторых, и это — главное, интерактивная работа в командной строке дополняется обращением к так называемым сценариям оболочки, или скриптам.

В общем случае сценарий — это просто одна или несколько команд и (или) командных конструкций с необходимыми опциями и аргументами, сохра-

ненные в виде обычного именованного текстового файла, которому присвоен атрибут исполнения хотя бы для одного из атрибутов принадлежности.

Сценарии в UNIX-системах несколько напоминают batch-файлы, памятные многим по временам MS-DOS, или макросы, знакомые пользователям любых развитых прикладных пакетов. Подобно тем и другим, они предназначены в первую очередь для автоматизации часто исполняемых рутинных операций. В частности, именно они позволяют избежать ввода длинных командных конструкций.

Однако значение сценариев не исчерпывается удовлетворением естественной человеческой лени. Они позволяют решать очень широкий круг задач — от конфигурирования системы (большинство конфигурационных файлов FreeBSD суть сценарии оболочки) до создания весьма сложных приложений и даже интерактивных Web-страниц. Именно практически неограниченным возможностям создания пользовательских сценариев интерфейс командной строки обязан своей эффективностью.

Создание пользовательского сценария — просто как правда. Для этого требуется:

- создать пустой файл;
- любым способом поместить в него требуемые команды, их опции и аргументы, необходимые операторы конструкций и т. д. в виде произвольного количества строк;
- при желании (и по необходимости) снабдить команды комментариями;
- сохранить файл в некоем каталоге, к которому создатель имеет доступ;
- командой `chmod` присвоить созданному файлу атрибут исполнения хотя бы для себя, любимого;
- запустить сценарий при потребности.

Создать файл можно в текстовом редакторе или (в случае короткого сценария) командой `cat`, о чем будет рассказано в соответствующей главе. Предназначенные для сценария команды могут быть набраны вручную или скопированы из командной строки мышью и при необходимости отредактированы. Ясно, что в виде сценариев следует сохранять команды или командные последовательности, по крайней мере более длинные, чем, имя скрипта. Последнее желательно дать а) коротким, б) мнемонически значимым и в) не совпадающим с именем встроенных команд оболочки или команд системы.

Наиболее подходящий каталог для файлов пользовательских сценариев — *$HOME/bin*, который желательно внести в переменную среды `PATH` данного пользователя, в этом случае для запуска скрипта достаточно будет задать в командной строке его имя без указания полного пути.

Строки с комментарием предваряются символом # — аналогично комментарию rem в batch-файлах MS DOS. Символы строки, следующие за ним, не рассматриваются как команды и не принимаются к исполнению, а служат исключительно внутренним целям, поясняя в необходимых случаях действия командных директив. Если комментарий занимает более чем одну строку, каждая из них должна начинаться с символа #.

Большинство сценариев начинается со строки вида:

```
#!/path/Shell_name
```

имеющей особый смысл. Это не комментарий в собственном смысле слова, а указание на имя и местоположение программы, которая должна запускаться для исполнения сценария. Дело в том, что сценарий исполняется не в том экземпляре командной оболочки, из которого запущен: для этого порождается новый процесс, в котором и запускается новая копия командного интерпретатора. Поэтому желательно, чтобы имя последнего было задано в явном виде.

Конечно, если имя командой оболочки не указано, это обычно не помешает исполнению сценария: просто для этой цели будет вызван командный интерпретатор по умолчанию, каковым в FreeBSD выступает /bin/sh. Однако строка вида

```
#!/bin/sh
```

является признаком хорошего тона при создании сценария. А если сценарий предназначен для исполнения в какой-либо другой командной оболочке, отличающейся синтаксически (например, в csh/tcsh) или использует дополнительные возможности мощных современных оболочек (например, zsh), сценарий обязательно должен начинаться с явного указания на них, например:

```
#!/bin/csh
```

или, соответственно,

```
#!/usr/local/bin/zsh
```

Если имя оболочки не указывается в сценарии явным образом, не следует помещать в первой строке комментарий, начинающийся с символа # — в некоторых случаях это может быть интерпретировано как вызов конкретной командной оболочки (в частности, csh). Лучше оставить первую строку пустой или начать сценарий с команды :, отвечающей пустому оператору, не совершающему никакого действия.

Изменение прав доступа к файлу сценария — очень важный момент: только наличие атрибута исполнения по крайней мере для одного из атрибутов принадлежности (пользователя, группы или прочих) отличает пользовательский сценарий от просто текстового файла со списком команд. Делается это многократно упоминавшейся командой chmod, например:

```
$chmod u+x script_name
```

для себя, любимого, или с опциями `g+x`, `o+x`, `a+x`, если есть желание поделиться своим скриптом с народом: группой, прочими, а то и всеми вместе.

Впрочем, пользовательский сценарий может быть выполнен и без присвоения ему атрибута исполнения: для этого следует только явным образом вызвать любой совместимый с ним командный интерпретатор и указать имя сценария в качестве аргумента, например:

```
$/bin/sh script_name
```

Вызываемый командный интерпретатор не обязан совпадать с пользовательской оболочкой (login Shell) — его существование как процесса прекратится после завершения исполнения сценария.

Наконец, можно выполнить сценарий без атрибута исполнения и в текущем экземпляре командной оболочки. Для этого существует специфическая команда, встроенная во все командные оболочки и обозначаемая символом точки:

```
$. script_name
```

Подчеркну, что в данном случае точка — это именно имя встроенной команды, а не символическое обозначение текущего каталога; при необходимости явного указания на последний это делается так:

```
$../script_name
```

Кроме описанных выше командных конструкций, в сценариях могут использоваться и различные операторы:

- циклические (`for`, `while`, `until`);
- условные (`if... then... else`);
- вызова (`case`).

Это делает их пригодными для реализации сложных прикладных задач. Однако синтаксис операторов в наибольшей степени зависит от конкретной командной оболочки — в частности, именно в этом случае проявляются принципиальные отличия Shell-совместимых оболочек от csh-совместимых.

Так, в Shell-совместимых оболочках условная конструкция имеет форму:

```
if условие
then
  cmd1
else
  cmd2
fi
```

Под условием здесь понимается просто команда, которая может завершиться успешно (с выдачей кода завершения 0) или безуспешно (ошибочно с кодом

завершения, отличным от нуля). В первом случае выполняется команда `cmd1`, во втором — `cmd2`.

В csh-совместимых оболочках условная конструкция приобретает форму:

```
if (выражение) then
  cmd1
else
  cmd1
endif
```

То есть условие определяется результатом не выполнения команды, а вычисления арифметического или логического выражения.

Для создания пользовательских сценариев можно использовать команду `script`. После ее ввода все действия пользователя по вводу в командной строке протоколируются в файле с именем typescript (или — с именем, указанным в качестве значения опции `-a`). Завершается режим протоколирования командой `exit` (или нажатием комбинации <Ctrl>+<D>). Созданный файл протокола может быть в дальнейшем отредактирован обычным способом.

Вообще говоря, создание пользовательских сценариев — отдельная, и очень обширная, тема, рассмотрение которой далеко выходит за рамки настоящей книги. Для ее углубленного изучения целесообразно обратиться к дополнительным источникам информации.

Дополнительные источники

- Одно из лучших пособий по ознакомлению с принципами командного интерфейса — весьма древний, по меркам информационной индустрии, труд: Керниган Б. В., Пайк Р. UNIX — универсальная среда программирования. Пер. с англ. А. М. Березко и В. А. Иващенко. М.: Финансы и Статистика, 1992.
- Правда, боюсь, он уже — библиографическая редкость. И потому для замены его можно обратиться к прекрасному пособию Александра Соловьева "Программирование на Shell (UNIX)", доступное на нескольких сайтах, например, пермской группы пользователей Linux (**http://linux.perm.ru**).
- Почти исчерпывающие сведения о работе в командной оболочке можно найти в книге Тэйнсли Д. Linux и UNIX: программирование в Shell. — К.: Издательская группа BHV, 2001.
- Наконец, книга Рассела Сейджа "Приемы профессиональной работы в UNIX", русский перевод которой доступен, например, по адресу **http://lib.ru/LINUXGUIDE/profunix.txt**, также в значительной своей мере посвящена именно принципам (и практическим приемам) использования интерфейса командной строки.

Глава 14

Обзор пользовательских команд

На протяжении всего предыдущего повествования неоднократно (и, как правило, без комментариев) упоминались команды самого различного назначения. Наступил психологический момент навести порядок в этом вопросе и рассмотреть классификацию команд, дать их описание, а также примеры применения.

Команд в FreeBSD насчитывается весьма много: даже не выходя за пределы базовой системы (без учета дополнительных пакетов), их можно насчитать около 750: для этого достаточно подсчитать количество файлов в каталогах /bin, /sbin, /usr/bin и /usr/sbin. Детальное их описание — задача непосильная да, пожалуй, и ненужная. И потому ниже я остановлюсь только на командах, наиболее интересных с точки зрения типичных пользовательских задач. Однако, поскольку на локальной машине пользователь должен быть и администратором, командам администрирования системы также будет уделено некоторое внимание.

Думаю, не ошибусь, если скажу, что основными действиями пользователя на любой платформе, за исключением специальных задач, решаемых специфическими программными средствами, являются управление файлами и обработка их содержимого. Кроме того, в многопользовательской и многозадачной системе, подобной FreeBSD, очень важно получение разного рода общесистемной информации — и не только для администратора, но и для пользователя. Именно этим вопросам и будет уделено основное внимание в настоящей главе. В заключение же ее — несколько небольших утилит разного назначения, которые могут упростить жизнь пользователю.

Следует учесть, однако, что некоторые команды являются многофункциональными, и потому о различных аспектах их применения речь пойдет в разных разделах этой главы.

Все рассмотренные здесь команды входят в штатный комплект FreeBSD, располагаясь в каталогах /bin, /usr/bin, реже — /sbin и /usr/sbin. Это, так сказать, внутренние ресурсы системы, без использования портов или пакетов.

Команды для работы с файлами

В этом разделе будут рассмотрены команды, предназначенные для операций с файлами как целостными сущностями, не затрагивая их содержимого.

Работа с файлами, в сущности, сводится к:

- их созданию;
- навигации по файловой системе;
- получению информации о файлах;
- манипулированию существующими файлами;
- архивации и компрессии.

Всем этим целям в FreeBSD служат многочисленные специальные команды. Кроме того, имеется команда `find`, представляющая собой почти универсальное средство решения всех поставленных задач.

Создание файлов

Поскольку для того, чтобы манипулировать файлами, нужно как минимум иметь файлы, рассмотрение начнем с команд для их создания.

Разумеется, в большинстве случаев файлы создаются специализированными прикладными пакетами — текстовыми процессорами и редакторами, электронными таблицами, редакторами растровой и векторной графики и т. д. Однако в ряде случаев для создания простых текстовых (или просто пустых) файлов вызов громоздких приложений нецелесообразен. Кроме того, файлы специальных типов и не могут быть созданы прикладными пакетами — для этого требуются специальные средства.

Однако начнем по порядку — с создания обычного (регулярного) пустого файла; такая задача может возникнуть, например, при отработке системы навигации по сайту. Для этой цели может использоваться команда `touch` без опций, с именем несуществующего файла в качестве аргумента:

```
$touch filename
```

Хотя вообще-то команда `touch` предназначена для установки временных атрибутов файла, о чем будет сказано в свое время.

Для создания небольших текстовых файлов (типа простых сценариев) можно применить команду `cat`. Делается это так: отдается команда `cat` с перенаправлением вывода в несуществующий файл

```
$cat > filename
```

После этого следует нажать <Enter>, ввести требующийся текст, при необходимости — многострочный, создавая новые строки той же клавишей <Enter>, и ввести символ окончания файла — комбинацией <Ctrl>+<Z>,

или, правильнее, <Ctrl>+<D> — в ответ на нее последует сообщение об удачном создании файла. При этом символ конца файла в любом случае необходимо предварить переходом на новую строку (нажатием клавиши <Enter>) — иначе он будет интерпретирован как специальный символ внутри текста.

Если перенаправить вывод команды `cat` в режиме присоединения в существующий файл

```
$cat >> filename
```

последний может быть дополнен с клавиатуры новым текстом. Хотя это, разумеется, не лучший способ создания многотомных романов (и даже небольших заметок).

Интересно создание файлов с помощью команды `tee`. Смысл ее — в раздвоении входного потока, выводимого одновременно и на стандартный вывод, и в файл, указанный в качестве ее аргумента. То есть если использовать ее для создания файла с клавиатуры, это выглядит так, будто строки удваиваются на экране. Но это не так: просто весь вводимый текст копируется одновременно и на экран, и в файл. И потому ее удобно применять в командных конструкциях, когда требуется одновременно и просмотреть результаты исполнения какой-либо команды, и запечатлеть их в файле:

```
$ls dir | tee filename
```

По умолчанию команда `tee` создает новый файл с указанным именем или перезаписывает одноименный, если он существовал ранее. Однако с опцией `-a` она добавляет новые данные в конец существующего файла.

Вслед за созданием файлов необходимый шаг — это создание каталогов. Для этого предназначена команда `mkdir`, в качестве аргумента которой выступает имя создаваемого каталога (или каталогов — одной командой можно создать любое их количество). При этом могут создаваться подкаталоги любого уровня вложенности, нужно только указать промежуточные родительские каталоги:

```
$mkdir ~/dir1/dir2/dir3/newdir
```

Если родительский по отношению к создаваемому каталог отсутствует, он также может быть создан посредством опции `-p`, и опять же с любым уровнем вложенности:

```
$mkdir -p ~/dir1/newdir1/newdir2/newdir3
```

Одновременно с созданием каталога для него можно задать произвольные атрибуты доступа (по умолчанию создаваемый каталог получает атрибуты доступа `rwx` для владельца и `r-x` — для группы и прочих): для этого потребуется опция `-m`, значением которой являются атрибуты доступа в символьной форме, например:

```
$mkdir -m 777 newdir
```

для предоставления полного доступа к создаваемому каталогу, или

```
$mkdir -m 700 newdir
```

для закрытия доступа к каталогу всем, кроме его владельца.

Далее, перед пользователем может возникнуть необходимость в создании ссылок обоих видов — как жестких, так и символических. И то, и другое делается одной командой — `ln`. Для создания обычной (жесткой) ссылки достаточно дать ее без опций с указанием имени файла и затем — имени ссылки, т. е. — просто второго имени для того же набора данных. Жесткая ссылка может быть создана только на обычный или специальный файл, но не на каталог. Кроме того, она не может пересекать границы файловой системой.

Символическая ссылка таких ограничений не имеет. Она создается тем же образом, что и жесткая, но с указанием опции `-s`:

```
$ln -s name linkname
```

При этом, если `name` является именем каталога, автоматически создаются символические ссылки на все входящие в его состав файлы и вложенные подкаталоги.

Если имя ссылки совпадает с именем существующего файла (любого типа), по умолчанию она создана не будет. Однако это можно сделать принудительно, указав опцию `-f`, в этом случае существующий файл, одноименный создаваемой ссылке, будет удален. Опция же `-i` дополнительно запросит подтверждение на производство удаления.

Для создания специальных файлов устройств применяется специальная же команда — `mknod`. Она требует указания имени создаваемого устройства, затем — его типа (`c` — символьное устройство, `b` — блочное), а также старшего (`major` — номер родовой группы устройств) и младшего (`minor` — порядковый номер устройства в своей группе) номера устройства:

```
$mknod /dev/dev_name [c|b] maior# minor#
```

После этого дополнительно можно указать владельца (`owner`) и группу принадлежности (`group`) для создаваемого файла устройства. Впрочем, в большинстве случаев для создания файлов устройств используется сценарий `/dev/MAKEDEV`, позволяющий автоматизировать этот процесс (в частности, делающий ненужным знание старшего и младшего номеров устройств).

Наконец, есть и `mkfifo` — специальная команда для создания именованных каналов — специальных файлов, предназначенных для обмена данными между процессами. Однако у пользователя вряд ли может возникнуть необходимость в обращении к ней (мне, например, не пришлось ей воспользоваться ни разу в жизни). Так что на этом можно закрыть вопрос создания файлов.

Навигация по файловой системе

Следующие действия, необходимые пользователю, — ориентация среди существующих файлов и каталогов. Для начала при этом неплохо определиться со своим текущим положением в файловой системе. Для этого предназначена команда `pwd`, в ответ на которую выводится полный путь к текущему каталогу. Например, если текущим является домашний каталог пользователя, в ответ на:

```
$pwd
```

последует

```
/home/username
```

Команда `pwd` имеет всего две опции: `-L` и `-P`. Первая выводит т. н. логический путь к текущему каталогу. Если таковым является, скажем, каталог /sys (представляющий собой символическую ссылку на каталог /usr/src/sys), то в ответ на

```
$pwd -L
```

так и будет выведено

```
/sys
```

Впрочем, тот же ответ последует и на команду `pwd` без опций вообще. Если же дать эту команду в форме

```
$pwd -P
```

будет выведен путь к физическому каталогу, на который ссылается текущий:

```
/usr/src/sys
```

Далее, по каталогам неплохо как-то перемещаться. Что делается многократно упоминавшейся ранее командой `cd`. В отличие от прочих команд, рассматриваемых в этом разделе, это — внутренняя команда, встроенная во все командные оболочки — бесполезно было бы искать соответствующий ей исполняемый файл. Однако это не уменьшает ее важности. Использование ее очень просто —

```
cd pathname
```

где `pathname` — путь к искомому каталогу в абсолютной (относительно корня) или относительной (относительно текущего каталога) форме. Впрочем, на этот счет было уже сказано достаточно.

Определить местоположение команды (и вообще исполняемых файлов) в структуре файловой системы можно с помощью команды `which`. В качестве аргумента ее можно указать одно или несколько имен файлов, в ответ на что будет выведен полный путь к каждому из них:

```
$which csh tcsh zsh
/bin/csh
```

```
/bin/tcsh
/usr/local/bin/zsh
```

При наличии одноименных исполняемых файлов в разных каталогах по умолчанию будет выведен путь только к первому из них: для вывода всех файлов-"тезок" можно прибегнуть к опции `-a`.

Более широкие возможности поиска — у команды `whereis`. По умолчанию, без опций, она для заданного в качестве аргумента имени выводит список бинарных файлов, man-страниц и каталогов с исходными текстами:

```
$whereis zsh
zsh: /usr/local/bin/zsh /usr/local/man/man1/zsh.1.gz
/usr/ports/Shells/zsh
```

Соответствующими опциями можно задать поиск файлов одного из этих типов: `-b` — бинарных, `-m` — страниц руководств, `-s` — каталогов с исходниками. Дополнительные опции `-B`, `-M`, `-S` (в сочетании с опцией `-f`) позволяют определить исходные каталоги для их поиска.

Наконец, команда `locate` осуществляет поиск всех файлов и каталогов, содержащих компонент имени, указанный в качестве аргумента, и вывод содержимого найденных каталогов. Так, в ответ на команду

```
$locate zsh
```

будет выведен список вроде следующего:

```
/usr/local/bin/zsh
/usr/local/info/zsh.info
...
/usr/local/lib/zsh
...
/usr/local/man/man1/zsh.1.gz
...
/usr/local/share/doc/zsh
...
/usr/ports/distfiles/zsh-4.0.4-doc.tar.gz
/usr/ports/distfiles/zsh-4.0.4.tar.gz
/usr/ports/Shells/zsh
...
```

и т. д. Команда `locate` при этом обращается к базе данных, расположенной в каталоге /var/db/, т. е. по адресу /var/db/locate.database. По умолчанию она пуста — и перед использованием команды `locate` должна быть наполнена содержанием. Для этого предназначен сценарий /usr/libexec/locate.updatedb, извлекающий сведения из базы данных пакетов — /var/db/pkg. При актив-

ной доустановке пакетов база эта нуждается в периодическом обновлении, что можно сделать с помощью сценария /etc/periodic/weekly/310.locate.

Приведенные команды относятся к командам поиска исполнимых файлов и программных компонентов. Однако чаще перед пользователем возникает необходимость поиска неких произвольных файлов. На сей предмет существует команда `find`. Однако возможности ее не сводятся к поиску — это практически универсальный инструмент для файловых операций. И потому она будет подробно рассмотрена под занавес — в конце раздела о работе с файлами.

Получение информации о файлах

Наиболее универсальным средством получения практически исчерпывающей информации о файлах является команда

```
$ls [options] names
```

где в качестве аргумента `names` могут выступать имена файлов или каталогов в любом количестве. Команда эта многократно упоминалась на протяжении всего предшествующего повествования, поэтому здесь остается только систематизировать уже известные сведения. Для чего рассмотрим основные из крайне многочисленных опций команды `ls`.

Начать с того, что команда `ls`, данная без всяких опций, по умолчанию выводит только имена файлов, причем опуская т. н. dot-файлы, имена которых начинаются с точки (это — некие аналоги hidden-файлов в MS-DOS). Кроме того, если в качестве аргумента указано имя каталога (или аргумент не указан вообще, что подразумевает текущий каталог), из списка имен его файлов не выводятся текущий (.) и родительский (..) каталоги.

Для вывода всех без исключения имен файлов (в том числе и скрытых) предназначена опция `-a`. Смысл опции `-A` близок — она выводит список имен всех файлов, за исключением имен текущего (.) и родительского (..) каталогов.

Как говорилось в *главе 9*, кроме своего имени файл идентифицируется своим номером `inode`. Для его вывода используется опция `-i`:

```
$ls -i
694402 dir1     673058 dir3     673059 file2
694404 dir2     673099 file1    673057 file3
```

Как и многие другие, команда `ls` обладает способностью рекурсивной обработки, для чего предназначена опция `-R`, выводящая список имен файлов не *только текущего каталога, но и всех вложенных подкаталогов*:

```
$ls -R
dir1    dir2    dir3    file1   file2   file3
```

```
./dir1:
f1      f2      f3

./dir2:
dir4    f1      f2      f3

./dir2/dir4:
f1      f2      f3
```

В выводе команды `ls` по умолчанию имена файлов разных типов даются абсолютно одинаково. Для их визуального различия используется опция `-F`, завершающая имена каталогов символом слэша, исполнимых файлов — символом звездочки, символических ссылок — "собакой"; имена регулярных файлов, не имеющих атрибута исполнения, никакого символа не включают:

```
$ls -F
dir1/   dir2/   dir3@   file1   file2*  file3@
```

Другое средство для визуального различия типов файлов — колоризация, для чего применяется опция `-G`. Цвета шрифта, воспроизводящего имена, по умолчанию — синий для каталогов, лиловый (`magenta`) для символических ссылок, красный — исполнимых файлов и т. д. Для файлов устройств, исполнимых файлов с атрибутом "суидности", каталогов, имеющих атрибут `sticky`, дополнительно колоризуется и фон, на котором выводится шрифт, воспроизводящий их имена. Подробности можно посмотреть в секции ENVIRONMENT man-страницы для команды `ls`.

По умолчанию команда `ls` выводит список файлов в порядке ASCII-кода первого символа имени. Однако есть возможность его сортировки в обратном порядке (`-r`), в порядке времени модификации (`-t`) или времени доступа (`-tu`). Кроме того, опция `-f` отменяет какую-либо сортировку списка вообще.

Информацию об объеме файлов можно получить, используя опцию `-s`, выводящую для имени каждого файла его размер в блоках, а также суммарный объем всех выведенных файлов:

```
$ls -s../book
total 822
 656 book.html              4 content1.html          86 var_part2.html
 24 command.html           38 part2.html
  6 command.txt             8 Shell_tmp.html
```

Добавление к опции `-s` еще и опции `-k` (`ls -sk`) выведет всю ту же информацию в килобайтах.

Как можно видеть из всех приведенных выше примеров, списки файлов по команде `ls` выводятся в многоколоночном виде (чему соответствует опция

-C, однако указывать ее нет необходимости — многоколоночный вид принят для краткого формата по умолчанию). Но можно задать и одноколоночное представление списка посредством опции -1:

```
$ls -1
dir1
dir2
dir3
file1
file2
file3
```

До сих пор речь шла о кратком формате команды ls. Однако более информативным является т. н. длинный ее формат, вывод в котором достигается опцией -l и автоматически влечет за собой одноколоночное представление списка:

```
ls -l
total 8
drwxr-xr-x  2 alv  alv  512  8 май 18:04 dir1
drwxr-xr-x  3 alv  alv  512  8 май 17:43 dir2
lrwxr-xr-x  1 alv  alv    4  9 май 07:59 dir3 -> dir2
-rw-r--r--  1 alv  alv   14  8 май 10:39 file1
-rwxr-xr-x  1 alv  alv   30  9 май 08:02 file2
lrwxr-xr-x  1 alv  alv    2  8 май 10:57 file3 -> f1
```

Можно видеть, что по умолчанию в длинном формате выводятся:

- сведения о типе файла (- — регулярный файл, d — каталог, l — символическая ссылка, c — файл символьного устройства, b — файл блочного устройства) и атрибуты доступа для различных атрибутов принадлежности (о чем было сказано достаточно);
- количество жестких ссылок на данный идентификатор inode;
- имя пользователя — владельца файла и группы пользователей, которой файл принадлежит;
- размер файла в блоках;
- время модификации файла с точностью до месяца, дня, часа и минуты (в формате, принятом в данной локали);
- имя файла и (для символических ссылок) имя файла-источника.

Однако это еще не все. Добавив к команде ls -l еще и опцию -i, можно дополнительно получить идентификатор inode каждого файла, опция -n заменит имя владельца и группу на их численные идентификаторы (UID и

GUID соответственно), а опция -T выведет в поле времени модификации еще и годы, и секунды:

```
ls -linT
total 8
694402 drwxr-xr-x  2 1000  1000  512  8 май 18:04:56 2002 dir1
694404 drwxr-xr-x  3 1000  1000  512  8 май 17:43:31 2002 dir2
673058 lrwxr-xr-x  1 1000  1000    4  9 май 07:59:08 2002 dir3 -> dir2
673099 -rw-r--r--  1 1000  1000   14  8 май 10:39:38 2002 file1
673059 -rwxr-xr-x  1 1000  1000   30  9 май 08:02:23 2002 file2
673057 lrwxr-xr-x  1 1000  1000    2  8 май 10:57:07 2002 file3 -> f1
```

Разумеется, никто не запрещает использовать в длинном формате и опции визуализации (-F и -G), и опции сортировки (-r, -t, -tu), и любые другие, за исключением опции -C — указание ее ведет к принудительному выводу списка в многоколоночной форме, что естественным образом подавляет длинный формат представления.

Я столь подробно остановился на описании команды ls, потому что это — основное средство визуализации файловой системы FreeBSD, при умелом использовании ничуть не уступающее развитым файловым менеджерам (типа Midnight Commander или Konqueror) по своей выразительности и информативности. И отнюдь не требующее для достижения таковых использования многочисленных опций: в главе о командных оболочках будет показано, что соответствующей настройкой последних можно добиться любого "умолчального" вывода команды ls.

Существуют и другие команды для получения информации о файлах. Например, команда под характерным именем file с аргументом в виде имени файла в состоянии определить тип его с большой детальностью. Так, для регулярных файлов она распознает:

- исполняемые бинарные файлы с указанием их формата (например, ELF), архитектуры процессоров, для которых они скомпилированы, характер связи с разделяемыми библиотеками (статический или динамический);
- исполняемые сценарии с указанием оболочки, для которой они созданы;
- текстовые и html-документы, часто с указанием используемого набора символов.

Последнему, впрочем, для русскоязычных документов доверять особо не следует: кодировка KOI8-R в них вполне может быть обозвана ISO-8859.

Определяет она так же каталоги, символические ссылки, специальные файлы устройств, указывая для последних старшие и младшие номера устройств.

Наконец, команда stat (с именем файла в качестве аргумента) выводит большую часть существенных сведений о файле в удобном для восприятия

виде, например, включая идентификатор inode, режим доступа (в символьной форме), идентификаторы владельца и группы, временные атрибуты, количество жестких и символических ссылок.

Приведенных способов получения информации о файле, как кажется, пользователю должно быть достаточно. Перейдем к манипуляциям с существующими файлами — копированию, перемещению, переименованию, удалению.

Манипулирование файлами

Начнем с копирования — это выполняется очень простой командой cp, имеющей, однако, весьма разнообразные аспекты применения. В самом простом своем виде она требует всего двух аргументов — имени файла-источника на первом месте и имени целевого файла — на втором:

```
$cp file_source file_target
```

Этим в текущем каталоге создается новый файл (file_target), идентичный по содержанию копируемому (file_source). То есть область данных первого будет дублировать таковую последнего. Однако области метаданных у них будут различны изначально. Целевой файл — это именно новый файл со своим идентификатором inode, заведомо иными временными атрибутами; его атрибуты доступа и принадлежности в общем случае также не обязаны совпадать с таковыми файла-источника.

Новый файл может быть создан и в произвольном каталоге, к которому пользователь имеет соответствующий доступ: для этого следует только указать полный путь к нему:

```
$cp file_source dir/subdir/file_target
```

Если в качестве второго аргумента команды указано просто имя каталога, то новый файл будет создан в нем с именем, идентичным имени файла-источника. Однако подчеркну, что в любом случае копирования создается именно новый файл, никак после этого не связанный с файлом исходным.

Если в качестве последнего аргумента выступает имя каталога, он может предваряться любым количеством аргументов — имен файлов:

```
$cp file1 file2... file3 dir/
```

В этом случае в целевом каталоге dir/ будут созданы новые файлы, идентичные по содержанию файлам file1, file2 и т. п.

Если в целевом (или текущем) каталоге уже имеется файл с именем, совпадающим с именем вновь создаваемого файла, он в общем случае будет без предупреждения заменен новым файлом. Единственное средство для пре-

дотвращения этого — задание опции -i (от interactive) — при ее наличии последует запрос на перезапись существующего файла:

```
$cp -i file1 file2
overwrite file2? (y/n [n])
```

Как будет показано в главе о командных оболочках, некоторые из них (например, zsh) могут быть настроены так, чтобы по умолчанию не допускать перезаписи существующих файлов. Однако если такая потребность осознанно возникнет, это можно выполнить с помощью опции -f (от force). К слову сказать, она также аннулирует действие опции -i, например, при использовании ее в псевдониме команды cp.

Имя каталога может выступать и в качестве первого аргумента команды cp. Однако это потребует опции -R (иногда допустима и опция -r — в обоих случаях от recursive). В этом случае второй аргумент также будет воспринят как имя каталога, который не только будет создан при этом, но в нем также будет рекурсивно воспроизведено содержимое каталога источника (включая и вложенные подкаталоги).

При копировании файлов, представляющих собой символические ссылки, они будут преобразованы в регулярные файлы, копирующие содержимое файлов — источников ссылки. Однако при рекурсивном копировании каталогов, содержащих символические ссылки, возможно их воспроизведение в первозданном виде. Для этого вместе с опцией -R должна быть указана одна из опций -H или -L. Однако обе они при отсутствии -R игнорируются.

Как уже было сказано, создаваемые при копировании целевые файлы по умолчанию получают атрибуты доступа и времени, не зависящие от таковых файла-источника. Обычно они определяются значением переменной umask, заданной глобально, через класс учетной записи пользователя или в профильном файле командной оболочки пользователя (по умолчанию в FreeBSD значение umask — 022, о чем было сказано в *главе 9*). Однако при желании атрибуты исходного файла можно сохранить в файле целевом — для этого предназначена опция -p. Разумеется, атрибуты эти будут сохранены только в том случае, когда это допустимо целевой файловой системой: не следует ожидать, что атрибуты доступа и принадлежности будут сохранены при копировании на носитель с файловой системой FAT.

Для выполнения операции копирования файла он должен иметь атрибут чтения для пользователя, выполняющего копирование; кроме того, последний должен обладать правом на изменение каталога, в который производится копирование.

Кроме простого копирования файлов, существует команда для копирования с преобразованием — dd. Обобщенный ее формат весьма прост

```
$dd [options]
```

т. е. она просто копирует файл стандартного ввода в файл стандартного вывода, а опции описывают условия преобразования входного потока данных в выходной. Реально основными опциями являются `if=file1`, подменяющая стандартный ввод указанным файлом, и `of=file2`, проделывающая ту же операцию со стандартным выводом.

А далее — прочие условия преобразования весьма обширны. Большинство из них принимают численные значения в блоках:

- опции `ibs=n` и `obs=n` устанавливают размер блока для входного и выходного потоков, `bs=n` — для обоих сразу;
- опция `skip=n` указывает, сколько блоков нужно пропустить перед записью входного потока;
- опция `count=n` предписывает скопировать из входного потока лишь указанное количество блоков.

Имеется и опция `conv=value`, которая преобразует входной поток в соответствии с принятыми значениями, например, из формата ASCII в формат EBCDIC, рекомендуемый для использования в ОС на базе UNIX System V.

Сфера применения команды `dd` далеко выходит за рамки простого копирования файлов. Именно с ее помощью изготавливаются загрузочные дискеты, добавляются разделы подкачки, и многое другое. Один из примеров использования команды `dd` — для конвертации шрифтов Linux-консоли в формат для syscons FreeBSD (был приведен в *главе 12*).

Следующие две файловые операции — переименование и перемещение, — выполняются одной командой — `mv`. Она требует минимум двух аргументов — имени источника и целевого имени. Если оба они — имена файлов, происходит переименование первого файла во второй. Если последним аргументом выступает имя уже существующего каталога, то файл или каталог, указанные в качестве первого аргумента, перемещаются в каталог назначения. Причем если первый аргумент — файл, между первым и последним аргументами может быть сколько угодно аргументов — имен файлов (но не каталогов).

Как и при копировании, при перемещении и переименовании одноименные файлы, ранее существовавшие в целевом каталоге, затираются, замещаясь файлами-источниками без предупреждения. Чтобы этого не случилось, используется опция `-i`, требующая запрос на подтверждение действия. Напротив, опция `-f` в принудительном порядке перезаписывает существующий файл.

Операции копирования и перемещения/переименования выглядят сходными, однако по сути своей глубоко различны. Начать с того, что команда `mv` не совершает никаких действий с перемещаемыми или переименовываемыми файлами — она модифицирует каталоги, к которым приписаны имена

этих файлов. Это имеет два важных следствия. Во-первых, при перемещении/переименовании файлы сохраняют первозданными атрибуты доступа, принадлежности и, частично, времени. Во-вторых, для выполнения этих действий можно не иметь никаких вообще прав доступа к файлам — достаточно иметь право на изменение каталогов, в которых они переименовываются или перемещаются: ведь из *главы 9* мы помним, что имя файла фигурирует только в составе каталога и нигде более.

Аналогичный смысл имеет и удаление файлов, выполняемое командой

```
$rm filename
```

в которой аргументов, означающих имена подлежащих удалению файлов, может быть произвольное количество. Как и при перемещении, при этом не затрагиваются ни метаданные, ни данные файлов, а только удаляются их имена из родительских каталогов. И потому для удаления файлов опять же не обязательно иметь какие-либо права в их отношении — достаточно прав на изменение содержащих их каталогов.

Командой `rm` файлы-аргументы будут удалены в общем случае без предупреждения. Подобно командам `cp` и `mv`, для команды `rm` предусмотрены опции `-i` (запрос на подтверждение) и `-f` (принудительное удаление вне зависимости от настроек оболочки).

Интересный момент — удаление случайно созданных файлов с именами, "неправильными" с точки зрения системы или командной оболочки. Примером этого могут быть имена, начинающиеся с символа дефиса. Если попробовать сделать это обычным образом

```
$rm -file
```

в ответ последует сообщение об ошибке типа

```
rm: illegal option -- l
```

т. е. имя файла будет воспринято как опция. Для предотвращения этого такое "неправильное" имя следует предварить символом двойного дефиса и пробелом, означающими конец списка опций:

```
$rm -- -file
```

В принципе, команда `rm` ориентирована на удаление обычных и прочих файлов, но не каталогов. Однако с опцией `-d` она в состоянии справиться и с этой задачей — в случае, если удаляемый каталог пуст. Наконец, опция `-R` (или `-r`) производит рекурсивное удаление каталогов со всеми их файлами и вложенными подкаталогами.

Это делает использование опции `-R` весьма опасным: возможно, набивший оскомину пример

```
$rm -R /
```

когда при наличии прав суперпользователя уничтожается вся файловая система, и утрирован, но в локальном масштабе такая операция более чем реальна.

Специально для удаления каталогов предназначена команда

```
$rmdir
```

которая способна удалить только пустой каталог. Кроме того, с опцией `-p` она может сделать это и в отношении каталогов родительских — но также только в том случае, если они не содержат файлов.

Следующий аспект манипулирования файлами — изменение их атрибутов. Об этом достаточно говорилось в *главе 9*, поэтому лишь вкратце напомню, что атрибуты принадлежности изменяются командами `chown` (изменение владельца файла) и `chgroup` (изменение группы владельцев). Впрочем, и то, и другое — скорее прерогатива администратора, а не пользователя.

К изменению же атрибутов доступа пользователю приходится прибегать довольно часто. Повторю, что делается это командой `chmod` с указанием принадлежности и прав в отношении файлов-аргументов. Например, команда

```
$chmod a+x scriptname
```

волшебным образом превратит текстовый файл с набором команд в исполняемый сценарий.

Для изменения временных атрибутов используется команда `touch` — та самая с помощью которой можно создать пустой файл. Данная без опций с указанием существующего файла, она придаст ему атрибуты текущего момента времени. А с указанием опции `-d` [значение] — придаст файлу-аргументу любые временные метки (в том числе и из далекого будущего).

Архивация и компрессия

Архивация и компрессия — это уже не только манипулирование файлами, но и изменение их контента. Тем не менее, рассмотрим их в этом разделе — ведь с позиций пользователя их смысл близок копированию файлов. И, собственно, целям резервного копирования и архивация, и компрессия призваны служить.

Для пользователя DOS/Windows, привыкшего к программам типа Zip/WinZip, архивация и компрессия неразрывны, как лошади в упряжке. Однако это — разные действия. Архивация — это сборка группы файлов или каталогов в единый файл, содержащий не только данные файлов-источников, но и информацию о них — имена файлов и каталогов, к которым они приписаны, атрибуты принадлежности, доступа и времени, что позволяет восстановить как данные, так и их структуру из архива в первозданном виде. Компрессия же предназначена исключительно для уменьшения объема, занимаемого файлами на диске (или ином носителе).

Для архивации и компрессии в FreeBSD предназначены самостоятельные команды. Хотя архивацию и компрессию можно объединить в одной конструкции или представить так, будто они выполняются как бы в едином процессе.

Традиционные средства архивации UNIX-систем — команды `cpio` и `tar`. Суть первой, как можно понять из названия, — копирование файлов в файл архива и из файла архива. Используется она в трех режимах.

Первый режим, `copy-out`, определяемый опцией `-o` (или `--create`), предусматривает считывание списка файлов (`name list`) со стандартного ввода и объединяет их в архив, который может быть направлен в архивный файл или на устройство для записи резервных копий. Список файлов для архивирования может представлять собой вывод какой-либо иной команды. Так, в примере

```
$find./ -name \* | cpio -o > arch.cpio
```

файлы текущего каталога, выведенные командой `find`, при посредстве команды `cpio` будут направлены в архивный файл `arch.cpio`.

Второй режим (`copy-in`, опции `-i` или `--extract`) осуществляет обратную процедуру: развертывание ранее созданного архива в текущем каталоге:

```
$cpio -i < arch.cpio
```

Здесь нужно заметить, что если разворачиваемый архив включает подкаталоги, автоматически они созданы не будут, и последует сообщение об ошибке. Для создания промежуточных каталогов команда `cpio` должна использоваться с опцией — `d` (`--make-directories`).

В третьем режиме (`copy-pass`, опции `-p` или `--pass-through`) команда `cpio` выполняет копирование файлов из одного дерева каталогов в другой, комбинируя режимы `copy-out` и `copy-in`, но без образования промежуточного архива. Список файлов для копирования (`name list`) считывается со стандартного ввода, а каталог назначения указывается в качестве аргумента:

```
$cpio -p dir2 < name_list
```

Команда `cpio` имеет множество опций, позволяющих создавать, в частности, архивы в различных форматах (для межплатформенной переносимости). Однако я на них останавливаться не буду, отсылая заинтересованных к соответствующей man-странице: она не кажется мне удобной в применении. И упомянута здесь, во-первых, для полноты картины, а во-вторых — как одно из средств использования Linux-программ в FreeBSD, о чем будет сказано в *главе 23*).

Основным же средством архивирования в FreeBSD, как и во всех иных UNIX-системах, является команда `tar`. Обобщенный формат ее —

```
$tar [options] archiv_name [arguments]
```

где `archiv_name` — обязательный аргумент, указывающий на имя архивного файла, с которым производятся действия, определяемые главными опциями. Формы указания опций для команды `tar` очень разнообразны. Исторически первой была краткая форма без предваряющего дефиса, что поддерживается и поныне. Однако в текущих версиях команды в целях единообразия утверждена краткая форма с предваряющим дефисом или дублирующая ее полная форма, предваряемая двумя дефисами. Некоторые опции (например `--help` — получение справки об использовании команды) предусмотрены только в полной форме.

Главные опции и указывают на то, какие действия следует выполнить над архивом в целом:

- создание архива (опция `c`, `-c` или `--create`);
- просмотр содержимого существующего архива (опция `t`, `-t` или `--list`);
- распаковка архива (опция `x`, `-x`, `--extract` или `--get`).

Легко понять, что при работе с архивом как целым одна из этих главных (т. н. функциональных) опций обязательна. При манипулировании же фрагментами архива они могут подменяться другими функциональными опциями, как то:

- `r` (`-r` или `--append`) — добавление новых файлов в конец архива;
- `u` (`-u` или `--update`) — обновление архива с добавлением не только новых, но и модифицированных (с меньшим значением атрибута `mtime`) файлов;
- `-A` (`--catenate` или `--concatenate`) — присоединение одного архива к другому;
- `--delete` — удаление именованных файлов из архива;
- `--compare` — сравнение архива с его источниками в файловой системе.

Прочие (очень многочисленные) опции можно отнести в разряд дополнительных — они определяют условия выполнения основных функций команды. Однако одна из таких дополнительных опций — `f` (`-f` или `--file`), значение которой — имя файла (в том числе файла устройства, и не обязательно на локальной машине), также является практически обязательной. Дело в том, что команда `tar` (от tape archiv) изначально создавалась для прямого резервного копирования на стриммерную ленту, и именно это устройство подразумевается в качестве целевого по умолчанию. Так что, если это не так (а в нынешних условиях — почти наверняка не так), имя архивного файла в качестве значения опции `f` следует указывать явно. Причем некоторые реализации команды `tar` требуют, чтобы в списке опций она стояла последней (к версии FreeBSD это не относится).

Проиллюстрируем сказанное несколькими примерами. Так, архив из нескольких файлов текущего каталога создается следующим образом:

```
$tar cf arch_name.tar file1... file#
```

Если задать дополнительную опцию v, ход процесса будет отображаться на экране — это целесообразно, и в дальнейших примерах эта опция будет использоваться постоянно.

С помощью команды tar можно заархивировать и целый каталог, включая его подкаталоги любого уровня вложенности, причем — двояким образом. Так, если дать команду

```
$tar cvf arch_name.tar *
```

файлы из текущего каталога (включая подкаталоги) будут собраны в единый архив, но без указания имени каталога родительского. А командой

```
$tar cvf arch_name.tar dir
```

каталог dir будет упакован с полным сохранением его структуры.

С помощью команды

```
$tar xvf arch_name.tar
```

будет выполнена обратная процедура — распаковка заархивированных файлов в текущий каталог. Если при архивировании в качестве аргумента было указано имя каталога, а не набора файлов (пусть даже в виде шаблона) — этот каталог будет восстановлен в виде корневого для всех разархивируемых файлов.

При извлечении файлов из архива никто не обязывает нас распаковывать весь архив — при необходимости это можно сделать для одного нужного файла, следует только указать его имя в качестве аргумента:

```
tar xvf arch_name.tar filename
```

Правда, если искомый файл находился до архивации во вложенном подкаталоге, потребуется указать и путь к нему — от корневого для архива каталога, который будет различным для двух указанных схем архивации. Ну, а для просмотра того, каким образом был собран наш архив, следует воспользоваться командой

```
$tar tf arch_name.tar
```

Если архив собирался по первой схеме (с именами файлов в качестве аргументов), вывод ее будет примерно следующим:

```
dir2/
dir2/file1
example
```

```
new
newfile
tee.png
```

При втором способе архивации мы увидим на выводе нечто вроде

```
dir1/
dir1/example
dir1/new
dir1/newfile
dir1/tee.png
dir1/dir2/
dir1/dir2/file1
```

В данном примере опция `v` была опущена. Включение ее приведет к тому, что список файлов будет выведен в длинном формате, подобном выводу команды `ls -l`:

```
drwxr-xr-x alv/alv          0 10 май 11:03 2002 dir2/
-rw-r--r-- alv/alv          0 10 май 11:03 2002 dir2/file1
...
```

Команда `tar` имеет еще множество дополнительных опций, призванных предотвращать перезапись существующих файлов, осуществлять верификацию архивов, учитывать при архивации разного рода временные атрибуты, вызывать для исполнения другие программы. К некоторым опциям я еще вернусь после рассмотрения команд компрессии, другие же предлагается изучить самостоятельно, воспользовавшись страницей экранной документации.

Команд для компрессии файлов в FreeBSD несколько, но реальный интерес ныне представляют две парные утилиты — gzip/gunzip и bzip2/bunzip2. Первый член каждой пары, как легко догадаться из названия, отвечает преимущественно за компрессию, второй — за декомпрессию файлов (хотя посредством должных опций они легко меняются ролями).

Команда `gzip` — это традиционный компрессор UNIX-систем, сменивший в сей роли более старую утилиту compress. Простейший способ ее использования —

```
$gzip filename
```

где в качестве аргумента будет выступать имя файла. При этом (внимание!) исходный несжатый файл подменяется своей сжатой копией, которой автоматически присваивается расширение gz.

В качестве аргументов может выступать и произвольное количество имен файлов — каждый из них будет заменен сжатым файлом с расширением gz. Более того, посредством опции `-r` может быть выполнено рекурсивное сжатие файлов во всех вложенных подкаталогах. Подчеркну, однако, что ника-

кой архивации команда gzip не производит, обрабатывая только единичный файл. Фактически форма

```
$gzip file1 file2... file#
```

просто эквивалент последовательности команд

```
$gzip file1
$gzip file2
...
$gzip file#
```

Правда, объединение компрессированных файлов возможно методом конкатенации (с помощью команды cat) или посредством архивирования командой tar — и о том, и о другом будет сказано в свое время.

Команда gzip имеет и другие опции, указываемые в краткой (однобуквенной) или полной нотации. В отличие от tar, знак дефиса (или, соответственно, двойного дефиса) обязателен в обоих случаях. Так, опциями -1... -9 можно задать степень сжатия и, соответственно, время процедуры: -1 соответствует минимальному, но быстрому сжатию, -9 — максимальному, но медленному. По умолчанию в команде gzip используется опция -6, обеспечивающая компромисс между скоростью и компрессией.

Благодаря опции -d (--decompress) команда gzip может выполнить развертывание сжатого файла, заменяя его оригиналом без расширения gz. Хотя в принципе для этого предназначена команда gunzip:

```
$gunzip file.gz
```

Использование этой команды настолько прозрачно, что я задерживаться на ней не буду.

В последнее время широкое распространение получил компрессор bzip2, обеспечивающий большую (на 10—15%) степень сжатия, хотя и менее быстродействующий. Использование его практически идентично gzip, с деталями его можно ознакомиться с помощью страницы экранной документации man bzip2. Итоговый компрессированный файл получает имя вида bz2 и может быть распакован командой bunzip2 (или командой bzip2 -d). Следует только помнить, что форматы gz и bz2 не совместимы между собой. Соответственно, первый не может быть распакован программой bunzip2, и наоборот.

Поскольку команды tar и gz обеспечивают каждая свою сторону обработки файлов, возникает резонное желание использовать их совместно. Самый простой способ сделать это — воспользоваться командой tar с опцией z:

```
tar cvzf dir.tar.gz dir/
```

Обратите внимание, что расширение gz в этом случае нужно указывать в явном виде — автоматически оно к имени архива не присоединяется и компрессированный архив будет иметь вид dir.tar. Поскольку в FreeBSD расширения имен файлов не играют той сакральной роли, что в MS-DOS, это не помешает распаковке такого файла командой

```
tar xvzf dir.tar
```

Опция z сама по себе никакой компрессии не выполняет — она просто вызывает компрессор gzip для сжатия каждого из архивируемых файлов. Аналогичный смысл имеет и опция j — только ею для этой цели привлекается команда bzip2.

При использовании команды tar с опцией z (или j) исходные файлы остаются в неприкосновенности. Следует, однако, помнить, что архив сжатых файлов не может быть обновлен командой tar с параметрами r или u.

Есть и другой способ совместной архивации и компрессии — просто последовательность команд

```
$tar cf dir.tar *
$gzip dir.tar
```

В результате образуется сжатый архив — внешне такой же файл dir.tar.gz. Хотя в принципе архив сжатых файлов и сжатый архивный файл — это разные вещи (можно заметить, что они даже различаются по объему, хотя и всего на несколько байтов), сжатый архив также может быть благополучно развернут командой tar с опцией z. И столь же очевидно, что он не может быть ни пополнен, ни обновлен средствами архиватора tar.

Компрессированные архивы, созданные сочетанием программ tar и gzip — общепринятый в UNIX-системах метод распространения файлов. Однако иногда для совместимости с ОС, не допускающими двух точек в имени файла (знаете такую ОС?), компрессированным tar-архивам на присваивается расширение *.tgz.

Команда *find* — универсальное средство работы с файлами

Настало время обратиться к команде find. Столь высокая честь выделения в специальный подраздел выпадает ей потому, что посредством этой команды, кроме поиска, можно решить если не все, то большинство проблем, возникающих при управлении файлами. Строго говоря, вопреки своему названию, команда find выполняет не поиск файлов, а рекурсивный обход дерева каталогов, начиная с заданного, отбирает из них файлы в соответствии с некоторыми критериями и выполняет над отбракованным файловым хозяй-

ством некоторые действия. Именно эту ее особенность подчеркивает резюме команды `find`, получаемое посредством

```
$whatis find
find(1)                    - walk a file hierarchy
```

что применительно случаю можно перевести как "прогулка по файловой структуре".

В обобщенном виде формат команды можно представить следующим образом:

```
$find путь_поиска [критерий_поиска] [значение критерия] [опция_действия]
```

Путь поиска — это каталог, с которого следует искать файлы, например, корневой (/) или домашний каталог пользователя ($HOME). Критерий поиска — признак, по которому следует искать файл (файлы). Ну, а опция действия указывает на то, что надлежит сделать с найденным файлом или файлами (а сделать с ними, надо заметить, можно немало). Критерий поиска и опция действия предваряются знаком дефиса, значение критерия отделяется от его названия пробелом.

Начнем по порядку. Основные опции команды `find` позволяют выполнить поиск по следующим признакам (символ дефиса перед опциями ниже опущен, но не следует забывать его ставить):

- `name` — поиск по имени файла или по маске имени; в последнем случае метасимволы маски должны обязательно экранироваться (например, `-name \*.tar.gz`) или заключаться в кавычки (одинарные или двойные, в зависимости от правил, принятых для данной командной оболочки; этот критерий чувствителен к регистру, но близкий по смыслу критерий `iname` позволяет производить поиск по имени без различения строчных и заглавных букв;
- `type` — поиск по типу файла; этот критерий принимает следующие значения: `f` (регулярный файл), `d` (каталог), `s` (символическая связь), `b` (файл блочного устройства), `c` (файл символьного устройства); есть и опция `fstype`, предписывающая выполнять поиск только в файловой системе указанного типа;
- `user` и `group` — поиск по имени или идентификатору владельца или группы, выступающим в качестве значения критерия; существуют также критерии `nouser` и `nogroup` — они отыскивают файлы, владельцев не имеющие (то есть тех, учетные записи для которых отсутствуют в файлах /etc/passwd и /etc/group); эти критерии в значениях, разумеется, не нуждаются;
- `size` — поиск по размеру, задаваемому в виде числа (с последующим символом `c`) в блоках или в байтах; возможны значения `n` (равно `n` блоков), `+n` (более `n` блоков), `-n` (менее `n` блоков);
- `perm` — поиск файлов по значениям их атрибутов доступа, задаваемых в символьной форме;

- `atime`, `ctime`, `mtime` — поиск файлов с указанными временными атрибутами; значения временных атрибутов указываются в сутках (точнее, в периодах, кратных 24 часам); возможны формы значений этих атрибутов: `n` (равно указанному значению `n*24` часа), `+n` (ранее `n*24` часа), `-n` (позднее `n*24` часа);
- `newer` — поиск файлов, созданных после файла, указанного в качестве значения критерия (т. е. имеющего меньшее значение `mtime`);
- `maxdepth` и `mindepth` — позволяют конкретизировать глубину поиска во вложенных подкаталогах — меньшую или равную численному значению для первого критерия и большую или равную — для второго;
- `depth` — производит отбор в обратном порядке, т. е. не от каталога, указанного в качестве аргумента, а с наиболее глубоко вложенных подкаталогов; смысл этого действия — получить доступ к файлам в каталоге, для которого пользователь не имеет права чтения и исполнения;
- `prune` — позволяет указать подкаталоги внутри пути поиска, в которых отбора файлов производить не следует.

Критерии отбора файлов могут группироваться практически любым образом. Так, в форме

```
$find ~/ -name \*.tar.gz newer filename
```

команда `find` выберет в домашнем каталоге пользователя все компрессированные архивы, созданные после файла с именем `filename`. По умолчанию между критериями отбора предполагается наличие логического оператора "И". Если требуется использование оператора "ИЛИ", он должен быть явно определен в виде опции `-o` между критериями. Так, команда

```
$find ~/ -mtime -2 -o newer filename
```

призвана отобрать файлы, созданные менее двух суток назад или позднее, чем файл `filename`.

Особенность реализации команды `find` в FreeBSD — то, что она по умолчанию выводит список отобранных в соответствии с заданными критериями файлов на экран, не требуя дополнительных опций действия. Однако в других UNIX-системах (в том числе и в Linux) указание таких опций — обязательно. Так что рассмотрим их по порядку.

Для выведения списка отобранных файлов на экран в общем случае предназначена опция `-print`. Вывод этот имеет примерно следующий вид:

```
find. -name f\* -print
./file1
./file2
./dir1/file3
```

Сходный смысл имеет и опция `-ls`, однако она выводит более полные сведения о найденных файлах, аналогично команде `ls` с опциями `-dgils`:

```
find. -name f\* -ls
673098 0 -rw-r--r-- 1 alv alv 0  9 май 17:15./file1
673101 0 -rw-r--r-- 1 alv alv 0  9 май 17:16./file2
694405 0 -rw-r--r-- 1 alv alv 0  9 май 17:16./dir1/file3
```

Далее, опция `-delete` уничтожит все файлы, отобранные по указанным критериям. Так, командой

```
$find ~ -atime +100 -delete
```

будут автоматически стерты все файлы, к которым не было обращения за последние 100 дней (из молчаливого предположения, что раз к ним три месяца не обращались — значит, они и вообще не нужны). Истреблению подвергнутся файлы в подкаталогах любого уровня вложенности — но не включающие их подкаталоги (если, конечно, последние сами не подпадают под критерии отбора).

И, наконец, опция `-exec` — именно ею обусловлено величие утилиты `find`. В качестве ее значения можно указать любую команду с необходимыми опциями — и она будет выполнена над отобранными файлами, которые будут рассматриваться в качестве ее аргументов. Проиллюстрируем это на примере.

Использовать опцию `-delete`, как мы это только что сделали — не самое здоровое решение — файлы при этом удаляются без запроса, и можно случайно удалить что-нибудь нужное. И потому достигнем той же цели следующим образом:

```
$find ~/ -atime +100 -exec rm -i {} \;
```

В этом случае на удаление каждого отобранного файла будет запрашиваться подтверждение.

Обращаю внимание на последовательность символов `{} \;` (с пробелом между закрывающей фигурной скобкой и обратным слэшем) в конце строки. Пара фигурных скобок `{}` символизирует, что свои аргументы исполняемая команда (в примере — `rm`) получает от результатов отбора команды `find`, точка с запятой означает завершение строки (без нее на нажатие <Enter> последует вторичное приглашение), а обратный слэш экранирует ее специальное значение.

Приведенный пример, хотя и вполне жизненный, достаточно элементарен. Рассмотрим более сложный случай — собирание в один каталог всех скриншотов в формате PNG, разбросанных по древу домашнего каталога:

```
$find ~/ -name \*.png -exec cp {} imagesdir \;
```

В результате все png-файлы будут изысканы и скопированы (или — перемещены, если воспользоваться командой `mv` вместо `cp`) в одно место.

И напоследок — вариант решения задачи, которая казалась мне сначала трудно разрешимой: рекурсивное присвоение необходимых атрибутов доступа в разветвленном дереве каталогов — различных для регулярных файлов и каталогов.

Зачем это нужно? Поясню на примере. Давеча, обзаведясь огромным (40 Гбайт) винчестером, я решил собрать на него все нужные мне данные, рассеянные по дискам CD-R/RW (суммарным объемом с полкубометра) и нескольким сменным винчестерам, одни из которых были отформатированы в FAT16, другие — в FAT32, третьи — вообще в ext2fs. Сгрузив все это богачество в один каталог на новом диске, я создал в нем весьма неприглядную картину.

Ну, во-первых, все файлы, скопированные с CD и FAT-дисков, получили биты исполняемости, хотя все это были файлы данных. Казалось бы, мелочь, но это не позволяет, например, просмотреть html-файл в Midnight Commander простым нажатием клавиши <Enter>. Во-вторых, для некоторых каталогов, напротив, исполнение не было предусмотрено ни для кого — т. е. я же сам перейти в них не мог. В-третьих, каталоги (и файлы) с CD часто не имели атрибута изменения — а они нужны мне были для работы (в т. ч. и редактирования). Ситуация явно требовала исправления, однако проделать вручную такую работу над данными более чем в 20 Гбайт видилось немыслимым.

Да так оно, собственно, и было бы, если б не опция `-exec` утилиты find. Каковая позволила изменить права доступа требуемым образом. Итак, сначала отбираем все регулярные файлы и снимаем с них бит исполнения для всех:

```
$find ~/dir_data -type f -exec chmod a-x,u+w {} \;
```

Далее — поиск каталогов и обратная процедура над итоговой выборкой:

```
$find ~/dir_data -type d -exec chmod a+xr,u+w {} \;
```

И дело — в шляпе, все права доступа стали единообразными (и теми, что мне нужно). Именно после этого случая я, подобно митьковскому Максиму, проникся величием философии марксизма (пардон, утилиты find). А ведь это еще не предел ее возможностей — последний устанавливается только встающими задачами и собственной фантазией...

Так, с помощью команды `find` легко наладить периодическое архивирование результатов текущей работы. Для этого перво-наперво создаем командой `tar` полный архив результатов своей деятельности:

```
tar cvf alldata.tar ~/*
```

А затем в меру своей испорченности (или, напротив, аккуратности) время от времени запускаем команду

```
$find ~/ -newer alldata.tar -exec tar uf alldata.tar {} \;
```

Еще один практически полезный вариант использования команды `find` в мирных целях — периодическое добавление отдельно написанных фрагментов к итоговому труду жизни (например, мемуарам эникейщика). Впрочем, чтобы сделать это, необходимо сначала ознакомиться с командами обработки файлов.

Команды для обработки файлов

В предыдущем разделе речь шла о командах, которые манипулируют файлами как целыми, не затрагивая их содержания (и, в общем случае, от такового не зависящими). Правда, манипулирование файлами (копирование, перемещение и т. д.) также подразумевает изменение содержания файлов одного-единственного типа (а именно — каталогов), однако собственно внутренняя сущность обычных файлов при этом не изменяется.

Ныне же речь пойдет о командах, позволяющих в той или иной мере учитывать контент файлов и манипулировать ими. Разумеется, манипулирование контентом возможно только для регулярных файлов. При этом многие их разновидности (бинарные файлы, файлы графических форматов и текстовых редакторов) требуют для изменения своего содержания специальных средств — а именно, прикладных программ, в которых они создавались. Так что на самом деле команды модификации контента действенны только для файлов текстовых.

Однако круг объектов таких команд не столь уж узок, как может показаться. Ведь именно в виде обычных текстовых файлов в FreeBSD хранится масса общесистемной информации, исполняемых сценариев, баз данных атрибутов самых разных объектов. Далее — собственно нарративные тексты любого содержания: ведь чисто текстовый формат для них куда роднее, чем doc сотоварищи. Ну и никем не возбраняется использовать такие команды в отношении текстов с разметкой — HTML ли, XML, TeX или еще чего. Так что поле приложения рассматриваемых команд — достаточно обширно.

Просмотр файлов

Однако прежде чем как-то манипулировать контентом файлов, желательно этот самый контент некоторым образом просмотреть. И тут можно вспомнить о команде `cat`, посредством которой мы некогда создавали файлы. Данная с именем файла в качестве аргумента, она выведет его содержимое на экран. Можно использовать и конструкцию перенаправления:

```
$cat < filename
```

Несмотря на то, что, в принципе, это разные вещи, результат будет тот же — содержимое файла на экране.

Недостаток команды `cat` как средства просмотра — невозможность перемещения по телу файла: выведя содержимое, она завершает свою работу. Конечно, "пролистывание" выведенного возможно, но — средствами системной консоли, а не самой команды.

Поэтому обычно для просмотра содержимого файлов используются специальные программы постраничного просмотра — т. н. pager's, очередной пример того, что передача этого термина исконно русским (вернее, скорее ново-русским) словом "пейджер" (а мне попадалось и такое) может вызвать превратное представление о сути дела.

В UNIX-системах имеются две основные программы постраничного просмотра — more и less. Первая из них допускает только однонаправленный (вперед) просмотр и имеет слабые интерактивные возможности. Почему ныне и представляет лишь исторический интерес, так что о ней я говорить не буду. Тем более, что в современных версиях FreeBSD она как таковая отсутствует: файл с именем /usr/bin/more на самом деле представляет собой жесткую ссылку на ту же самую программу, что и команда `less`, в чем легко убедиться командой

```
$ls -i /usr/bin/more /usr/bin/less
334245 /usr/bin/less* 334245 /usr/bin/more*
```

вывод которой показывает одинаковый идентификатор `inode` для имен обоих файлов. Хотя эта программа проявляет несколько различные свойства, в зависимости от того, какой командой вызвана, функции ее от этого не меняются. Так что дальше я буду говорить только о команде `less` (с оговорками относительно `more` в необходимых случаях).

Самый простой способ вызова команды

```
$less filename
```

после чего на экран выводится содержимое файла, указанного в качестве аргумента, по которому можно перемещаться в обоих направлениях, как построчно, так и постранично. В нижней строке экрана можно видеть символ двоеточия — приглашения для ввода команд различного назначения. В частности, нажатие клавиши <h> выводит справку по использованию `less`, а нажатие клавиши <q> — выход из программы просмотра (или из просмотра справочной системы, если она была перед этим вызвана). Если команда была вызвана как `more`, вместо символа двоеточия в нижней строке будет выведено имя файла с указанием процента просмотра:

```
command.txt 3%
```

что, однако, не воспрещает и здесь вводить команды.

Большинство встроенных команд `less` предназначено для навигации по телу файла. Осуществляется она несколькими способами:

- с помощью стандартных клавиш управления курсором: <PageDown> или <Spacebar> (вперед на один экран), <PageUp> (назад на один экран),

<Down> или <Enter> (вперед на одну строку), <Up> (назад на одну строку), <Right> (на пол-экрана вправо), <Left> (на пол-экрана влево);

- с помощью предопределенных клавишных комбинаций, сходных с управляющими клавиатурными последовательностями командных оболочек и таких текстовых редакторов, как emacs и joe (хотя и не всегда с ними совпадающими): <Ctrl>+<V> (на один экран вперед), <Escape>–<V> (на один экран назад), <Ctrl>+<N> (на одну строку вперед), <Ctrl>+<P> (на одну строку назад), <Escape>–<(> и <Escape>–<)> (на пол-экрана вправо и влево соответственно);
- с помощью фиксированных символьных клавиш, иногда подобных таковым командного режима редактора vi: <z> и <w> (вперед и назад на один экран), <e> и <y> (вперед и назад на одну строку, можно использовать также привычные по vi клавиши <j> и <k> соответственно), <d> и <u> (вперед и назад на пол-экрана).

Последний способ интересен тем, что допускает численные аргументы перед символьной командой: так, нажатие <3>–<e> приведет к перемещению на три строки вперед, а <2>–<w> — на два экрана назад.

Помимо "плавной", так сказать, навигации, можно перемещаться по файлу и скачками (jumping): нажатие клавиши с символом <g> (или последовательности <Escape>–<<>) позволяет переместиться к первой строке файла, клавиши <G> (регистр важен! дублирующий вариант — <Escape>–<>>) — к последней его строке, клавиши <p> — к началу файла (честно говоря, я не очень понял, чем это отличается от первой его строки).

Кроме навигации, имеется и возможность двустороннего поиска — в направлении как конца, так и начала файла. Для поиска вперед требуется ввести символ прямого слэша (</>) и за ним — искомое сочетание символов. Поиск в обратном направлении предваряется символом вопроса (<?>). В обоих случаях в шаблоне поиска можно использовать стандартные регулярные выражения `*`, `?`, `[список_символов]` или `[диапазон_символов]`. Нажатие клавиши <n> (в нижнем регистре) приводит к повторному поиску в заданном ранее направлении, клавиши <N> (в верхнем регистре) — к поиску в направлении противоположном.

Управляющие комбинации команды `less` могут быть переопределены с помощью команды `lesskey`. Формат ее

```
$ lesskey -o output input
```

В качестве входных данных выступает простой текстовый файл (по умолчанию в качестве такового выступает ~/.lesskey, однако его следует создать самостоятельно), описывающий клавишные последовательности в следующем, например, виде:

```
#command
\r        forw-line
```

```
\n        forw-line
...
k         back-line
...
```

Выходные данные — создаваемый из текстового двоичный файл, который собственно и используется командой `less`. Стандартное для него имя — ~/.less.

Команда `less` допускает одновременный просмотр нескольких файлов. Для этого ее следует вызвать в форме

```
$less file1 file2... file#
```

после чего между открытыми файлами можно переключаться посредством `:-n` (к следующему файлу), `:p` (к предыдущему файлу), `:x` (к первому файлу). Путем нажатия `:d` текущий файл исключается из списка просмотра. Символ двоеточия во всех этих случаях вводится с клавиатуры в дополнение к приглашению на ввод команд.

Команда `less` имеет великое множество опций — описание их в экранной документации занимает более дюжины страниц, поэтому задерживаться на них я не буду. Следует заметить только, что опции эти могут использоваться не только в командной строке при запуске `less`, но и интерактивно — после символа дефиса в приглашении ввода. Так, указав там `-m`, можно включить т. н. промежуточный формат приглашения (с отображением процентов просмотренного объема файла), а с помощью `-M` — длинный формат, при котором в приглашении дополнительно указываются имя файла, его положение в списке загруженных файлов, просматриваемые ныне строки:

```
command.html (file 2 of 10) lines 1-29/1364 2%
```

Значение команд постраничного просмотра файлов еще и в том, что именно с их помощью осуществляется доступ к экранной документации (man-страницам). Команда

```
$man cmd_name
```

на самом деле вызывает определенный по умолчанию pager для просмотра соответствующего файла /usr/share/man/man#/cmd_name.gz. Какой именно — определяется переменной `PAGER` в пользовательских настройках, о чем будет сказано в главе о командных оболочках.

Кроме команд постраничного просмотра, существуют команды для просмотра фрагментарного. Это — `head` и `tail`, выводящие на экран некоторую фиксированную порцию файла, указанного в качестве их аргумента, с начала или с конца соответственно. По умолчанию эта порция для обеих команд составляет десять строк (включая пустые). Однако ее можно переопределить

произвольным образом, указав опции -n [число_линий] или -c [количество_байт]. Например, команда

```
$head -n 3 filename
```

выведет три первые строки файла filename, а команда

```
$tail -c 100 filename
```

его последние 100 байт. При определении выводимого фрагмента в строках название опции (n) может быть опущено — достаточно числа после знака дефиса.

Существуют и средства просмотра компрессированных файлов. Для файлов, сжатых программой gzip, можно использовать команды zcat и zmore, для спрессованных командой bzip2 — команду bzcat. Использование их ничем не отличается от аналогов для несжатых файлов — в сущности, именно они и вызываются для обеспечения просмотра. В случае команды zmore, как нетрудно догадаться, на самом деле используется команда less (сама по себе она аналога для компрессированных файлов не имеет).

Сравнение, объединение и деление файлов

Следующая важная группа операций над контентом файлов — сравнение файлов по содержанию и различные формы объединения файлов и их фрагментов. Начнем со сравнения. Простейшая команда для этого — cmp в форме

```
$cmp file1 fil2
```

производит построчное сравнение файлов, указанных как первый и второй аргументы (а более их и не предусмотрено, все указанное после второго аргумента игнорируется). В случае идентичности сравниваемых файлов не происходит ничего, кроме возврата приглашения командой строки. Если же между файлами имеются различия, выводится номер первого различающегося символа и номер строки, в которой он обнаружен:

```
file1 file2 differ: char 27, line 4
```

Это означает, что различия между файлами начинаются с 27-го от начала файла символа (включая пробелы, символы конца строк и т. п.), который имеет место быть в строке 4. С помощью опций -l и -z можно заставить команду cmp вывести номера всех различающихся символов в десятичном или шестнадцатеричном формате соответственно.

Более информативный вывод обеспечивает команда diff. Она также осуществляет построчное сравнение двух файлов, но выводит список строк, в которых обнаружены отличия. Например, для двух файлов вида

```
$less file1
line 1
```

```
line 2
line 3
line 4
line 5
```

и

```
$less file2
line 1
line 2
line 3
line 3a
line 4
line 5
```

это будет выглядеть следующим образом:

```
$diff file1 file2
3a4
> line 3a
```

Если различия будут выявлены более чем в одной строке, для каждого расхождения будет выведен аналогичный блок. Смысл его — в том, какие строки первого файла должны быть преобразованы, и как именно, для того, чтобы файлы стали идентичными. Первая линия блока вывода содержит номер строки первого файла, подлежащей преобразованию, номер соответствующей строки второго файла и обозначенное буквенным символом преобразование, во второй линии приведена собственно строка — предмет преобразования. Символы преобразования — следующие:

- `a` (от append) указывает на строку, отсутствующую в первом файле, но присутствующую во втором;
- `c` (от change) фиксирует строки с одинаковым номером, но разным содержанием;
- `d` (от delete) определяет строки, уникальные для первого файла.

То есть в данном примере для преобразования `file1` в `file2` в него после строки 3 должна быть вставлена строка 4 из второго файла, что символизирует вторая линия блока — `> line 3a`, где `>` означает строку из первого сравниваемого файла. Если же аргументы команды `diff` дать в обратном порядке, вывод ее будет выглядеть следующим образом:

```
$diff file2 file1
4d3
< line 3a
```

показывающим, что для достижения идентичности из file2 должна быть удалена четвертая строка (< line 3a, где < означает строку из второго файла). Если же произвести сравнение file1 с file3, имеющим вид

```
$less file3
line 1
line 2
line 3a
line 4
line 5
```

вывод команды

```
$diff file1 file3
3c3
< line 3
---
> line 3a
```

будет означать необходимость замены третьей строки из file1 (символ <) на третью строку из file3 (символ >).

Такая форма вывода команды diff называется стандартной. С помощью опции -c можно задать т. н. контекстную форму вывода, при которой на экран направляются не только различающиеся строки, но и окружающие ее (т. е. контекст, в котором они заключены):

```
diff -c file1 file2
*** file1       Sun May 12 11:44:44 2002
--- file2       Mon May 13 15:17:27 2002
***************
*** 1,5 ****
--- 1,6 ----
  line 1
  line 2
  line 3
+ line 3a
  line 4
  line 5
```

Количество строк контента задается опцией -C:

```
diff -C 1 file1 file2
*** file1       Sun May 12 11:44:44 2002
--- file2       Mon May 13 15:17:27 2002
***************
```

```
*** 3,4 ****
--- 3,5 ----
  line 3
+ line 3a
  line 4
```

В этом примере значение опции -C (единица) предписывает вывод по одной строке контентного окружения вокруг различающейся строки. Сами же различающиеся строки помечаются следующим образом: знаком – (минус, или дефис) — строки, подлежащие удалению из первого файла, знаком + (как в примере) — строки, которые должны быт добавлены, знаком ! — просто различающиеся строки.

Кроме контекстного формата, используется еще и вывод в унифицированном формате, что предписывается опцией -U [значение], в качестве значения указывается число строк. В нем для обозначения изменяемых строк используются только символы + и -, а сам вывод чуть короче, чем при использовании контекстного формата.

С помощью многочисленных опций команды diff сравнение файлов может быть детализировано и конкретизировано. Так, опция -b предписывает игнорировать "пустые" символы пробелов и табуляции в конце строк, а опция -w — вообще "лишние" пробелы (и те, и другие обычно имеют случайное происхождение). При указании опции -B игнорируются пустые строки, т. е. не содержащие никаких иных символов, кроме перевода каретки; строки с символами табуляции или пробела как пустые не рассматриваются, для их игнорирования требуется опция -w. Благодаря опции -i при сравнении не принимаются во внимание различия регистров символов, а опция -I regexp определяет регулярные выражения, строки с которыми также игнорируются при сравнении.

В качестве аргументов команды diff (одного или обоих) могут выступать также каталоги. Если каталогом является только один из аргументов, для сравнения в нем отыскивается файл, одноименный второму аргументу. Если же оба аргумента суть каталоги, в них происходит сравнение всех одноименных файлов в алфавитном порядке (вернее, в порядке ASCII-кода первого символа имени, разумеется). Благодаря опции -r сравнение файлов может осуществляться и во вложенных подкаталогах.

Вывод команды diff может быть перенаправлен в файл. Такие файлы различия именуются diff-файлами или, применительно к исходным текстам программ, патчами (patches), о которых будет сказано несколько позже. Именно с помощью таких патчей обычно распространяются изменения к программам (дополнения, исправления ошибок и т. п.).

В принципе, команда diff и придумана была именно для сравнения файлов исходников, над которыми ведут работу несколько (в пределе — неограни-

ченное количество, как в случае с операционной системой Linux) человек. Однако невозбранно и ее использование в мирных целях — т. е. для сравнения просто повествовательных текстов. Единственное, что следует понимать при этом абсолютно ясно — то, что `diff` выполняет именно построчное сравнение. То есть: сравнение последовательностей символов, ограниченных символами конца строки с обеих сторон. И, соответственно, непрерывная абзацная строка в стиле emacs и vi — совсем не то, что строка, образуемая joe на границе экрана. Впрочем, это — вопрос, к которому еще не раз придется возвращаться.

Как уже было отмечено, команда `diff` осуществляет сравнение двух файлов (или — попарное сравнение файлов из двух каталогов). Однако, поскольку Бог, как известно, любит троицу, есть и команда `diff3`, позволяющая сравнить именно три файла, указываемые в качестве ее аргументов. По действию она не сильно отличается от двоичного аналога. И потому изучение ее особенностей предлагается в качестве самостоятельного упражнения приверженцам троичной идеологии.

Существуют и средства для сравнения сжатых файлов. Это — zcmp и zdiff. Подобно командам просмотра, ими просто вызываются соответствующие команды `cmp` и `zdiff`. И потому использование их не имеет никаких особенностей.

От вопроса сравнения файлов плавно перейдем к рассмотрению способов их объединения. Для этого существует немало команд, из которых по справедливости первой должна идти команда `cat`, поскольку именно сие есть ее титульная функция (`cat` — от `concatenation`, сиречь объединения). Ранее уже упоминалось, что она способна добавлять информацию со стандартного ввода в конец существующего файла. Однако дело этим не ограничивается. В форме

```
$cat file1 file2... file# > file_all
```

она создает новый файл, включающий в себя содержимое всех файлов-аргументов (и именно в том порядке, в каком они приведены в командной строке). Операция, казалось бы, не хитрая — однако представьте, сколько действий потребовалось бы в текстовом процессоре (например, в Word) для того, чтобы создать синтетический вариант из полутора десятков фрагментов, раскиданных по разным каталогам?

А вот команда `patch` выступает в качестве диалектической пары для команды `diff`, именно она вносит в файл те изменения, которые документируются последней. Выглядит эта команда примерно так:

```
$patch file1 diff_file
```

в ответ на что последует нечто вроде следующего вывода:

```
Hmm...  Looks like a normal diff to me...
Patching file file1 using Plan A...
```

```
Hunk #1 succeeded at 4.
done
```

В результате исходная версия file1 будет сохранена под именем file1.orig, а сам он преобразован в соответствии с описанием diff-файла. Возможна и форма

```
patch < diff_file
```

В этом случае команда `patch` попытается сама определить имя файла-оригинала, и, если это ей не удастся, даст запрос на его ввод. Обращаю внимание на символ перенаправления ввода, поскольку если его опустить, имя diff-файла будет воспринято как первый аргумент команды (т. е. имя файла-оригинала).

В качестве второго аргумента команды `patch` могут использоваться diff-файлы не только в стандартном, но и в контекстном или унифицированном формате. Это следует указать посредством опции `-c` или `-u` соответственно.

Сочетание команд `diff` и `patch` очень широко используется при внесении изменений в исходные тексты программы. Так, если мы обратимся к коллекции портов, то в большинстве каталогов портированных программ можно обнаружить подкаталог (например, /usr/ports/editors/joe-devel/files), содержимое которого как раз и есть те патчи, которые должны накладываться на исходные тексты программы для ее корректной сборки, установки и функционирования в FreeBSD. Сам же исходный текст при этом в неизменном, т. е. распространяемом разработчиком, виде получается с его сайта (или любого другого сервера, в том числе и каталога distfiles FTP-сервера проекта FreeBSD.

Не менее часто, чем в слиянии, возникает необходимость и в разделении файлов на части. Цели этой служит команда `split`. Формат ее:

```
$split [options] filename [prefix]
```

В результате исходный файл будет разбит на несколько отдельных файлов вида `prefixaa`, `prefixab` и т. д. Значение аргумента `prefix` по умолчанию — `x` (т. е. без его указания итоговые файлы получат имена `xaa`, `xab` и т. п.).

Опции команды `split` задают размер выходных файлов — в байтах (опция `-b`) или количестве строк (опция `-l`). Первой опцией в качестве единицы, кроме байтов, могут быть заданы также килобайты или мегабайты — добавлением после численного значения обозначения `k` или `m` соответственно.

Команда `split` может использоваться для разбиения файлового архива на фрагменты, соответствующие размеру резервных носителей. Так, в форме

```
split -b 1474560 arch_name
```

она обеспечит разбиение архива на части, каждая из которых может быть записана на стандартную трехдюймовую дискету. А посредством

```
split -b 650m arch_name
```

архив можно подготовить к записи на носители CD-R/RW. Напомню, что именно командой `split` создаются файлы дистрибутива FreeBSD. Легко догадаться, что обратное слияние таких фрагментированных файлов можно выполнить командой `cat`.

Однако этим возможности команды `split` не исчерпываются. С помощью опции `-p` файл может быть разделен на фрагменты, ограниченные строками, содержащими текст, приведенный в качестве значения шаблона. Так, командой

```
$split -p Глава filename chapter-
```

текст произвольной длинны будет легко разбит на файлы, соответствующие отдельным главам, которым будут присвоены имена `chapter-aa` и далее. В качестве значения опции `-p` могут использоваться тэги HTML или символы разметки TeX, а также регулярные выражения.

Поиск в файлах

В одном из предыдущих разделов говорилось о поиске файлов посредством команды `find`. Ныне же речь пойдет о поиске внутри файлового контента — т. е. поиске текстовых фрагментов. Для этого в UNIX-системах используется семейство утилит grep — собственно grep, egrep и fgrep, несколько различающихся функционально. Впрочем, в FreeBSD все это суть разные имена (жесткие ссылки) одной и той же программы (проверяется это, как всегда, командой `ls` с опцией `-i`), именуемой grep, но включающей ряд функций, свойственных ее расширенному аналогу, egrep. Соответственно, поиск текстовых фрагментов в файлах может быть вызван любой из этих команд, хотя в каждом случае функциональность их будет несколько различаться.

Однако начнем по порядку. Самой простой формой команды `grep` является следующая:

```
$grep pattern files
```

где `pattern` — искомая последовательность символов, а `files` — файлы, среди которых должен производиться поиск (или — просто одиночный файл). В указании имен файлов допустимы обычные маски, например, командой

```
$grep line./*
```

будут найдены строки вида `line` во всех файлах текущего каталога. Шаблон для поиска не обязан быть односложным. Правда, если в нем используются

последовательности символов, разделенные пробелами, последние должны тем или иным способом экранироваться, иначе в качестве шаблона будет воспринято только первое слово. Например, каждый пробел может предваряться символом обратного слэша (\) или просто все искомое выражение заключается в одинарные или двойные кавычки.

Шаблоны могут включать в себя регулярные выражения. Причем список таковых для команды `grep` существенно шире, чем для команд манипулирования файлами. Так, кроме маски любой последовательности символов (*), любого одиночного символа, списка и диапазона символов (`[a...z]` и `[a-z]`), могут встречаться:

- . (точка) — маска любого одиночного (но, в отличие от маски `?`, присутствующего, т. е. ненулевого) символа; таким образом, при задании шаблона `lin.` будут найдены строки, содержащие `line`, но не `lin`;
- `^` и `$` — маски начала и конца строки соответственно: по шаблону `^line`, будут найдены строки, начинающиеся с соответствующего слова, по шаблону же `line$` — им заканчивающиеся;
- маски повторения предыдущего шаблона, `\{n\}` — ровно `n` раз, `\{n,\}` — не менее `n` раз, `\{,m\}` — не более `m` раз, `\{n,m\}` — не менее `n` раз и не более `m` раз.

Маски повторения относятся к так называемым расширенным регулярным выражениям. Для их использования команда `grep` должна быть дана с опцией `-e` (или в форме `egrep`). При этом становятся доступными и другие возможности поиска — например, нескольких текстовых фрагментов (соединенных логическим оператором "ИЛИ") одновременно. Делается это двумя способами. Первый способ — просто перечисление искомых фрагментов, каждый из которых предваряется опцией `-e` (и при необходимости экранируется кавычками):

```
$grep -e pattern1 -e pattern2 files
```

При втором способе оператор между искомыми шаблонами задается в явном виде:

```
$egrep 'pattern1|pattern2' files
```

Таким способом очень легко, например, составить оглавление для длинного текста (при условии некоторой системы рубрикации в нем, разумеется). Для этого достаточно дать команду вроде следующей:

```
$egrep 'Часть|Глава|Раздел|Параграф' filename
```

Для текста, включающего html- или TeX-разметку, роль рубрик могут выполнять соответствующие ее элементы, например:

```
$egrep '<h1>|<h2>|<h3>|<h4>' filename
```

Вывод команды `grep` может быть перенаправлен в файл, а при необходимости и предварительно отсортирован с помощью соответствующих командных конструкций перенаправления и конвейеризации.

Разумеется, тем же способом можно создать общее оглавление для серии фрагментов, записанных в виде самостоятельных файлов — для этого достаточно только перечислить их имена в качестве аргументов команды. Так, например, если есть необходимость составления детальной карты сайта, включающей ссылки на подрубрики внутри отдельных html-документов, следует применить конструкцию типа:

```
$egrep '<h1>|<h2>|<h3>|<h4>' path/*.html > sitemap.html
```

Еще одно замечательное свойство команды `grep` (`egrep`) — возможность получения шаблона не со стандартного ввода (т. е. не путем набора его с клавиатуры), а из файла. Так, если для приведенного выше случая создать простой текстовый файл shablon, содержащий строку

```
Часть|Глава|Раздел|Параграф
```

операцию по сборке оглавления впредь (и в любом тексте, хоть частично совпадающем по структуре с рассмотренным) можно будет выполнять таким образом:

```
$egrep -f shablon filename
```

Опция `-f` и указывает команде, что список параметров должен извлекаться из файла, объявляемого в качестве значения опции.

Список опций команды `grep` не исчерпывается указанными выше. Так, опция `-i` предписывает игнорировать различие регистров внутри искомого выражения, опция `-h` — подавляет вывод имен файлов (выводится только содержание найденных строк), тогда как опция `-l`, напротив, выводит только имена файлов, содержащих найденный шаблон, опция `-n` выводит номера найденных строк в соответствующих файлах. Весьма важной представляется опция `-v`, обеспечивающая инверсию поиска: при указании ее выводятся строки, не содержащие шаблона поиска.

Команда `grep` имеет и аналог для поиска в сжатых файлах — команду `zgrep`. Использование ее в целом аналогично, и останавливаться на нем я не буду.

Сфера применения команды `grep` во всех ее проявлениях очень широка, далеко не исчерпываясь приведенными примерами. Так, эффективно ее использование в едином конвейере с командой `find`, что позволяет выбирать нужные файлы практически по любым критериям (или, напротив, отбраковывать ненужные файлы). Иными словами, как и для команды `find`, возможности, предоставляемые командой `grep`, ограничиваются только потребностями пользователя и его фантазией.

Некоторые общесистемные команды

В заключение главы кратко рассмотрим программы для получения общесистемной информации. Этот вопрос уже затрагивался в ходе предшествующего повествования. Однако здесь он будет рассмотрен с позиций пользователя.

Первой в ряду команд информационного профиля должна стоять, как ни странно, команда echo, исходное назначение которой — просто-напросто вывод сообщений, данных в качестве ее аргумента, на экран. Почему — попробую аргументировать на ближайших же страницах.

Итак, команда echo, в полном соответствии со своим названием, воспроизводит значение своего аргумента на экране. Типичным (и расхожим) примером чего является пресловутое:

```
$echo 'Привет, мир!'
```

ответом на что, прямо как в старом туристском анекдоте, будет:

```
Привет, мир!
```

Если аргумент однословен, необходимости в его экранировании нет — без конкретизации нашего привета допустимо

```
$echo Привет!
```

Справшивается, и в чем же здесь заключена общесистемная информация? Вспомним, что в качестве аргументов и значений опций многих команд ранее мы широко использовали специальные символы (в том числе и в составе регулярных выражений). И далеко не всегда ясно, как и кем эти специальные символы будут обрабатываться — командой ли, или непосредственно интерпретатором оболочки, и какие методы экранирования требуется применить в данном случае. Так вот, именно для проверки этого и может быть использована команда echo — ее аргумент обрабатывается именно командным интерпретатором.

Однако echo в состоянии выполнить и более важную информационную функцию. Если в качестве ее аргумента указать не приветствие в мировом масштабе, а имя переменной среды (что это — будет подробно говориться в главе о командных оболочка), то на выводе команды будет значение этой переменной.

Поясню сказанное на примере. Скажем, вследствие приступа склероза пользователь забыл, под именем кого он авторизовался в текущем сеансе работы. На помощь приходит команда echo в форме

```
$echo $USER
```

ответом на что будет вывод имени пользователя:

```
username
```

В случае аналогичного затруднения с пользовательской командной оболочкой для его разрешения достаточно команды

```
$echo $SHELL
```

на что будет выведено нечто вроде

```
/usr/local/bin/zsh
```

В данных примерах имена переменных даны в верхнем регистре, и в общем случае это правильно. Однако некоторые командные оболочки (например, zsh) допускают ввод имен переменных и в нижнем регистре.

Впрочем, для получения полных сведений о переменных среды предназначена команда `printenv`. Данная без опций и аргументов, она выводит список всех установленных (через профильные ли файлы, или вручную) переменных среды. Если же в качестве аргумента указать имя переменной, на выводе будет дано ее значение:

```
$printenv PATH
/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:\
/usr/local/bin:/usr/X11R6/bin:/home/alv/bin
```

Здесь верхний регистр в имени переменной обязателен.

Большой интерес для пользователя могут представлять и другие общесистемные сведения, например, об оставшемся свободным дисковом пространстве. Для этого предназначена команда `df`, ответом на которую (без опций и аргументов) будет вывод о всех смонтированных файловых системах с указанием степени их заполненности:

```
$df
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/ad0s1a    128990    60764    57908    51%    /
devfs               1        1        0   100%    /dev
/dev/ad0s1h  29838884  1654170 25797604     6%    /home
/dev/ad0s1f    257998       12   237348     0%    /tmp
/dev/ad0s1g   5161198  1058528  3689776    22%    /usr
/dev/ad0s1e   2064302     5706  1893452     0%    /var
/dev/ad1s1   19625094 15472268  2582820    86%    /mnt/d2
```

Как легко видеть, в качестве единицы измерения дискового пространства по умолчанию выступает блок размером 1 Кбайт. Однако указанием опции `-m` можно обеспечить вывод этих же сведений в мегабайтах, а опции `-g` — в гигабайтах. Опции `-h` и `-H` обеспечивают "человеческое" (Human-readable) представление вывода — т. е. с указанием, в чем именно измеряются дисковые разделы:

```
$df -h
Filesystem    Size   Used  Avail Capacity  Mounted on
```

```
/dev/ad0s1a   126M    59M    57M    51%    /
devfs         1.0K   1.0K     0B   100%    /dev
/dev/ad0s1h    28G   1.6G    25G     6%    /home
```

или

```
Filesystem    Size    Used   Avail Capacity  Mounted on
/dev/ad0s1a   132M     62M    59M    51%    /
devfs         1.0K   1.0K     0B   100%    /dev
/dev/ad0s1h    31G   1.7G    26G     6%    /home
```

Различие между этими опциями в том, что `-h` исчисляет "истинные" мегабайты, гигабайты и прочее (т. е. как степени числа 2), а опция `-H` очеловечивает вывод до предела, полагая 1 Гбайт равным 1000 Мбайт и т. д.

Опция `-i` дополнительно выводит сведения об используемых inode, опция `-l` ограничивает сведения локально смонтированными файловыми системами (без учета сетевых накопителей), опция `-t` [значение] выводит сведения только о файловых системах указанного типа. Если же в качестве аргумента команды указать имя файла устройства (или соответствующую ему точку монтирования), сведения будут выведены только о его файловой системе, например:

```
$df /dev/ad0s1a
Filesystem   1K-blocks     Used    Avail Capacity  Mounted on
/dev/ad0s1a     128990    60764    57908    51%    /
```

или

```
$df /home
Filesystem   1K-blocks     Used    Avail Capacity  Mounted on
/dev/ad0s1h   29838884  1654172 25797602     6%    /home
```

Информацию о протекающих в системе процессах, используемой памяти и тому подобных материях можно получить с помощью программы top. Впрочем, на эту тему было сказано достаточно в соответствующей главе.

Некоторые полезные команды разного назначения

В заключительном разделе я приведу некоторые команды разного назначения, которые иногда существенно упрощают выполнение ряда элементарных действий.

```
$ banner [-d] [-t] [-w width] message
```

Вывод заголовка message на стандартный вывод консоли. Заголовок передается символами #. Опцией `-w` задается ширина заголовка в символах. Опция

`-d` дополнительно выводит цифровую матрицу заголовка, опция `-t`, кроме того, подавляет вывод собственно заголовка (остается одна цифровая матрица). Если не указать аргумент `message`, последует запрос на его ввод:

```
banner -w 10
Message:
```

В документации (правда, к Linux-версии) сказано, что выводимый заголовок большой и красивый. Что большой — верно, а вот касаемо красоты... Да и не понятно, что в нем.

```
$ bc [options] [file...]
```

Программа для выполнения интерактивных вычислений. В элементарной форме, без опций и аргументов, являет собой внешне примитивный, но вполне пригодный к употреблению калькулятор. В этом случае после запуска выводится краткая информация о программе (главное в ней — отказ от гарантий), вслед за чем можно приступать собственно к вычислениям:

- сложению (`число1+число2`);
- вычитанию (`число1-число2`);
- умножению (`число1*число2`);
- делению (`число1/число2`);
- нахождению остатка (`число1%число2`);
- возведению в степень (`число1^число2`);
- вычислению квадратного корня — `sqrt(число)`.

В качестве чисел могут выступать любые целые числа и десятичные дроби. В качестве разделителя дроби используется, вне зависимости от установленной локали, именно точка. Результат выводится на экран после нажатия клавиши <Enter>. Предварительно можно задать точность вычислений:

```
scale=#
```

где # — количество цифр после десятичной точки (по умолчанию результат выводится в целочисленной форме), а также основание системы счисления, причем отдельно для входных

```
ibase=#
```

и выходных

```
obase=#
```

данных. Основание (число #) может меняться в интервале от 2 до 16. То есть допустимы вычисления в двоичной, восьмеричной, десятичной и шестнадцатеричной системах счисления, однако заняться расчетами в стиле жрецов древних майя (использовавших двадцатеричную систему счисления) — не удастся.

Если совместно с командой `bc` в качестве аргумента (`file`) загрузить какую-либо стандартную математическую библиотеку, становится возможным вычисление функций — синусов, косинусов, логарифмов и прочих экспонент.

Строка ввода `bc` ведет себя подобно командной строке оболочки, в частности, клавишами <PageUp>/<PageDown> можно просматривать ранее введенные вычислительные конструкции и при необходимости редактировать их, используя стандартные управляющие последовательности: <Ctrl>+<B> — перемещение на один символ назад, <Ctrl>+<F> — на один символ вперед, и т. д. (аналогично `bash`, `tcsh`, `zsh`).

Для выхода нужно в строке ввода набрать `quit` или нажать комбинацию <Ctrl>+<D>.

```
$ cal [options] [месяц] [год]
```

Вывод простенького, но полезного (когда праздники перестаешь отличать от буден) календаря. Без опций и аргументов выводит календарь на текущий месяц в виде:

```
Июнь 2002
вс пн вт ср чт пт сб
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

Аргументы — месяц и год, — задаются в численной форме:

```
$ cal 6 2002
```

причем год — обязательно полностью, иначе можно получить календарь на II в. по Р.Х. Если опустить месяц — будет выведен календарь на все двенадцать месяцев указанного года. Если месяц указан, а год — забыт, то номер месяца будет воспринят как год по Р.Х., со всеми вытекающими.

Посредством опции `-j` можно задать вывод дат для юлианского календаря, при этом даты будут исчисляться в количестве дней с 1 января указанного (или текущего) года:

```
cal -j
Июнь 2002
вс  пн  вт  ср  чт  пт  сб
                       152
153 154 155 156 157 158 159
160 161 162 163 164 165 166
```

```
167 168 169 170 171 172 173
174 175 176 177 178 179 180
181
```

Есть и другой вариант той же команды — `ncal`. Она выводит календарь чуть в другой форме:

```
    Июнь 2002
    пн      3 10 17 24
    вт      4 11 18 25
    ср      5 12 19 26
    чт      6 13 20 27
    пт      7 14 21 28
    сб  1   8 15 22 29
    вс  2   9 16 23 30
```

И плюс к этому, обладает некоторыми дополнительными особенностями: с опцией `-J` выводит юлианские даты в удобопонятном виде:

```
    Июнь 2002
    пн      4 11 18 25
    вт      5 12 19 26
    ср      6 13 20 27
    чт      7 14 21 28
    пт  1   8 15 22 29
    сб  2   9 16 23 30
    вс  3 10 17 24
```

А посредством опции `-p` можно узнать, когда какая страна перешла с юлианского календаря на григорианский. Ну, и опция `-e` способна ответить, какой нынче на дворе день (уж не канун ли Каты Праведного?).

```
$ colrm [start [stop]]
```

Утилита для удаления колонок из входного потока, например, из вывода команд `ls`, `ps` и аналогичных. В качестве стартового аргумента указывается номер символа в строке, начиная с которого, будет производиться удаление колонки вниз), в качестве стопового — номер конечного символа; если последний не указан, все колонки будут удалены до конца самой длинной строки. В учет принимаются и пустые символы. Таким образом, если вывод команды `ps` будет

```
  PID  TT  STAT      TIME COMMAND
  282  v0  I      0:00,22 -zsh (zsh)
  290  v0  I+     0:00,01 joe command.html
  301  v1  I      0:00,13 -zsh (zsh)
```

```
311  v1  I+     0:00,01 less
294  v2  S      0:00,08 -zsh (zsh)
312  v2  R+     0:00,00 ps
```

то для удаления колонки TIME следует перенаправить его на ввод команды colrm следующим образом:

```
$ps | colrm 16 25
```

в результате чего получится следующая картина:

```
PID  TT  STAT COMMAND
282  v0  I    -zsh (zsh)
290  v0  I+   joe command.html
301  v1  I    -zsh (zsh)
311  v1  I+   less
294  v2  S    -zsh (zsh)
335  v2  R+   ps
336  v2  R    -zsh (zsh)
```

Команду colrm с соответствующими аргументами можно определить в качестве глобального псевдонима оболочки zsh — дабы раз и навсегда обеспечить подходящий вывод команды ps.

```
$ column file
```

Используется для форматирования текста в виде таблицы, поля которой объединены в колонки и строки. В качестве разделителя полей по умолчанию воспринимаются символы пробела, что обеспечивается опцией -t, при наличии которой выполняется собственно форматирование. Оно выполняется построчно, однако при опции -x сначала формируются колонки, а уже затем строки. Опцией -c число задается количество колонок.

С помощью команды column легко превратить произвольный текстовый список в квазитабличный структурированный файл, пригодный для считывания в электронную таблицу или базу данных. Например, для форматирования файла list следующего содержания:

```
Имя Отчество Фамилия
Алексей Викторович Федорчук
Елена    Владиславовна  Тихомирова
Виктор   Алексеевич     Федорчук
Ольга    Алексеевна     Федорчук
```

ее можно применить в форме:

```
$ column -t -c 3 list
```

результатом чего будет такой вывод:

```
Имя       Отчество        Фамилия
Алексей   Викторович      Федорчук
Елена     Владиславовна   Тихомирова
Виктор    Алексеевич      Федорчук
Ольга     Алексеевна      Федорчук
```

который может быть перенаправлен в новый (или тот же самый) файл:

```
$ column -t -c 3 list > table
```

Кроме того, команда `column` может воспринимать данные со стандартного ввода — это позволяет создавать заголовки для выводных списков команд типа `ls -l` и т. п. (примеры приведены в документации man column).

```
$ expand file
```

Команда для преобразования символов табуляции в пробелы в текстовых файлах. Результат направляется на стандартный вывод или может быть перенаправлен в файл.

```
$ fmt file
```

Очень простенькая, но полезная команда для элементарного форматирования текста. Она преобразует длинные строки, создаваемые в текстовых редакторах, не имеющих режима переноса слов, типа классического vi, в абзацы со строками удобочитаемой длины. В качестве разделителей абзацев воспринимаются пустые строки. Длина строки по умолчанию — 75 символов, что может быть изменено опцией `-w число_символов`. Пустые строки, символы табуляции, пробелы между словами (в том числе лишние) по умолчанию сохраняются. Однако с опцией `-s` происходит унификация абзацев: между словами устанавливается один пробел, между предложениями — два. Дополнительно может выполнять еще некоторые действия, например, заменять несколько пробелов подряд символом табуляции.

```
$ fold file
```

Команда, родственная предыдущей. Она разбивает абзацы на строки фиксированной длины, которая задается опцией `-w` (по умолчанию — 80 символов). Некоторые реализации этой команды располагают дополнительной опцией `-s`, запрещающей разрывать слова (строки разрываются только по пробелам).

```
$ od
```

Команда для просмотра файла в виде восьмеричных (по умолчанию) или иных других кодов. Формат вывода задается следующими опциями:

- `-c` — вывод в виде символов ASCII и управляющих последовательностей;
- `-a` — в виде символов ASCII и именованных символов;

- `-o` — в виде восьмеричных символов;
- `-u` — в виде десятичных символов;
- `-x` — в виде шестнадцатеричных символов;
- `-f` — в виде чисел с плавающей запятой.

Например файл с именем `file`, имеющий следующее содержание:

```
$ less file
content file
```

при использовании этих опций будет выглядеть таким образом:

```
$ od -c file
0000000   c   o   n   t   e   n   t       f   i   l   e  \n
0000015

$ od -a file
0000000   c   o   n   t   e   n   t  sp   f   i   l   e  nl
0000015

$ od -o file
0000000   067543  072156  067145  020164  064546  062554  000012
0000015

$ od -d file
0000000   28515   29806   28261   08308   26982   25964   00010
0000015

$ od -x file
0000000    6f63    746e    6e65    2074    6966    656c    000a
0000015

$ od -f file
0000000   7,5563101e+31   2,0704130e-19   6,9776422e+22   1,4012985e-44
0000015
```

Кроме этого, опциями можно задать размер выходной строки, считывание только указанного количества байтов или пропуск некоторого количества начальных байтов.

```
$ sort file1... file#
```

Выполняет сортировку строк внутри текстового файла. Имеет три режима работы — собственно сортировка (по умолчанию), режим проверки (опция

-c), сортировался ли ранее данный файл, и режим слияния (опция -m), объединяющий индивидуально отсортированные файлы.

```
$ uniq
```

Команда для выявления уникальных или повторяющихся строк. Обычно применяется к файлам, отсортированным командой `sort`.

```
$ wc file
```

Подсчет слов (опция -w), строк (опция -l) или байтов (опция -c) в заданном файле (или файлах). Без опций выводит все три характеристики файла.

```
$ write user term
```

Отправка текстового сообщения на другой терминал (`term`) некоему пользователю `user`, в том числе и самому себе. Текст сообщения берется со стандартного ввода или может быть получен из файла:

```
$ write user term < file
```

Эту команду можно использовать для формирования сводного документа из нескольких коротких фрагментов.

Дополнительные источники

- Практически единственным источником информации по командам FreeBSD являются соответствующие страницы экранных руководств (manual pages). В дистрибутиве системы они представлены исключительно в англоязычном варианте. Многочисленные русские переводы man-страниц (например, на **http://linux.nevod.ru** и **http://linux.perm.ru**) относятся к командам Linux, которые могут отличаться набором и значением используемых опций.
- Существуют и справочные издания по командам UNIX или Linux. Наиболее полное из известных автору — справочник Скловской С. Команды Linux. — К.: Издательство "Диасофт", 2001. Однако оговорка из предыдущего абзаца относится и к нему.

Глава 15

Командные оболочки

В предыдущей главе мы рассмотрели некоторые команды FreeBSD. Однако команды эти исполняются не в безвоздушном пространстве, а в некоторой среде, отвечающей за их ввод, интерпретацию и исполнение. В качестве такой среды выступает командная оболочка, именуемая также командным интерпретатором, или просто Shell.

Что такое Shell

С одной стороны, термин Shell — родовое обозначение группы программ, объединяющих в себе свойства:

- интерпретатора команд, встроенных и внутренних (отдаленным аналогом чего являлись COMMAND.COM в MS-DOS или командный интерпретатор из Norton Utilites);
- развитого пользовательского интерфейса командной строки;
- мощного интерпретируемого языка программирования.

Иными словами, Shell — это интегрированная, хотя и своеобразная, рабочая среда пользователя, резко отличающаяся по своим принципам от интегрированных сред графического режима, но от этого не менее эффективная. Именно поэтому общепринятый перевод термина Shell как "оболочки" не кажется мне удачным, но именно его придется использовать во избежание неоднозначности.

Командная оболочка запускается различными способами: процессом login после авторизации пользователя в системе, непосредственно командой в командной строке (например, /bin/sh), а также при исполнении команд и сценариев, каждый из которых порождает собственную копию Shell. В зависимости от этого любая оболочка может быть интерактивной (или диалоговой), в которой возможен ввод команд пользователя, и неинтерактивной, непосредственно с пользователем не взаимодействующей, выход из которой происходит автоматически по завершении исполнения команды или сценария.

Кроме того, среди интерактивных оболочек особое место занимает т. н. login Shell, что по-русски я назвал бы оболочкой пользователя — именно в ней последний оказывается после авторизации в системе. Как будет показано в дальнейшем, свойства одной и той же оболочки могут существенно различаться в зависимости от того, каким образом она была запущена.

С другой стороны, Shell — это собственное имя первой командной оболочки первых UNIX-систем, именуемой также оболочкой Борна (Bourne Shell) или, сокращенно, sh (это — имя ее исполняемого файла). Она получила широкое распространение во всех UNIX- и UNIX-подобных системах, именно она (или ее производные) запускаются обычно для исполнения общесистемных конфигурационных сценариев. Совместимость с классическим Shell — одно из условий соответствия стандарту POSIX (Portable Operating System Interface — интерфейсу портируемой, т. е. платформенно-независимой, операционной системы).

Не смотря на это, в чистом виде Shell Борна ныне почти не используется в открытых UNIX-подобных системах. Причина тому, с одной стороны, лицензионные ограничения (среда Борна не является свободно распространяемой в понимании GPL- или BSD-лицензий). С другой же стороны, созданный более четверти века назад, чистый Shell давно прекратил свое развитие, и не поддерживает многие функции, ставшие обычными для современных командных сред.

Однако по образу и подобию Shell Борна был создан ряд командных сред, в большинстве своем открытых и свободно распространяемых, а также наделенных многими дополнительными возможностями по сравнению с оригинальным sh, касающимися в первую очередь интерактивной работы. Наибольшую известность и распространение из них получила среда bash (Bourne Again Shell, что иногда переводят как "Заново рожденный Shell"). Именно bash принят как пользовательская рабочая среда во всех известных мне дистрибутивах Linux. Имеются и другие оболочки, более или менее совместимые с Shell и обладающие своими особенностями. В частности, в FreeBSD в качестве пользовательской оболочки по умолчанию принята /bin/sh — весьма точное воспроизведение классической оболочки Борна.

Однако наряду с производными Shell достаточно распространена и другая группа командных оболочек, происходящая от оболочки C-Shell (сокращенно csh — опять же по имени исполнимого файла). Она была разработана в Университете Беркли одновременно с созданием ранних вариантов BSD UNIX и получила распространение во всех производных последней.

В настоящее время оригинальный csh используется столь же редко, как и классический sh, и по тем же причинам: лицензионным ограничениям и недостаточно развитым возможностям интерактивной работы. Однако в мире UNIX-подобных систем получил распространение один из открытых и свободных ее клонов — оболочка tcsh, сохраняющая совместимость с csh и

включающая множество дополнительных функций, аналогично bash. Оболочка tcsh (маскирующаяся, впрочем, под партийной кличкой /bin/csh) в FreeBSD устанавливается по умолчанию как пользовательская (login Shell) для администратора системы.

Наконец, существует еще и третья линия командных оболочек — линия Shell Корна (ksh), ограниченно совместимая с Shell Борна. Подобно остальным прототипам, она не является свободно распространяемой, и потому в мире открытых систем практически не используется (хотя имеет и открытый клон — оболочку pdksh). Однако именно из нее берут начало многие дополнительные возможности, включенные в упомянутый выше стандарт POSIX (и отсутствующие в изначальном Shell Борна). И еще — на ней в значительной мере основывается оболочка zsh, обладающая наиболее широкими возможностями настройки и интерактивной работы.

Впрочем, все современные представители командных оболочек — bash, tcsh, zsh, — обладают примерно равной функциональностью, выраженной в количестве поддерживаемых внутренних команд и возможностях интерактивной работы. Различия между ними лежат в первую очередь в области синтаксиса собственного языка программирования. В оболочке C-Shell и ее производных синтаксис этот, как и следует из названия, схож с таковым всамделишнего языка программирования C (хотя не следует думать, что от этого сам по себе язык имеет что-то общее с C), тогда как в семействах Shell и K-Shell — ни на что, насколько я понимаю, особенно не похож. Так что для пользователя, не имеющего врожденной склонности к языку C, выбор между ними — дело вкуса, настроения и даже случая.

Следует заметить, что в качестве пользовательской среды могут выступать не только Shell разного рода, но и иные программы. Так, в этом качестве используются интерпретаторы языков программирования (Perl, Tcl). Теоретически рассуждая, средой пользователя можно назначить какой-либо файловый менеджер (Midnight Commander, например) или даже текстовый редактор. Однако это оправданно только при специфических задачах, и к этому вопросу я возвращаться не буду.

Я уже говорил, что csh и ее производные — традиционные среды BSD-систем. Однако, как ни странно, в FreeBSD пользовательской средой по умолчанию выступает оболочка /bin/sh — весьма точное функциональное воспроизведение классического Shell Борна (хотя, по лицензионным соображениям, ею и не являющаяся). Объяснение этому приводится следующее: "Потому что в стандарте POSIX сказано, что все командные процессоры должны вести себя так же, как Shell" (часто задаваемые вопросы по FreeBSD).

На самом деле, конечно, выбор такой среды обусловлен, с одной стороны, требованиями совместимости, с другой — компактности и быстродействия, т. к. многие другие программы требуют для исполнения именно среды sh.

И потому разработчики пошли на сознательное ограничение ее функциональности в интерактивном режиме. Однако такое решение пользователю не навязывается, и в состав дистрибутива включены (и как пакеты, и как порты) основные развитые современные среды — и bash, и tcsh, и zsh, и многие другие: список их в каталоге /usr/ports/Shells превышает два десятка.

Кроме того, при установке системы ее администратор по умолчанию получает в свое распоряжение среду tcsh. Правда, именуется она /bin/csh, но это — не более чем жесткая ссылка на /bin/tcsh, в чем легко убедиться, сравнив идентификаторы `inode` соответствующих файлов (кому лень сравнивать — открою секрет — идентификаторы эти одинаковы). Так что и пользователю не возбраняется выбрать среду по своему вкусу и потребностям. Но к вопросу выбора я еще вернусь. А пока следует рассмотреть то общее, что имеется между всеми командными оболочками, по крайней мере из числа тех, которые условно можно назвать "развитыми".

К числу общих особенностей, в той или иной мере и форме поддерживаемых всеми развитыми оболочками, относятся:

- управляющие последовательности;
- встроенные команды;
- буфер истории команд;
- управление заданиями;
- переменные оболочки и среды;
- псевдонимы;
- средства профильной настройки.

Ниже будет дан обзор всех этих особенностей на примере Shell-совместимых оболочек. Отличительные черты оболочек — клонов C-Shell будут рассматриваться по ходу дела.

Управляющие последовательности

Управляющие последовательности — это комбинации клавиш, предназначенные для автоматизации интерактивных действий в командной строке. Именно они, с точки зрения пользователя, отличают "развитые" оболочки от своих классических прототипов.

Управляющие последовательности по своему назначению могут быть сгруппированы следующим образом:

- средства навигации по командной строке;
- средства автодополнения;
- средства редактирования командной строки;

- средства доступа к истории команд;
- средства автокоррекции и получения помощи;
- средства управления задачами.

Перемещение внутри командной строки может осуществляться стандартными клавишами управления курсором — <Left> и <Right>. Однако, во-первых, они не всегда одинаково работают (и, более того, не всегда работают вообще) на разных типах терминалов. А во-вторых, управляющие последовательности предоставляют в этом отношении много больше возможностей. В современных "развитых" оболочках они более или менее унифицированы, хотя в деталях могут и различаться.

В режиме навигации по умолчанию многие клавишные последовательности перемещения курсора сходны (хотя и не идентичны) клавиатурным комбинациям, применяемым с аналогичными целями в таких текстовых редакторах, как emacs и joe. В большинстве случаев они двухкомпонентные, образуясь сочетанием клавиши <Ctrl> или <Esc> с символьной.

Относительно Escape-комбинаций нужно заметить следующее. Часто в документации к программам и командам оболочки употребляется обозначение клавиши <Meta>, хотя поиски таковой на стандартной клавиатуре были бы безуспешными. Так вот, обычно (по крайней мере для клавиатур стандартных PC) действию клавиши <Meta> эквивалентно нажатие и отпускание (из главы о системной консоли мы помним, что при этом генерируются разные скан-коды) клавиши <Esc>. И потому обозначения <Meta>+<клавиша> и <Esc>−<клавиша> имеют один и тот же смысл.

Так, комбинация клавиш <Ctrl>+<f> вызывает перемещение курсора на один символ вперед (аналогично клавише <Right>), комбинация <Ctrl>+ +<b> — на один символ назад (аналогично клавише <Left>). Комбинации <Ctrl>+<a> и <Ctrl>+<e> служат для перемещения курсора в начало или конец строки соответственно.

Действие комбинаций с использованием клавиши <Esc>−<символ> как бы "усилено" по сравнение с <Ctrl>-комбинациями: <Esc>−<f> перемещает курсор на одно командное "слово" вперед, <Esc>−<b> — на одно "слово" назад.

Регистр литерных клавиш, как правило, значения не имеет (комбинации <Ctrl>+<b> и <Ctrl>+<B>, например, эквивалентны). <Ctrl>-комбинации обычно нечувствительны и к раскладке клавиатуры, тогда как <Esc>-последовательности обычно отказываются работать при переключении, скажем, на кириллицу.

Автодополнение — могучее средство автоматизации интерактивной работы в командной строке. Все "развитые" оболочки (начиная с csh) поддерживают автодополнение имен команд. Почти универсальным средством для этого является клавиша <Tab> (в некоторых вариантах csh-клонов этой цели слу-

жит <Esc>). Однако многие оболочки (bash, zsh, tcsh) позволяют использовать для этой цели управляющие последовательности <Ctrl>+<I> для безальтернативного дополнения и <Ctrl>+<D> — для вывода списка возможных дополнений.

Замечу к слову, что последовательность <Ctrl>+<D> в большинстве оболочек при настройках по умолчанию действует неоднозначно. Указанное выше действие она оказывает, только будучи нажата при положении курсора в конце строки; в середине оной удаляет символ под курсором, а в пустой строке — вызывает выход из текущего экземпляра командной оболочки, о чем я скажу чуть ниже.

Кроме этого, "развитыми" оболочками поддерживается также автоматическое дополнение путей к файлам, в том числе — и когда они указываются как аргументы встроенных и, с некоторыми ограничениями, внешних команд.

Редактирование уже сформированной в командной строке директивы — также отличительная черта "развитых" оболочек. Принципы формирования управляющих последовательностей для этого — сходны с таковыми последовательностей навигационных. Так, в большинстве оболочек комбинация клавиш <Ctrl>+<D> (при положении курсора в середине строки) удаляет символ в позиции курсора, <Ctrl>+<H> — символ перед курсором. Последовательности с использованием клавиши <Esc>, как и при навигации, отличаются "усилением" действия: последовательность <Esc>−<D> удаляет символы от позиции курсора до конца командного "слова", <Esc>−<H> (или <Esc>−<Ctrl>+<H>) — до его начала.

Имеются и средства для преобразования символов: <Esc>−<U> и <Esc>−−<C> — в верхний регистр, <Esc>−<L> — в нижний (в одних случаях действие их распространяется только на символ под курсором, в других — на все командное "слово". Комбинации <Ctrl>+<T> и <Esc>−<T> часто вызывают "перетасовку" символов.

В развитых оболочках имеется и возможность отмены последних действий по редактированию строки (иногда — многоступенчатой). Это, как правило, комбинация клавиш <Ctrl>+<U> или <Ctrl>+<_> (символ подчеркивания).

Большинство оболочек с помощью соответствующей настройки позволяют также использование для навигации и редактирования клавишных команд и комбинаций, аналогичных таковым командному режиму редактора vi (Vim), о котором будет говориться в главе о текстовых редакторах.

Универсальное средство доступа к истории команд — клавиши управления курсором <↑> (последовательный просмотр буфера истории назад) и <↓> (просмотр вперед). Дублируются они комбинациями клавиш <Ctrl>+<P> и <Ctrl>+<N> соответственно. Последовательности <Esc>−<<> и <Esc>−<>> обеспечивают переход к первой и последней команде в буфере истории.

Кроме этого, в некоторых оболочках предусмотрены средства поиска команды в буфере истории — простым перебором (обычно <Esc>−<P> — назад и <Esc>−<N> — вперед) или так называемым наращиваемым (о чем я подробнее скажу далее).

Наконец, наиболее развитые оболочки имеют встроенные функции автокоррекции (именуемые по английски spelling, однако перевести это как "проверка орфографии" я бы не рискнул). Одни из них способны самостоятельно вносить исправления в случае неправильного набора имен команд, другие (например, zsh) могут проделывать это и для имен файлов и путей к ним, указываемым в качестве аргументов. Кроме того, есть и средства оперативного вызова помощи по конкретной команде (например, в zsh — <Esc>−<H>) и определения пути к исполнимому ее файлу (<Esc>−<?>).

Последняя группа управляющих последовательностей обеспечивает управление задачами. Так, комбинации клавиш <Ctrl>+<M> и (или) <Ctrl>+<J> вызывают выполнение введенной (или выуженной из буфера истории) команды (аналогично клавише <Enter>), <Ctrl>+<Z> вызывает перевод текущей задачи в фоновый режим, <Ctrl>+<C> обычно прерывает выполнение текущей задачи. Комбинация <Ctrl>+<D> в пустой командной строке по умолчанию вызывает выход из текущего экземпляра оболочки (обычно и из сеанса пользователя). Впрочем, большинство их позволяет запретить такое действие соответствующими настройками — во избежание случайного прерывания сеанса.

Встроенные команды

Как уже неоднократно говорилось, запуск и исполнение команд — одна из основных задач командной оболочки. Любая из них способна интерпретировать и исполнять любые внешние команды системы. Однако оболочки располагают и собственными средствами — т. н. встроенными командами (Shell Builtins). Отличительная их особенность, во-первых, — отсутствие соответствующего команде исполняемого файла (хотя имена внешних команд подчас совпадают с именами команд встроенных, но это — несколько разные команды). Во-вторых, при исполнении встроенных команд не порождается новый процесс и не происходит запуска новой копии командного интерпретатора — они исполняются в текущем экземпляре Shell. Благодаря этому встроенные команды исполняются обычно быстрее внешних и требуют меньше компьютерных ресурсов.

Набор встроенных команд во многом обуславливает функциональные отличия между оболочками. Наиболее скромен он — в оболочке Борна (и ее римейке — штатной /bin/sh из комплекта FreeBSD), где насчитывается около двух десятков встроенных команд. Однако именно это — тот самый базовый минимум, который работоспособен в любой UNIX-системе. Поэтому имеет

смысл привести список основных команд оболочки /bin/sh и их краткие характеристики — поскольку аналоги таковых имеются и в любой другой оболочке, знание их не помешает в любом случае.

```
$:
```

Так называемая "пустая" команда, не выполняющая никаких действий, а просто анализирующая командную строку.

```
$. file
```

Выполняет команды из указанного файла в текущем экземпляре оболочки, без запуска нового, как если бы содержимое файла набиралось на клавиатуре. Файл, из которого берутся команды, может при этом не иметь атрибута исполнения.

```
$alias псевдоним='команда [опции]'
```

Присвоение команде псевдонима, т. е. краткого имени, подменяющего полное (возможно, с опциями); о псевдонимах подробнее будет сказано в одном из следующих подразделов.

```
$bg [job]
```

Перевод задачи по номеру, указанному в качестве аргумента (при отсутствии последнего — текущей задачи) в фоновый режим.

```
$command команда
```

Выполнение указанной встроенной команды оболочки; используется в том случае, если имя встроенной команды совпадает с определенной ранее функцией.

```
$cd путь
```

Переход в указанный каталог; синонимом ее является команда `chdir`.

```
$echo символы
```

Вывод указанной после команды последовательности символов. С опцией `-n` — без образования новой строки. В форме `echo $ПЕРЕМЕННАЯ` выводит значение указанной переменной.

```
$exec команда
```

Исполнение внешней (не встроенной в оболочку) команды, причем — в новом экземпляре оболочки.

```
$exit
```

Выход из командной оболочки (в случае login Shell — и из сеанса пользователя).

```
$export имя=значение
```

Экспорт переменной оболочки в переменную среды, о чем будет говориться в следующем разделе.

```
$fc [-l]
```

Вывод списка команд из буфера истории; дополнительные опции `-n` и `-r` предписывают вывод без нумерации команд или в обратном (от конца к началу) порядке соответственно.

```
$fg [job]
```

Перевод фоновой задачи (по номеру или текущей) в активный режим.

```
$getopts
```

Действует аналогично С-функции `getopt`.

```
$hash
```

Показывает таблицу запомненных расположений команд.

```
$jobid [job]
```

Вывод идентификатора задачи.

```
$jobs
```

Вывод списка текущих фоновых задач.

```
$pwd
```

Вывод полного пути к текущему каталогу.

```
$read name1... name#
```

Считывание строки из стандартного ввода оболочки, которая используется для присвоения значений указанным переменным `name#`; в качестве значений переменных принимаются слова, разделенные пробелами; если таких слов больше, чем имен, указанных в качестве аргументов, значение последнего составят все избыточные слова.

```
$readonly name
```

Определение переменной `name` как предназначенной только для чтения, что предотвращает ее переопределение.

```
$set
```

Без аргументов выводит все переменные оболочки.

```
$type команда
```

Выводит полный путь к указанной команде (для внешних команд, например, sh is /bin/sh), или указывает, что данная в качестве аргумента команда — встроенная (например, `type is a Shell builtin`).

```
$ulimit
```

Устанавливает или выводит на экран ограничения ресурсов. С опцией `-a` показывает список установленных ограничений, как то: на максимальный размер файла, на количество одновременно запущенных процессов и т. п.

```
$umask
```

Без аргументов — выводит маску для атрибутов вновь создаваемых файлов. Если в качестве аргумента указать новую маску — файлы будут создаваться в соответствии с ней.

```
$unalias псевдоним
```

Удаление указанного псевдонима (противоположность команды `alias`). С опцией `-a` (и без имени псевдонима) — удаление всех установленных псевдонимов команд.

```
$unset name
```

Снимает значение переменной, указанной в качестве аргумента (противоположность команды `set`).

```
$wait job
```

Ожидание завершения задачи `job`.

В "развитых" оболочках этот набор дополняется иными командами, иногда специфичными для данного командного интерпретатора. В частности, в оболочке bash таких встроенных команд более 40, в оболочке tcsh — более 50, а в оболочке zsh — превышает 80.

Буфер истории команд

Поддержка буфера истории команд — еще один непременный атрибут развитых командных оболочек. Как можно было заключить из предыдущего раздела, история команд некоторым образом поддерживается и оболочкой /bin/sh — доступ к ней осуществляется командой `fc`. Однако здесь последовательность ранее введенных команд сохраняется только в период работы текущего экземпляра оболочки — по выходе из него (или завершении сеанса работы) она исчезает.

В "развитых" же оболочках буфер истории команд существует неограниченно долго, и обращение к нему возможно в любом последующем сеансе работы. Вернее, размер буфера определяется некоей переменной, значение которой может быть выражено произвольным числом. Все введенные пользователем команды в пределах этого числа сохраняются в специальном файле в домашнем каталоге пользователя, имя которого может быть различным в разных оболочках, но обычно включает компонент `history`.

Доступ к буферу истории осуществляется специальной командой, встроенной во все оболочки, таковой поддерживающие, — командой `history`. Данная без опций, она выводит полный список команд в их исторической (издревле к современности) последовательности, или некоторое количество команд, определенных соответствующими настройками (о которых будет

говориться позднее). В качестве опции можно указать желаемое количество одновременно выведенных команд. Например, команда

```
$history -2
```

выведет две последние команды из буфера истории:

```
 1023  joe Shell.html
 1024  less ~/.zshrc
```

В случае большого размера буфера можно прибегнуть к его постраничному просмотру с помощью конвейера:

```
$history -1000 | less
```

Важное замечание: просмотр буфера истории следует отличать от просмотра экранного буфера с помощью клавиш <ScrollLock>–<PageUp>/<PageDown>, который обеспечивается средствами системной консоли, а не командной оболочки.

Любая из команд в буфере истории может быть повторно запущена на исполнение. Для этого достаточно набрать в командной строке символ ! (восклицательный знак) и затем, без пробела — номер команды в списке буфера. Например,

```
$!1023
```

для приведенного выше примера повторно откроет файл Shell.html в текстовом редакторе joe.

Как говорилось в разделе об управляющих последовательностях, буфер истории можно построчно просматривать в обоих направлениях с помощью клавиш управления курсором или комбинаций <Ctrl>+<P> и <Ctrl>+<N>, а также переходить к первой или последней команде из буфера. Нажатием клавиши <Enter> текущая просматриваемая команда может быть повторно запущена на исполнение, если необходимо — после ее редактирования стандартными средствами.

Однако при обширной истории команд любая из этих процедур последовательного просмотра может быть весьма длительной. Для ее сокращения предусмотрена такая интересная возможность, как наращиваемый поиск нужной команды в истории команд по одному (или нескольким) из составляющих ее символов. Выполняется он так: после нажатия (при пустой командной строке) клавишной комбинации <Ctrl>+<R> появляется предложение ввести алфавитный символ (или — последовательность символов произвольной длины), заведомо входящий в состав требуемой команды:

```
$
bck-i-search: _
```

Ввод такого символа выведет последнюю из команд, его содержащих. При этом введенный символ будет отмечен знаком курсора. Он не обязательно

будет входить в имя команды, но может быть составляющим ее опций или аргументов (имени файла или пути к нему, например). Следующее нажатие <Ctrl>+<R> зафиксирует курсор на предыдущем символе, в пределах этой же или более ранней по списку команды, и т. д. По ходу дела можно вводить дополнительные символы, детализирующие условия поиска команды (или — ее опций и аргументов).

Например, если команда `history` вывела на экран список следующего вида:

```
$history -5
 1019  links work/demoniada/index.shtml
 1020  ls
 1021  cd work/book
 1022  ls -l
 1023  joe Shell.html
```

и если после нажатия <Ctrl>+<R> для поиска задать символ l

```
$
bck-i-search: _
```

в командной строке будет выведена последняя команда, содержащая этот символ, с фиксацией курсора на нем:

```
$joe Shell.html
bck-i-search: l_
```

При повторном нажатии <Ctrl>+<R> курсор переместится на следующий (назад) символ `l`, и т. д., при одном из следующих нажатий все той же комбинации будет выведена команда

```
$ls -l
bck-i-search: l_
```

Процедуру эту можно продолжать вплоть до достижения требуемого результата — т. е. нахождения той команды, которая нужна именно сейчас. Нажатие клавиши <Enter> в любой из этих моментов запускает найденную (т. е. помещенную в командную строку) команду на исполнение, с завершением поиска, каковой обрывается также и при нажатии комбинации <Ctrl>+<C>. Перед запуском найденная команда может быть отредактирована стандартными средствами — с использованием управляющих последовательностей.

Некоторые командные оболочки допускают весьма изощренное обращение с буфером истории команд. Например, в zsh предусмотрены способы извлечения из него отдельных командных "слов", входящих в сложные конструкции, о чем я расскажу в свое время.

Управление заданиями

Управление заданиями — одна из важных функций командных оболочек, именно благодаря ей практически реализуется истинная многозадачность UNIX-систем. И поэтому она появилась с самого момента их рождения, уже в Shell Борна.

Управление заданиями, которое не следует смешивать с управлением процессами (это явления хотя и родственные, но различные), включает в себя следующие возможности:

- параллельный запуск нескольких программ, исполняемых в фоновом режиме;
- перевод в фоновый режим активной задачи;
- перевод в активный режим задачи, исполняемой как фоновая;
- просмотр списка текущих заданий.

Все эти действия осуществляются встроенными командами оболочки, о которых говорилось выше, поэтому пробежимся по ним в быстром темпе. Запуск нескольких заданий осуществляется посредством конструкции

```
$cmd1 & cmd2
```

и т. д. При этом каждая задача получает свой уникальный идентификатор — номер задания. Его ни в коем случае не следует путать с идентификатором процесса. Номер задания присваивается только в текущем экземпляре оболочки. Если в процессе выполнения какой-либо задачи создается новая копия оболочки (или вызывается другая командная оболочка, например, для выполнения сценария), задания в ней будут нумероваться уже заново.

Просмотреть список текущих заданий (вместе с их номерами) можно посредством встроенной команды `jobs`. Например (не типичная ли это ситуация?), если у нас одновременно запущены проигрыватель mpeg-файлов, электронная таблица и текстовый редактор с помощью конструкции

```
$mpg123 music/almazov/* & sc & joe &
```

вывод команды `jobs` буде выглядеть следующим образом:

```
[1]    running    mpg123 music/almazov/*
[2]  - suspended (tty output)  sc
[3]  + suspended (tty output)  joe
```

из которого видно, что задача `[1]` находится в состоянии исполнения, а прочие — в ожидании (в данном случае — ввода/вывода). Причем — по разному: символ `+` в строке задания `[3]`, вызванного в командной строке последним, показывает, что именно оно будет активизировано командой `fg`

без указания задания. Однако конкретное задание, например, электронную таблицу sc, можно активизировать командой

```
$fg sc
```

Напротив, перевести задание в фоновый режим можно командой `bg [имя_задачи]`. На фоновые задачи не распространяется действие некоторых управляющих последовательностей. Так, в текущем примере прервать проигрывание аудиофайлов стандартной управляющей комбинацией <Ctrl>+<C> не удастся, даже если остальные задачи будут закрыты: для этого придется сначала сделать ее активной командой `fg mpg123`.

Переменные оболочки и среды

Понятие переменных может представлять некоторую трудность для пользователей, не имевших ранее навыков программирования (к коим принадлежит и автор этих строк). Однако в первом приближении все достаточно просто: переменная — это нечто, имеющее имя (начинающаяся с буквы последовательность букв, цифр и подчеркиваний), которому присваиваются некоторые значения, образованные последовательностью символов.

Переменная и ее значение соединяются оператором присваивания. В Shell-совместимых оболочках таковым является символ равенства (=), причем ни слева, ни справа от него не должно быть пробелов. Например, ранее уже неоднократно говорилось об определении таких переменных, как редактор по умолчанию или программа постраничного просмотра. Делается это следующим образом:

```
EDITOR=joe
```

или

```
PAGER=less
```

Здесь `EDITOR` и `PAGER` — имена переменных, а `joe` и `less` — присвоенные им значения. Значение переменной может определяться как результат выполнения какой-либо команды. Так, переменная

```
DATE=`date`
```

принимает значение текущей даты и времени, потому что при такой форме определения сначала выполняется команда `date`, а результат ее не выводится на экран, а присваивается переменной; обращаю внимание на обратные одинарные кавычки, в которые заключена команда — именно это и предписывает ее выполнение. Чтобы узнать значение переменной, достаточно дать команду

```
$echo $DATE
```

в ответ на что будет выведено нечто вроде

```
пятница, 17 мая 2002 г. 14:41:17 (MSD)
```

Любимый народом символ `$` перед именем переменной показывает, что в качестве вывода команды `echo` должно быть принято ее значение. А полный список определенных на данный момент переменных может быть определен упомянутой в прошлом разделе командой `set`. Вывод ее может быть очень различен, однако с большой долей вероятности в нем будут встречены такие строки:

```
USER=alv
MAIL=/var/mail/alv
HOME=/home/alv
PS1=$
PS2=>
TERM=cons30r
PATH=/sbin:/bin:...:/home/alv/bin
SHELL=/usr/local/bin/zsh
MANPATH=/man:...:/usr/local/man
```

определяющие имя пользователя, авторизованного в текущем сеансе (`USER`), его командную оболочку по умолчанию (т. н. login Shell — `SHELL`), пути к его домашнему каталогу (`HOME`), почтовому ящику (`MAIL`), пути поиска исполняемых файлов и страниц документации (`MANPATH`), тип терминала (`TERM`), форму первичного (`PS1`) и вторичного (`PS2`) приглашения командной строки.

Следует разделять переменные оболочки (Shell variables) и переменные среды, или окружения (environmet variables). В чем разница между ними? В главе о процессах говорилось, что запуск в командной строке любой команды порождает новый процесс, а именно — очередную копию командной оболочки (Shell). Так вот, переменные, определенные вышеуказанным образом (через присваивание `name=value`), имеют силу только в родительском процессе, дочерний же Shell в общем случае может обладать своим набором переменных. И те, и другие будут переменными оболочки.

Однако в ряде случаев желательно, чтобы определение многих переменных (таких, как `PATH`, `PAGER`, `TERM`, `EDITOR` и многие другие) распространялись и на все процессы, порожденные в рамках текущей пользовательской оболочки. Для этого они должны быть превращены в переменные среды. Достигается это т. н. экспортом переменной после присваивания ей значения, например:

```
EDITOR=joe
export EDITOR
```

или

```
EDITOR=joe; export EDITOR
```

Данные примеры имеют силу для Shell-совместимых оболочек. В csh-клонах (например, в tcsh) переменные определяются несколько иначе. Для присваивания значения переменной оболочки используется форма

```
set NAME value
```

а переменные среды определяются специальной командой:

```
setenv NAME value
```

Таким образом, чтобы определить редактор по умолчанию как переменную среды, в tcsh достаточно строки

```
setenv EDITOR joe
```

Переменная оболочки, экспортированная в среду, в csh-совместимых оболочках может быть изъята из окружения — для этого достаточно команды `setenv` с указанием имени определенной и экспортированной переменной (без значения последней) в качестве аргумента. В sh-совместимых оболочках для этого приходится подчас снять определение переменной (хотя некоторые оболочки позволяют обойтись без этого).

Все командные оболочки располагают некоторым количеством предопределенных переменных, для которых компетенция пользователя ограничивается присвоением им значений. Число их различно, но обычно в него входят такие переменные, как `PATH` (список каталогов для поиска исполняемых программ), `HOME` (домашний каталог пользователя), `TERM` (тип терминала), `PS1` (первичное приглашение командной строки) и ряд других.

Псевдонимы

Псевдонимы (Aliases) — еще один способ упрощения интерактивной работы в командных оболочках, изначально им присущий. Суть его — в подмене длинного имени команды (или имени команды с некими фиксированными опциями) коротким и мнемонически значимым именем. Делается это в общем случае следующим образом:

```
$alias alias_name='cmd [options]'
```

в Shell-совместимых оболочках, и

```
$alias alias_name cmd [options]
```

в csh-клонах. Причем в первом случае кавычки необходимы только в случае наличия опций. Так, если для вызова Midnight Commander вместо неуклюжего `midc` есть желание использовать привычное `mc`, в кавычках необходимости нет:

```
alias mc=midc
```

Однако обычно псевдонимы используются для изменения опций команд по умолчанию. При этом имя псевдонима отнюдь не обязано отличаться от имени подменяемой команды. Например, как мы помним, команды типа `cp` и `mv` по умолчанию подменяют одноименные файлы без предупреждения, команда же `rm` также удаляет файлы без лишних вопросов. Как правило, это бывает нежелательно, и потому целесообразно присвоить им псевдонимы, совпадающие с собственным именем, но в определение псевдонима внести соответствующую опцию:

```
$alias cp='cp -i'
$alias mv='mv -i'
$alias rm='rm -i'
```

В результате этой операции "умолчальная" форма этих команд будет запрашивать подтверждения потенциально опасных действий.

Другой популярный пример использования псевдонимов — для комбинаций команды `ls` с различными опциями просмотра вывода:

```
alias ls='ls -FG'
alias ll='ls -l'
alias la='ls -A'
```

В результате первая ("умолчальная") команда обеспечивает вывод с использованием цвета и визуальным различением регулярных файлов и каталогов, вторая — вывод в "длинном" формате, третья — показ скрытых файлов.

Если требуется отменить действие какого-либо псевдонима, например, при уничтожении большого количества файлов для большей уверенности в правильности своих действий псевдоним для `rm` (или любой другой команды) можно временно отключить командой:

```
unalias rm
```

а по завершении процедуры — восстановить:

```
alias rm='rm -i'
```

Можно также гуртом аннулировать действие всех псевдонимов:

```
$unalias -a
```

Некоторые командные оболочки (например, zsh) позволяют присваивать псевдонимы командным конструкциям, что много расширяет сферу их применения.

Средства профильной настройки

Переменные среды и окружения, как и псевдонимы, отнюдь не обязательно задавать интерактивно, в командной строке. Наиболее важные из них опре-

деляются обычно глобально, в файлах настройки командных оболочек. Именно о них и пойдет речь в этом разделе.

В принципе свойства оболочек описываются в файлах инициализации, именуемых также файлами начальной загрузки, профильными файлами и еще многими иными именами. Однако суть дела от этого не меняется — это некие файлы в каталоге /etc, корневом каталоге и (или) домашнем каталоге пользователя. В последнем случае они часто называются также dot-файлами, поскольку имена их в каталоге $HOME начинаются с точки, что делает их не видными при просмотре командой `ls` по умолчанию.

Имена, количество и назначение профильных файлов варьируют в зависимости от конкретной оболочки, однако каждая из них имеет как минимум один такой файл. Для классической /bin/sh это файл /etc/profile, где определяются такие переменные, как `PATH`, `TERM` и `PAGER`, и осуществляется их экспорт в окружающую среду.

Очевидно, что файл /etc/profile доступен для редактирования только администратору. Однако пользователь может создать в своем домашнем каталоге его аналог — файл $HOME/.profile для индивидуальной настройки оболочки. В этом случае при запуске оболочки как пользовательской (login Shell) происходит считывание сначала общесистемного профильного файла, а затем — "домашнего" профильного файла. Соответственно, установленные в последнем переменные и псевдонимы "забивают" общесистемные настройки, обеспечивая требуемое поведение оболочки.

При интерактивном запуске /bin/sh из командной строки и, тем более, при ее вызове из сценариев обращения к профильным файлам не происходит. Если в такие копии оболочки требуется передать какие-либо установки, их следует указать в отдельном dot-файле, который должен быть определен в одном из основных профильных файлов как переменная оболочки, например, таким образом:

```
ENV=$HOME/.shinit; export ENV
```

Аналогичный профильный файл предусмотрен и для "умолчальной" оболочки csh (напомню, что в FreeBSD это — не более чем ссылка на оболочку tcsh, однако, вызванная таким образом, она имеет несколько иные свойства). Это — файл /etc/cshrc, при необходимости его аналог, $HOME/.cshrc, также может быть создан в домашнем каталоге пользователя. В отличие от /bin/sh, файлы csrc-группы считываются (сначала — общесистемный, затем — "домашний") не только при запуске пользовательской оболочки, но и при любом интерактивном запуске csh. Кроме того, они дополняются файлом /etc/csh.login (и его "домашним" аналогом — $HOME/.login), обращение к которому происходит только при запуске оболочки пользователя (login Shell), хотя и после файлов cshrc-группы. Впрочем, для csh порядок считы-

вания dot-файлов определяется при компиляции и может быть изменен пересборкой оболочки.

"Развитые" оболочки способны воспринимать настройки своих совместимых "примитивных" собратьев: bash считывает настройки из profile, tcsh — из cshrc и login, zsh же в состоянии почерпнуть сведения для своей работы из профильных файлов как /bin/sh, так и csh. Однако они имеют, разумеется, и собственные средства настройки, причем весьма гибкие.

Так, bash, будучи запущен как пользовательская оболочка, осуществляет работу с профильными файлами в следующем порядке. Сначала происходит обращение к файлу /etc/profile, если таковой существует. Затем считываются файлы ~/bash_profile и ~/.bash_login, если же их нет, bash берет за основу файл ~/.profile. Последним отыскивается файл ~/.bashrc. Впрочем, в bash порядок обращения к файлам настройки может быть изменен при компиляции. Для интерактивных, но не пользовательских копий оболочки дело ограничивается считыванием файла ~/bashrc. Наконец, неинтерактивные копии оболочки получают сведения о своих настройках из файла, определенного в переменной ENV, если таковая определена.

Кроме этого, в bash предусмотрено еще два профильных файла — ~/.inputrc, описывающий свойства собственно командной строки, и ~/.bash_logout, выполняемый при выходе из пользовательской оболочки (и — только из нее).

Оболочка tcsh, будучи запущенной как пользовательская, обращается к файлам /etc/csh.cshrc и /etc/csh.login, если они имеются, затем — к файлу ~/.tcshrc или, если он не обнаружится, к ~/.cshrc. После этого происходит считывание файла истории команд — ~/.history и, под конец, ~/.cshdirs, в котором описывается стартовый каталог оболочки. При выходе из пользовательской оболочки выполняются действия, предписанные в файлах /etc/csh.logout или ~/.logout. Для всех прочих копий оболочки, кроме пользовательской, сведения берутся только из файлов ~/.tcshrc или ~/.cshrc.

Самая сложная схема работы с профильными файлами, обеспечивающая максимальную гибкость настройки, — у оболочки zsh, однако это будет предметом специального рассмотрения.

Столь многоступенчатая форма настройки современных "развитых" оболочек может показаться излишне усложненной. Однако она позволяет настраивать окружение отдельных копий оболочки индивидуально, в зависимости от того, в каком качестве они выступают — как пользовательские, просто интерактивные или неинтерактивные. При этом очевидно, что наибольшее внимание следует уделить именно оболочке пользователя: во-первых, она — единственная и незаменимая, во-вторых, именно из нее, через переменные среды, наследуются (или — не наследуются) самые главные свойства дочерних оболочек.

И еще одно важное замечание. На протяжении всего предыдущего повествования неоднократно упоминалась команда `su`, позволяющая временно получить полномочия root-оператора. При этом пользователь мог оказаться в недоумении, попытавшись запустить какую-нибудь заведомо имеющуюся команду системного администрирования и получив в ответ сообщение об ошибке типа

```
command not found.
```

Дело же в том, что при запуске `su` без опций и пользовательская оболочка, и все переменные окружения остаются без изменений. В частности — переменная `PATH`, которая для оболочки пользователя и администратора может быть определена по-разному — а по умолчанию, так и есть, в пользовательской оболочке переменная эта, как правило, не включает каталоги /sbin и /usr/sbin, в которых команды административного назначения и собраны. И потому для полного использования возможностей root-оператора команда `su` должна запускаться с опцией `-l`, которая симулирует полную авторизацию от root-оператора, с установкой соответствующих переменных окружения.

Выбор пользовательской оболочки

Рассмотрев общие свойства командных оболочек, пора вернуться к вопросу выбора. Разумеется, он имеет смысл только для оболочки пользователя (login Shell), т. к. интерактивно можно запустить любой командный интерпретатор, а вызов его из сценария остается на совести автора последнего.

Очевидно, что выбор по умолчанию — оболочка /bin/sh, — в качестве пользовательской подходит в наименьшей степени, слишком слабы ее интерактивные возможности. Хотя совсем отказываться от ее использования было бы ошибкой: именно она обеспечивает переносимость пользовательских сценариев между различными UNIX-системами. Да и возможностей ее в этом плане более чем достаточно для большинства пользовательских задач. К тому же программированию на классической Shell посвящено много литературы, и именно эту оболочку можно избрать в качестве тренировочного инструмента при создании сценариев.

Еще большей ошибкой было бы, отказавшись от использования /bin/sh, удалить ее с диска — за ненадобностью. Система в этом случае просто откажется загружаться. Дело в том, что все файлы инициализации (rc-файлы из каталога /etc) представляют собой просто сценарии оболочки, в явном виде, первой строкой

```
#!/bin/sh
```

вызывающие именно этот, самый примитивный, но и самый быстрый, Shell. И при его отсутствии выполнение этих сценариев оказывается невозможным — со всеми вытекающими последствиями.

Конечно, собственно /bin/sh можно заменить одноименной символической ссылкой на bash или zsh — обе эти оболочки способны ее успешно эмулировать. Но — только в том случае, если при установке системы не создавался отдельный раздел для /usr/local — а именно в его подкаталоге, /usr/local/bin, окажутся и bash, и zsh после установки, из портов ли, или из пакетов. А поскольку rc-файлы выполняются до монтирования прочих файловых систем, кроме коренной, файлы-источники такой ссылки просто не будут найдены — с тем же результатом, что и просто при стирании /bin/sh.

Оболочка tcsh (особенно в своем исконном виде, а не как эмулятор csh) обладает многими очень привлекательными свойствами при интерактивной работе: большим количеством встроенных команд и управляющих последовательностей, широкими возможностями настройки и гибкостью оных. Однако главное ее преимущество сказывается при написании достаточно сложных и длинных сценариев: С-подобный синтаксис ее встроенного языка обеспечивает существенно больший их лаконизм, чем скрипты для Shell-совместимых интерпретаторов.

Правда, именно при создании сценариев использование tcsh может быть ограничено ее несовместимостью с классическим Shell и стандартом POSIX. Хотя, если не предназначать эти сценарии для иных UNIX-систем, пользователю FreeBSD именно tcsh может показаться оптимальным выбором. Освященным к тому же традицией — ведь именно в берклианских клонах UNIX зародился ее прототип, оболочка csh.

Думаю, не ошбусь, если скажу, что самой распространенной оболочкой в открытых UNIX-подобных системах является bash. Она принята по умолчанию во всех известных мне дистрибутивах Linux, число пользователей которых много превосходит приверженцев BSD-систем. Ее достоинства — совместимость со стандартом, способность полностью эмулировать классический Shell (в Linux-системах /bin/sh — обычно ссылка на bash, который и вызывается для исполнения системных и пользовательских сценариев). Возможности интерактивной работы и настройки — не у́же, чем для оболочки tcsh, хотя последняя и кажется мне более "продвинутой" (впрочем, это — субъективное представление).

Наконец, bash очень хорошо документирована и многократно описывалась как в Сети, так и на бумаге (притом — и на русском языке). Достаточно сказать, что описание свойств командной оболочки и приемов работы в ней, имеющееся в любой толстой книге по Linux или UNIX, в явном или неявном виде основывается именно на оболочке bash. Так что это — выбор почти беспроигрышный.

И наконец, zsh. Это, если можно так выразиться, квинтэссенция полезных свойств всех командных оболочек. Она практически вдвое превосходит своих самых "развитых" сестер по числу встроенных команд, имеет максималь-

ный набор управляющих последовательностей, самые мощные средства редактирования командной строки. Возможности настройки ее практически безграничны. Ну, а подключаемые модули самого разного назначения (вплоть до встроенного FTP-клиента) обеспечивают функциональность, недоступную ни в bash, ни в tcsh. А если добавить к этому более-менее полную совместимость со стандартом POSIX и возможность эмуляции не только sh, но и ksh, и даже некоторое подобие воспроизведения функций csh, — становится ясным, что zsh вполне достойный кандидат на звание универсальной пользовательской оболочки.

Единственный ее недостаток — малая распространенность и, как следствие, малая известность. В большинстве книг оболочка zsh только упоминается, хотя подчас и с указанием на наличие несравненных ее достоинств — но и все. Хотя она прекрасно документирована разработчиками, в русскоязычной литературе конкретных сведений о zsh практически не найти.

Дабы исправить это упущение, я и решил посвятить следующую главу специально оболочке zsh. Кроме того, как я уже говорил, она аккумулирует все свойства прочих оболочек, давая представление о максимальных возможностях программ этого класса. И, наконец, пользователь bash или tcsh найдет в ней немало для себя знакомого (а то и полезного).

Однако прежде следует сказать пару слов о программах, относимых обычно к категории файловых менеджеров, а именно — о потомках командира Нортона. Которые в ряде случаев могут играть (а то и играют по предпочтению) роль командных оболочек.

Оболочки-визуализаторы

— Кто твоя мать, рядовой Петров?
— Коммунистическая партия Советского Союза!
— Кто твой отец?
— Вы, товарищ замполит!
— Какова твоя заветная мечта?
— Сиротой бы остаться...

Из старого советского анекдота

Всенародная наша любовь к файловому менеджеру командира Нортона и многочисленным его потомкам для любых платформ известна и в комментариях не нуждается. Не миновала любовь эта и FreeBSD. Где в роли наследников славного командира выступают Midnight Commander и deco (Demos Commander).

Первая из этих программ весьма точно воспроизводит внешний облик своего родителя, но далеко превзошла его функционально. Что, впрочем, отно-

сится только к версии для Linux: FreeBSD-версию иначе чем командирской сироткой язык назвать не поворачивался. Начать с того, что по непонятным причинам вызывается она там не волшебной аббревиатурой mc, а весьма неуклюжим и непривычным буквосочетанием midc. Что, конечно, можно решить просто введением дополнительного псевдонима в профильном файле, например, для csh

```
alias mc  midc
```

или, для Shell-совместимых оболочек,

```
alias mc='midc'
```

Так что это еще полбеды. Однако в MC стандартной (из коллекции пакетов) сборки не обнаруживается встроенного редактора (таковым по умолчанию выступает не кто иной, как vi).

Конечно, и это поддается лечению — или переопределением переменной EDITOR, или использованием внешнего редактора. Благо последний мог имитироваться редактором le, идеологически и по интерфейсу весьма схожим с Norton Editor (и обладающим, к тому же, рядом полезных качеств).

Но с чем трудно примириться любому старому командирскому комбатанту — так это с непотребным поведением клавишных комбинаций по умолчанию. Так, комбинация <Ctrl>+<Enter> отнюдь не помещает файл под курсором в командную строку, а <Ctrl>+<O> вместо отключения панелей вызывает просто немедленный (и — без предупреждения) выход из midc.

Во избежание недоразумений повторю — все сказанное относится именно к midc под FreeBSD в сборке по умолчанию. И все эти "умолчальные" недостатки могут быть ликвидированы путем пересборки с соответствующими настройками. Однако, по данным Андрея Лаврентьева (см. дополнительные источники), они варьируют от версии к версии (и, на мой взгляд, не вполне тривиальны).

И тут наступает психологический момент вспомнить о другом командирском отпрыске для FreeBSD — Demos Commander (или, сокращенно, deco). Получившем свое имя от пионера российского провайдерства (фирмы Demos, автор же программы — Сергей Вакуленко). Конечно, ничего специфического для FreeBSD он не содержит, компилируясь, судя по всему, под любую UNIX-подобную систему. Однако в дистрибутивах Linux он обычно не встречается, а в FreeBSD входит стандартно, как в виде пакета, так и порта.

Надо заметить, что с первого взгляда deco производит еще более сиротское впечатление, чем даже MC (рис. 15.1): съежившиеся на пол-экрана панели удручающе-черного цвета (на некоторых терминалах приобретающие почему-то окраску в династические цвета дома Романовых), крайне ограниченные возможности интерактивной настройки, возможности файловых мани-

пуляций — на уровне ранних версий Norton Commander (в частности, невозможно скопировать, переместить или удалить каталог целиком, с содержащимися в нем файлами).

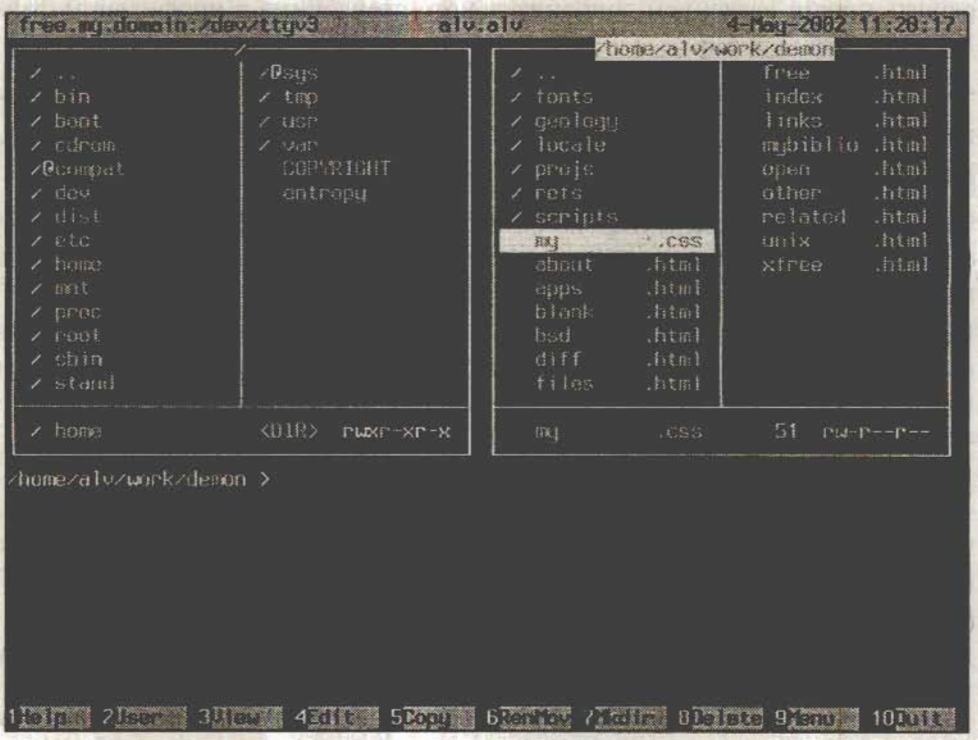


Рис. 15.1. Визуализатор командной оболочки deco

Возможности просмотра и редактирования также на первый взгляд разнообразием не блещут: можно воспользоваться встроенными вьювером и редактором (с весьма ограниченными возможностями) или подключить внешние аналоги. Ну, а о таких излишествах, как встроенный FTP-клиент — и говорить не приходится.

Однако если набраться некоторого терпения, начинаешь понимать всю великую сермяжную правду, заложенную в deco. Во-первых, поведение клавишных комбинаций в нем привычно (пользователю NC) и предсказуемо. Во-вторых, возможности настройки его не столь уж бедны, как кажется на первый взгляд. Есть возможность создания пользовательского меню (вызываемого клавишей <F2>) — неотъемлемого атрибута "командирских" файловых менеджеров. К любым типам файлов, определяемых по маскам, можно привязать не только команды, но и сколь угодно сложные их конст-

рукции. А самое главное — в deco обнаружился замечательный режим командной строки.

Включение этого режима (через меню или комбинацией клавиш <Ctrl>+<P>) приводит к тому, что действия в панелях становятся невозможными, но зато открываются самые широкие возможности манипулирования непосредственно командами оболочки. После чего становится ясным, почему по умолчанию панели deco свернуты на половину экрана (что, впрочем, легко изменить — включив через меню или комбинацией <Ctrl>+<F> режим Full Screen): нижняя его часть, в сущности, представляет собой обычное терминальное окно.

Сам по себе deco предлагает на выбор одну из двух встроенных командных оболочек — /bin/sh и /bin/csh. Правда, интерактивные их возможности очень бедны: не поддерживаются ни автодополнение, ни псевдонимы, ни прочие давно уже привычные радости. Попытки настройки оболочек также остаются безуспешными (для меня, честно говоря, так и осталось загадкой, откуда deco черпает сведения о переменных оболочки и окружения.

Тем не менее, работа в режиме командной строки оказывается весьма комфортной. При этом оказывается, что deco, в сущности, никакой не файловый менеджер; или, мягче сказать, эта ее функция — лишь дополнение к основной, а именно: визуализации действий в командной строке. Кстати сказать, в экранной документации deco так и называется — visual Shell, а отнюдь не файловый менеджер.

Дополнительные источники

Источники информации по командным оболочкам очень многочисленны. В первую очередь, разумеется, за дополнительными сведениями следует обращаться к man-страницам для соответствующих программ (sh, csh и т. п.).

Далее, разные аспекты использования командных оболочек освещены в следующих сетевых публикациях:

- Соловьев А. Программирование на Shell (UNIX) — **http://www.citforum.ru/programming/Shell/index.shtml**;
- Интерпретатор командного языка Shell — **http://www.citforum.ru/operating_systems/Shell/index.shtml**;
- BASH конспект — **http://linux.perm.ru/doc/devel/languages/bash-conspect.html**;
- Особенности работы оболочки bash — **http://linux.yaroslavl.ru/Docum/Gnu-util/bash/bash.html**;
- Ян Макдональд. Более продуктивная работа с bash 2.x — **http://www.softerra.ru/freeos/16449**;

- Крутиков М. П. Командный интерпретатор CShell — **http://www.citforum.ru/operating_systems/articles/tut_2.shtml**;
- сравнение особенностей различных командных оболочек (на английском языке, правда) можно обнаружить по адресу: **http://www.looking-glass.org/Shell.html**;
- в этой книге за кадром осталось рассмотрение вопросов программирования на встроенных языках оболочки. Для восполнения этого пробела можно обратиться к уже упоминавшейся книге Тэйнсли Д. Linux и UNIX: программирование в Shell. — К.: Издательская группа BHV, 2001.
- что же касается файловых менеджеров, выполняющих роль командных оболочек, сведения об использовании Midnight Commander под FreeBSD можно почерпнуть на сайте Андрея Лаврентьева — **http://unix1.jinr.ru/~lavr/**.

Глава 16

Командная оболочка zsh

Первоначальный вариант командной оболочки zsh был создан Паулем Фальстадом (Paul Falstad), ныне же она поддерживается и развивается сообществом энтузиастов при координации Петера Стефенсона (Peter Stephenson), который является также автором прекрасного руководства пользователя. Официальный сайт проекта — **http://www.zsh.org**. В текущей ветви FreeBSD оболочка zsh представлена версией 4.0.4 — последней стабильной на настоящий момент. Она может быть установлена как из пакета, так и посредством системы портов обычным образом, как это было описано в *главе 5*.

Обзор возможностей

Разумеется, zsh поддерживает все стандартные возможности современных "развитых" оболочек, о которых говорилось в прошлой главе. При этом реализация их наиболее богата — в числе особенностей оболочки очень развитые возможности редактирования командной строки (включая спеллинг введенных команд и их автокоррекцию), множество видов приглашений командной строки и практически неограниченные возможности их настройки, большое количество управляющих клавишных последовательностей, максимальное количество (среди известных мне Shell) встроенных команд (более 80), возможность подключения внешних модулей (plug-ins). Благодаря этому рискну предположить, что zsh представляет собой наиболее удобную среду для работы в командной строке. Что и постараюсь обосновать в настоящем разделе.

Для ввода в командную строку в zsh очень широко применяется дополнение команд и путей к файлам, осуществляемой клавишей <Tab> или управляющей комбинацией <Ctrl>+<I>. При безальтернативном дополнении оно *происходит автоматически, в ином же случае на экран выводятся варианты* выбора. Повторным нажатием клавиши табуляции или управляющей комбинации осуществляется перебор всех доступных вариантов дополнения.

Конечно, пользователя bash или tcsh этим удивить трудно. Однако zsh допускает автоматическое дополнение в аргументах тех команд, где эти оболочки бессильны, например, при вызове man-страниц. Так, для вызова полной экранной документации по zsh (на которой, кстати, я подробно остановлюсь в разделе о дополнительных источниках) достаточно набрать в командной строке

```
$man zsha
```

и нажать клавишу <Tab> — аргумент команды `man` будет автоматически дополнен до полного имени страницы документации, zshall.

Более того, zsh оказывается способным к автодополнению даже в опциях команд (разумеется, если они не имеют односимвольного формата). Например, для команды `tar` можно набрать

```
$tar --ext
```

что будет автоматически дополнено до

```
$tar --extract
```

Если же при наборе той же опции ограничиться двумя символами, на экран окажутся выведенными варианты:

```
$tar --ex
option
--exclude        --exclude-from   --extract
```

из которых можно вручную или перебором выбрать требуемый. Особенно показательно автодополнение в таких сложных синтаксически командах, как `find`. Здесь оно работает и для критерия поиска, и для опций действия. Так, последовательность

```
$find / -na
```

будет автоматически дополнена до

```
$find / -name
```

А затем для указания следующего действия можно ограничиться вводом символа дефиса

```
$find / -name filename -
```

и выбрать необходимое действие (например, вывод на экран) из предложенного списка.

Возвращаясь к вариантам выбора, предлагаемым при неоднозначности для автоматического дополнения, следует заметить, что он сопровождается весьма подробными комментариями к предлагаемым именам команд и файлов. Они объясняют, что среди предложенного является каталогом, что — регулярным файлом, что — командой, встроенной или внешней.

Для редактирования уже введенных команд в zsh предусмотрено большое количество управляющих последовательностей (описанных в соответствующем разделе). Они позволяют удалять отдельные символы и целые командные "слова", фрагменты командной строки, инвертировать регистр символов, менять символы местами.

Для осуществления редактирования командной строки zsh располагает эффективными средствами перемещения по ней — в конец и в начало строки, на один символ или на одно "слово" вперед и назад. Кроме того, посредством соответствующих настроек в конфигурационных файлах можно корректировать само понятие командного "слова" путем определения символов-разделителей.

Кроме средств навигации и редактирования по умолчанию (сходных с применяемыми в текстовых редакторах emacs и joe, т. н. emacs-режим), возможно переключение в режим редактирования в стиле редактора vi, что при должном навыке обеспечивает еще большую скорость обработки командной строки. Причем эмулируется как командный режим vi, так и его режим вставки.

Однако еще более замечательны средства автокоррекции ввода (т. н. spelling). В случае ошибки ввода команды на экран выводится правильный вариант, который пользователю предлагается подтвердить (нажатием клавиши <y>). А если прибегнуть к клавишной комбинации <Meta>+<S>, ошибка будет исправлена автоматически.

Средства автокоррекции действуют в отношении как встроенных команд zsh, так и любых внешних команд. Более того, они распространяются на пути к файлам, вводимым в качестве аргументов команд. Правда, исправлять ошибки в многосимвольных опциях zsh не умеет...

Использование псевдонимов (alias) — также традиционная особенность "развитых" командных оболочек: куда как проще раз и навсегда определить в профильном файле команду `ls` как псевдоним самой же себя, но с опциями `-FG`, нежели каждый раз вспоминать, как отличить в ее выводе каталоги от обычных файлов. Однако в zsh это дополняется возможностью определения псевдонимов для командных конвейеров.

Для этого встроенная команда `alias` снабжена опцией `-g`, позволяющей определить псевдоним для компонента командного конвейера, например:

```
$alias -g L='|less'
```

После этого данный псевдоним можно указывать после команды, требующей постраничного вывода, в качестве ее псевдоопции, например:

```
ls path L
```

что обеспечит постраничный вывод списка файлов указанного каталога path точно так же, как и в случае обычного конвейера

```
$ls path | less
```

Кроме этого, глобальные псевдонимы можно применять для конструкций перенаправления. Например, при поиске файлов утилитой find за пределами пользовательского каталога обычно выводится масса сообщений об ошибках — путях к каталогам, к которым отказано в доступе:

```
find: /path: Permission denied
```

Полезной информации эти сообщения не несут, и от них можно избавиться перенаправлением потока сообщений об ошибках (сиречь файла с идентификатором 2) в "нулевое" устройство:

```
2>/dev/null
```

Так вот, zsh позволяет присвоить псевдоним и такой конструкции, например:

```
alias -g N='2>/dev/null'
```

После этого для избавления от мешающих сообщений достаточно указать соответствующую "опцию" в конце командной строки, после опции действия:

```
$find /path -name file -print N
```

Определяемые пользователем переменные также имеют место быть в оболочке zsh. При этом она располагает большим количеством предопределенных переменных — например, только для описания приглашения командной строки их определено шесть.

В zsh предусмотрены весьма эффективные средства для работы с буфером истории команд. История команд может быть просмотрена обычными для "развитых" оболочек средствами, в ней можно осуществлять перебор команд в любом направлении, простой и наращиваемый поиск, извлекать из истории не только целые команды, но и отдельные командные "слова" (составляющие опций или аргументов команды).

Встроенные команды

Командный интерпретатор zsh поддерживает все стандартные встроенные команды классического Shell, описанные в предыдущей главе, а также большинство команд "развитых" оболочек типа bash и tcsh. Однако в дополнение к ним здесь предусмотрен еще ряд команд, специфичных для zsh. Перечислять их все (а тем более описывать) кажется мне неуместным — этому посвящена страница экранной документации man zshbuiltins. Поэтому ниже я кратко рассмотрю лишь наиболее важные для пользователя команды из числа неупомянутых в предыдущей главе.

```
$bindkey
```

Команда, позволяющая манипулировать управляющими клавишными последовательностями, в частности, включать режим навигации и редактирования командной строки в стиле vi.

`$bye`

Выход из командной оболочки, аналогично exit (но не logout).

`$dirs`

Вывод т. н. стэка каталогов — аналога буфера истории для команды `cd`.

`$disown [job]`

Удаление указанного (или, при отсутствии аргумента, текущего) задания из списка заданий.

`$emulate Shell_name`

Включение режима эмуляции одной из следующих командных оболочек: sh, ksh или csh.

`$kill`

В отличие от внешней команды с тем же именем *(см. главу 8)*, отправляет сигнал заданию, идентифицируемому по его номеру, а не по PID процесса.

`$print`

Выводит свой аргумент на стандартный вывод, подобно команде `echo` (отличия касаются несколько разной интерпретации специальных символов). Имеет многочисленные опции, из которых уникальна опция `-z` — направление аргументов в буфер редактирования, из которого они могут быть извлечены командой `read -z`.

`$pushd pathname`

Подобно команде `cd`, производит переход в каталог, указанный в качестве аргумента. Каталог, бывший текущим, помещается в стэк каталогов.

`$read`

Подобно одноименной команде из стандартного набора Shell, считывает строки из стандартного ввода оболочки и присваивает их переменной в качестве значения. Однако с опцией `-z` вместо стандартного ввода воспринимает строку из буфера редактирования, помещенную туда командой `print -z`.

`$setopt`

Установка опций оболочки. Без аргументов — вывод списка установленных опций.

`$typeset`

Установка параметров оболочки.

Командные конструкции

Как и в любой другой оболочке, в zsh команды могут объединяться в командные конструкции — фонового (оператор `&`), последовательного (опе-

ратор ;), условного выполнения (операторы && и ||), перенаправления ввода/вывода (операторы >, >>, <, <<) и конвейеризации (оператор |). Однако в дополнение к ним, есть и немалое количество операторов, специфических для zsh и расширяющих функции стандартных.

Рассмотрим дополнительные операторы перенаправления. Так, оператор >| (или >!, что эквивалентно) действует подобно >, однако если перенаправление указывает на существующий, но пустой (т. е. нулевой длины) файл, результат будет добавлен к нему. Оператор же >>| в случае указания на несуществующий файл — создаст его. При перенаправлении вида

```
$cmd <<< file
```

на стандартный ввод команды cmd передается не содержимое файла file, а его имя. Имеются и своего рода операторы-аббревиатуры. Например, |& представляет собой краткую форму для конструкции 2>&1 | — т. е. перенаправления стандартного вывода и потока ошибок предшествующей команды на стандартный ввод последующей.

Еще одна специфическая особенность zsh — возможность множественного перенаправления: конструкция

```
$cmd > file1 > file2
```

перенаправит вывод команды cmd сразу в два файла, а конструкция

```
$cmd > file1 >> file2
```

выполнит то же самое, но для второго файла — в режиме дополнения. Множественное перенаправление работает и на вывод:

```
$cat < file1 < file2
```

выведет на экран содержимое обоих файлов (как в старом китайском анекдоте — сначала один, потом второй), а конструкция

```
$sort < file1 < file2
```

в один присест произведет в них сортировку эквивалентно конструкции

```
$cat file1 file2 | sort
```

Далее, для zsh характерно т. н. перенаправление без команд, т. е. конструкция

```
$ < file
```

просто выведет на экран содержимое файла file, как если бы перенаправление ввода указывало бы на команду cat, less или more. Собственно говоря, именно последняя здесь и подставляется в неявном виде по умолчанию.

Синтаксис командных конструкций не отличается от такового Shell-совместимых оболочек. Однако в zsh не требуется обязательных пробелов

перед объединяющими операторами и (или) после них. То есть здесь допустимой оказывается форма

```
$cat>filename
```

или

```
$ls path |less
```

наряду с более строгими

```
$cat > filename
```

или

```
$ls path | less
```

Кроме того, возможна группировка аргументов перенаправления по шаблону. Так, приведенная выше конструкция для одновременной сортировки содержимого двух файлов может быть записана в форме

```
$sort < file{1,2}
```

Специфическая особенность интерпретатора zsh — возможность т. н. предисполнимой модификации команд (precommand modifier), осуществляющейся перед их интерпретацией. Модификаторы эти, как следует из названия, задаются в командной строке до имени команды и оказывают влияние на ее исполнение. В числе модификаторов:

- `nocorrect` — отключение автокоррекции вводимых команд и путей;
- `exec` — исполнение команды в текущей копии оболочки, без порождения нового процесса;
- `command` — исполнение внешней команды вместо одноименной встроенной;
- `builtin` — исполнение встроенной команды оболочки вместо одноименной внешней.

Кроме простых команд, интерпретатор zsh поддерживает т. н. сложные команды (complex command). Это — не что иное, как условные операторы (`if... then... else... fi`), операторы цикла (`for`, `while`), выбора (`case`, `select`) и подобные из классического Shell. Синтаксис операторов совпадает с таковым оболочки Борна, однако поддерживается также т. н. альтернативная, или краткая, их форма.

Управляющие последовательности

Управляющие последовательности оболочки — это то, что придает комфорт при работе в командной строке. Для оболочки zsh полный их список (более 130) вызывается встроенной командой `bindkey`. Как и в других оболочках, обычно это комбинации <Ctrl>+<символ>, <Meta>+<символ>, <Meta>+ +<Ctrl>+<символ>. Напомню здесь, что действие клавиши <Meta>, отсут-

ствующей в стандаратных раскладках клавиатуры (но определяемой дополнительно), эквивалентно нажатию и отпусканию клавиши <Escape>.

Ниже приведено описание главных управляющих последовательностей оболочки zsh, сгруппированных по назначению:

- навигация;
- редактирование;
- автодополнение;
- управление заданиями;
- управление историей команд;
- последовательности прочего назначения.

В zsh для выполнения некоторых действий предусмотрено несколько вариантов управляющих последовательностей. Иногда же, напротив, одна и та же управляющая последовательность оказывает разное действие в зависимости от положения курсора в строке (типичный пример — комбинация <Ctrl>+<D>), что будет оговариваться в каждом конкретном случае. В приводимых ниже списках алфавитные символы даны в верхнем регистре, однако на самом деле в zsh регистры для них не различаются.

Навигация

Как следует из названия, группа навигационных управляющих последовательностей предназначена для перемещений внутри командной строки. Разумеется, для этих целей можно пользоваться и стандартными клавишами управления курсором. Однако управляющие последовательности предоставляют множество дополнительных возможностей для навигации, весьма способствующих ее ускорению.

- <Ctrl>+<F> — перемещение курсора на один символ вперед.
- <Ctrl>+<B> — перемещение курсора на один символ назад.
- <Meta>+<F> — перемещение курсора на одно "слово" вперед.
- <Meta>+<B> — перемещение курсора на одно "слово" назад.
- <Ctrl>+<A> — перемещение курсора в начало строки.
- <Ctrl>+<E> — перемещение курсора в конец строки.
- <Ctrl>+<@> — установка метки в командной строке.
- <Ctrl>+<X>–<Ctrl>+<X> — перемещение курсора из текущего положения к метке (установленной посредством <Ctrl>+<@>) и обратно.
- <Ctrl>+<X>–<=> — вывод информации о позиции курсора в форме:

```
/=>> ls ~/work/
EOF  point 11 of 11(100%)  column 10
```

Редактирование

Как и для навигации, для редактирования командной строки могут применяться стандартные клавиши <Backspace>и <Delete>. Однако они не на всех типах терминалов работают одинаково. И к тому же возможности управляющих последовательностей для редактирования много шире.

- <Ctrl>+<H> — удаление символа перед позицией курсора.
- <Ctrl>+<D> — в середине строки — удаление символа в позиции курсора.
- <Ctrl>+<K> — удаление символов, начиная с позиции курсора и до конца строки.
- <Ctrl>+<W> — удаление символов влево от позиции курсора до конца командного "слова".
- <Meta>+<Ctrl>+<H> — удаление символов от позиции курсора до начала "слова" (назад).
- <Meta>+<D> — удаление символов от позиции курсора до конца слова (вперед).
- <Ctrl>+<U> — удаление всех символов в строке.
- <Ctrl>+<Q, <Meta>+<Q> — очистка командной строки.
- <Meta>+<C> — преобразование символа под курсором в верхний регистр.
- <Meta>+<U> — преобразование в верхний регистр символов от позиции курсора до конца "слова".
- <Meta>+<L> — преобразование в нижний регистр символов от позиции курсора до конца "слова".
- <Meta>+<'> — содержание строки (целиком) заключается в одинарные кавычки.
- <Ctrl>+<J>, <Ctrl>+<M> — переход на новую строку; обычно эквивалентно нажатию клавиши <Enter>, однако если строка заканчивается символом \ (обратный слэш), вызывает вторичное приглашение для ввода многострочной команды.
- <Meta>+<T>, <Ctrl>+<T> — перемещение символа под курсором на одну позицию влево (назад).
- <Ctrl>+<X>−<Ctrl>+<O> — переключение в режим замены и обратно в режим вставки.
- <Ctrl>+<_>, <Ctrl>+<X>−<U>, <Ctrl>+<X>—<Ctrl>+<U> — многоступенчатая отмена последнего ввода; для команды из буфера истории, без опций и аргументов — отмена команды целиком; для опций и аргументов из буфера истории не действует.
- <Meta>+<S, <Meta>+<$> — спеллинг "слова" в позиции курсора (как для команд, так и для путей).

Автодополнение

- <Ctrl>+<I> — дополнение команды или пути; в случае невозможности однозначного дополнения повторное нажатие вызывает перебор доступных вариантов (и для команд, и для путей).
- <Ctrl>+<D> — в конце строки — вывод списка возможных дополнений (с комментариями — встроенная команда, внешняя команда, путь).
- <Meta>+<Ctrl>+<D> — вывод списка возможных дополнений, в отличие от предыдущей последовательности, работает вне зависимости от положения курсора.

Управление процессами

- <Meta>+<A> — выполнение команды и возврат ее в командую строку для повторения.
- <Ctrl>+<G>, +<Ctrl>+<G> — прерывание с возвратом приглашения командной строки.

История команд

- <Ctrl>+<N> — просмотр команд из буфера истории вперед.
- <Ctrl>+<P> — просмотр команд из буфера истории назад.
- <Ctrl>+<O> — исполнение текущей команды и переход к следующей в буфере истории.
- <Meta>+<N> — поиск в буфере истории вперед.
- <Meta>+<P> — поиск в буфере истории назад.
- <Ctrl>+<R>, <Ctrl>+<X>−<R> — наращиваемый поиск в буфере истории назад.
- <Ctrl>+<S>, <Ctrl>+<X>−<S> — наращиваемый поиск в буфере истории вперед.
- <Meta>+<.> — вставка "слова" из буфера истории, начиная с последнего и далее перебором назад.
- <Meta>+<_>, +<Ctrl>+<_> — вставка последнего "слова" из буфера истории (аналогично предыдущему); это и другие сочетания с <Ctrl> после <Meta> работают и при кириллической раскладке клавиатуры.
- <Meta>+<<> — переход к началу буфера истории.
- <Meta>+<>> — переход к концу буфера истории.
- <Meta>+</> — вывод полного списка команд из буфера истории; повторные нажатия — перебор команд от начала к концу буфера.

- <Meta>+<,> — вывод полного списка команд из буфера истории; повторные нажатия — перебор команд от конца к началу буфера.
- <Ctrl>+<X>–<Ctrl>+<K> — очистка буфера истории команд.

Дисплей и помощь

- <Ctrl>+<L>, +<Ctrl>+<L> — очистка экрана.
- <Meta>+<H> — вывод man-страницы для текущей команды.
- <Meta>+<?> — вывод полного пути к файлу текущей команды.

Разное

- <Ctrl>+<X>–<~>> — вывод списка пользователей.
- <Ctrl>+<X>–<Ctrl>+<V> — переключение в режим редактирования и навигации в стиле редактора vi. Возврат в режим в стиле emacs (режим по умолчанию) — комбинацией <Ctrl>+<C>.

В заключение раздела об управляющих последовательностях остается добавить, что большинство из них совпадают с таковыми оболочек bash и tcsh, но, как правило, несколько шире функционально и к тому же более многочисленны.

Настройка zsh

Оболочка zsh обладает очень богатыми и гибкими возможностями конфигурирования посредством комплекса профильных файлов. В базовом комплекте таких файлов (в каталоге /usr/local/share/examples/zsh) — три:

- zshenv, считываемый при каждом запуске zsh, как в интерактивном, так и в неинтерактивном режиме;
- zshrc, исполняемый при интерактивном запуске zsh;
- zlogin, определяющий свойства zsh в случае, если он является пользовательской оболочкой (login Shell).

Они могут быть скопированы в домашний каталог пользователя для выполнения индивидуальных настроек. Кроме того, zsh, запущенный в качестве пользовательской оболочки, способен получать сведения о своей настройке из файла ~/zprofile, дублирующего функции файла ~/zlogin. При выходе же из пользовательской оболочки отрабатывается файл ~/zlogout. Впрочем, примеров двух последних файлов в штатном комплекте нет, и обязательными они не являются.

Рассмотрим сначала "умолчальное" содержание главных конфигурационных файлов, а затем посмотрим, какие изменения в них могут быть внесены для наиболее комфортной работы.

Очевидно, что, поскольку файл ~/.zshenv считывается при любом запуске оболочки zsh, в нем должны присутствовать лишь опции, определяющие самые общие ее свойства. И действительно, пример данного файла содержит только пути поиска команд, т. е. переменные оболочки PATH. Целесообразно дополнить их в соответствии со своими потребностями и, дабы не возвращаться к этому вопросу более, экспортировать их в окружающую среду. Так, в моей системе это выглядит следующим образом:

```
PATH=(
  /bin
  /sbin
  /usr/bin
  /usr/sbin
  /usr/X11R6/bin
  /usr/local/bin
  .
  "$HOME/bin"
)
export PATH
```

Очевидно, что внесение в переменную PATH текущего каталога (.) может (а в некоторых случаях и должно) быть пропущено из соображений безопасности. В большинстве случаев этого достаточно — прочие переменные среды важны при интерактивной работе, а не при исполнении сценариев.

Если предполагается использовать zsh преимущественно в интерактивном режиме (в том числе и как пользовательскую оболочку), основное внимание следует уделить файлу ~/.zshrc. Именно в нем определяются псевдонимы, большинство переменных среды, функции и т. п. Рассмотрим подробнее те опции этого файла, обуславливающие удобство интерактивной работы.

Пример файла /usr/local/share/examples/zsh/zshrc начинается с определения переменной cdpath. Она играет ту же роль для каталогов, что и переменная path для исполнимых файлов: ею описываются первоочередные пути поиска каталогов, указываемых в качестве аргумента команды cd. Определяется эта переменная следующим образом:

```
cdparh (path1 path2... path#)
```

Заключенным в скобки и разделенным пробелами значениям переменной целесообразно придать пути к наиболее часто используемым каталогам. После этого для перехода в любой его подкаталог в качестве аргумента команды cd достаточно набрать его имя. Например, если одним из значений этой переменной будет выступать /usr/local, то для перехода в каталог /usr/local/share потребуется только

```
$cd share
```

Кроме этого, следует уделить внимание переменной manpath, где указываются пути поиска страниц документации для команды man. Определенный в файле примера "умолчальный" вариант рассчитан на некую "усредненную" UNIX-систему и должен быть скорректирован в соответствии с конкретными реалиями, например:

```
manpath=(/usr/X11R6/man /usr/share/man /usr/local/man)
export MANPATH
```

Еще одна переменная, имеющая отношение к путям поиска файлов — fpath, описывающая, где следует искать пользовательские функции для автоматического считывания. По умолчанию это каталог ~/.zfunc, однако изменить его на любой другой не сложно.

Далее в файле zshrc определяются псевдонимы. Здесь целесообразно сохранить строки

```
alias mv='nocorrect mv'        # no spelling correction on mv
alias cp='nocorrect cp'        # no spelling correction on cp
alias mkdir='nocorrect mkdir' # no spelling correction on mkdir
```

отключающие автокоррекцию для соответствующих команд и их аргументов — иначе при любом копировании файла или создании каталога как альтернатива введенному аргументу будет предлагаться имя уже существующего. Кроме того, во избежание случайной перезаписи файла псевдонимы команд mv, cp, rm можно расширить опцией -i.

Далее определяются глобальные псевдонимы — те, которые определяются для фрагментов командных конструкций и играют роль своего рода опций, например:

```
alias -g H='|head'
alias -g T='|tail'
```

Указанные в качестве дополнительных опций для команд постраничного просмотра (после аргументов), они обеспечат вывод первых или последних десяти строк файла соответственно. Впрочем, в этих же псевдонимах можно указать и количество выводимых строк, например:

```
alias -g H='|head -3'
```

после чего ответом на команду типа

```
$less filename H
```

будет вывод первых трех строк указанного файла.

Следует уделить внимание и группе опций, связанных с управлением буфером истории команд. Перво-наперво, определяется файл для его сохранения, обычно это

```
HISTFILE=~/.zhistory
```

Затем определяем опции HISTSIZE и SAVEHIST, значением которых являются число команд, сохраняемых в буфере истории в текущем сеансе, в первом случае, и команд, запоминаемых в файле истории, — во втором. Очевидно, что второе бессмысленно делать больше первого, а руководство пользователя рекомендует для них равные значения, например:

```
HISTSIZE=1000
SAVEHIST=1000
```

Далее обеспечиваем наращивание файла истории после каждого сеанса работы:

```
setopt    APPEND_HISTORY
```

И одновременно предотвращаем засорение его дублирующими командами, например, в самой жесткой форме:

```
setopt    HIST_IGNORE_ALL_DUPS
```

В результате, как видно из названия опции, из истории будут удаляться все дублирующие команды. Существуют и более мягкие опции, например, HIST_IGNORE_DUPS — игнорирование дубликатов только в течении данного сеанса.

Одна из самых привлекательных особенностей zsh — настройка вида приглашений командной строки. Здесь предусмотрено четыре их уровня: различный вид можно придать первичному приглашению (PROMPT, или PS1, допустимы обе формы этой переменной), приглашениям при вводе многострочных команд (PROMPT2 или PS2), приглашению при использовании оператора select (PROMPT3 или PS3), приглашению при отладке (PROMPT4 или PS4). Есть также специальные переменные, определяющие вид приглашения при коррекции команд (SPROMPT), и дополнительного приглашения, выводимого в правой части первичной командной строки (RPROMPT).

В качестве приглашений могут использоваться:

- полное (%M) или сокращенное (%m) имя хост-машины;
- текущий каталог (%c) или путь к нему в различных формах — от корневого (%/) или домашнего каталога пользователя (%~);
- номер текущей команды в буфере истории (%!) или строки в данном сеансе работы (%i);
- имя пользователя (%n);
- имя командной оболочки или исполняемого сценария (%N);
- текущее значение переменной $SHLVL (%L), указывающей на уровень вложенности подоболочки; т. е. пользовательская оболочка (login Shell) имеет значение $SHLVL=1, вызванная из нее копия — $SHLVL=2, и т. п.;

- вывод кода завершения (0 — без ошибок, 1 — ошибка исполнения) последней исполнявшейся команды (`%?`);
- номер текущего терминала в различных формах — в виде `v#` (`%l`) или `ttyv#` (`%y`);
- дата (`%w`, `%W`) и время (`%t`, `%T`, `%*`) в разных форматах;
- индикация работы в режиме суперпользователя по команде `su` (`%#`);
- просто любые последовательности символов типа стрелок и текстовых сообщений, вплоть до поздравления с началом рабочего дня.

Плюс к этому, приглашения могут быть оформлены визуально различными видами: выделением жирным шрифтом (boldface mode) или повышенной яркостью (underline mode), инвертированием цветов (standout mode). Это достигается следующими последовательностями:

- `%S` — включение инвертированного (standout mode, черное на белом) режима отображения командной строки;
- `%U` — включение т. н. underline mode, выраженной в том, что текст передается ярко-белым цветом (или, напротив, приглушенно-серым при standout mode);
- `%B` — включение boldface mode, т. е. отображение вводимого текста полужирным начертанием.

Для отключения режимов standout, underline и boldface требуется задать последовательности `%s`, `%u` и `%b` соответственно: сами собой они не отключаются, даже если убрать включающую последовательность из приглашения. Кроме того, все режимы отображения распространяются только на приглашение того уровня, для которого установлены явным образом, например, `PS1`. Поэтому их удобно использовать для выделения, например, вторичного приглашения командной строки.

По умолчанию в качестве первичного приглашения принято краткое имя хост-машины, что на пользовательском, тем более локальном, компьютере не несет смысловой нагрузки. Посмотрим, как сделать его более информативным. Так, здесь целесообразно вывести путь к текущему каталогу, индикацию работы root-оператора и стрелку указания на поле ввода команды:

```
PROMPT='%~%#=>> '
```

что придаст ему вид

```
~/current_dir%=>>
```

при работе обычного пользователя и

```
/current_dir#=>>
```

при временном получении полномочий администратора командой su. В качестве дополнительного правостороннего приглашения можно вывести номер терминала и (или) текущее время:

```
RPROMPT=' %y-%T'
```

В результате командная строка примет вид

```
~/current_dir%=>>          ttyv#-чч:мин
```

Аналогично при желании и необходимости можно настроить и приглашения всех остальных уровней.

Завершающий штрих модификации файла .zshrc — предотвращение случайного выхода из командной оболочки комбинацией клавиш <Ctrl>+<D>. Для этого следует включить опцию

```
setopt    IGNORE_EOF
```

В файле .zlogin целесообразно определить некоторые переменные, имеющие смысл только при запуске zsh в качестве пользовательской оболочки, как то: тип терминала, текстовый редактор и программу постраничного просмотра по умолчанию, например:

```
EDITOR=joe; export EDITOR
PAGER=less; export PAGER
TERM=${TERM:-cons30r}; export TERM
```

Наконец, zsh, запущенному в качестве пользовательской оболочки, можно определить и действия, производимые при завершении сеанса работы. Для этого потребуется создать файл ~/.zlogout и внести в него соответствующие директивы. Какие — зависит от задач пользователя. Однако одна такая директива напрашивается —

```
clear
```

для очистки экрана по выходе из сеанса данного пользователя.

В этом разделе я остановился лишь на малой доле возможностей по настройке zsh. Более подробные сведения можно почерпнуть из дополнительных источников. В которых, в частности, приведены адреса страниц, содержащих примеры пользовательских конфигурационных файлов, подчас весьма изощренных.

Дополнительные источники

Как уже говорилось, русскоязычные источники информации по оболочке zsh практически отсутствуют. Однако это компенсируется чрезвычайно богатой англоязычной документацией. Достаточно сказать, что только штатный

комплект ее в упакованном архиве с расширением tar.gz составляет около 2 Мбайт, включая man-, info- и html-страницы.

В текущей версии zsh обнаруживается 12 man-страниц, посвященных описанию отдельных аспектов этой оболочки (man zsh — вводные сведения и список остальных страниц, man zshbuiltins — описание встроенных команд, man zshparam — описание опций оболочки и т. д.). Кроме того, имеется и сводная страница экранной документации — man zshall.

Кроме этого, на сайте проекта (**http://www.zsh.org**) можно обнаружить очень подробное руководство пользователя, написанное Петером Стефенсоном (Peter Stephenson) и доступное в форматах PostScript, TeX и HTML.

Наконец, на этом же сайте имеется много ссылок на Web-страницы с примерами конфигурационных файлов для оболочки zsh, примеров пользовательских сценариев и функций.

Глава 17

Текстовые редакторы

Думается, не будет большим преувеличением сказать, что из всех приложений UNIX и UNIX-подобных систем важнейшим является текстовый редактор. С его помощью (а иногда — и исключительно при его посредстве) настраивается система, пишутся скрипты и сценарии, составляются программы для пользователей. Некоторые из них используют текстовый редактор и по прямому назначению — для составления просто повествовательных текстов.

Вводные замечания

В этой главе речь пойдет исключительно о текстовых редакторах консольного режима. Все их можно разделить на два класса — редакторы командного стиля и меню-ориентированные редакторы. В первых навигация по тексту и его обработка осуществляются отдачей прямых директив, вроде: перейти на пять слов вперед, удалить пятую снизу строку, заменить строку номер пятнадцать и т. п. Действия в меню-ориентированных редакторах, как и следует из названия, осуществляются более интерактивно (и — более привычно для пользователя Windows-редакторов).

Пользователи Windows в качестве универсального средства для работы с текстами в большинстве случаев используют программы, именуемые текстовыми процессорами. Таковые представлены в FreeBSD, о чем я буду говорить в одной из следующих глав. Однако именно редакторы — традиционный инструмент всех UNIX-систем.

Текстовые редакторы обладают рядом преимуществ перед процессорами. Главное из них — это универсальность. Поскольку выходной материал в них представляет по определению чистый ASCII-файл, он может быть прочитан в любой среде и на любой платформе, не требуя специальных конверторов. Что особенно важно для документов с символами кириллицы. Кроме того, с помощью редакторов можно готовить html-страницы, осуществлять верстку

документов для TeX, править конфигурационные файлы, писать исходники программ и многое другое.

Кроме того, редакторы, в силу своей структуры, существенно удобнее процессоров для составления длинных структурированных текстов нарративного характера. Каковое, особенно при претензиях на оригинальность, требует сосредоточенности, трудно достижимой при изобилии форматирующих возможностей процессоров. Впрочем, это — мое субъективное мнение. Однако остается фактом, что подготовленный в редакторе документ в дальнейшем может быть трансформирован в любой текстовый процессор или систему верстки, где его можно подвергать любому оформлению.

Сказанное выше относится ко всем текстовым редакторам, как консольным, так и визуальным. Тем не менее, и внутри этого семейства первые имеют определенные плюсы по сравнению со вторыми. Первый, традиционный, — быстродействие и нетребовательность к ресурсам, — постепенно теряет актуальность с ростом мощи современных машин. Однако второй плюс — существенно меньшая нагрузка на зрение, — остается, как бы ни совершенствовались мониторы и видеокарты.

К тому же при составлении оригинальных текстов (будь то техническое руководство, программа или беллетристическое произведение) развитые возможности графического режима, такие как шрифтовое разнообразие, визуальное представление документа, возможность вставки иллюстраций и прочее, — фактически роли не играют (поскольку все это нужно на стадии оформления, а не создания). А возможности параллельного просмотра различных материалов, обмена данными между ними и прочего — прекрасно реализуются и в консольном режиме (по крайней мере, если речь идет о UNIX-подобных системах). Так что я просто не вижу необходимости обращаться к графическому режиму.

Наконец, третье соображение касается достоинств командных редакторов по сравнению с меню-ориентированными. Последние, конечно, легче в освоении, особенно при эпизодическом использовании. Однако при превышении некоего минимального уровня практических навыков командные редакторы обеспечивают много большую скорость работы. Доказать это количественными измерениями довольно сложно (поскольку определяется пресловутым человеческим фактором), но имеющие соответствующий опыт, думаю, со мной согласятся, остальных же прошу поверить на слово — все сказанное выше опробовано на собственной шкуре и выстрадано годами создания текстов самого разного объема, характера и назначения.

Следует заметить, что большинство традиционных UNIX-редакторов принадлежат к командному классу. Среди его представителей, с одной стороны, — наиболее простые и легковесные (с точки зрения требований к ре-

сурсам) программы. К ним принадлежит, например, редактор ee, кратко (а больше о нем и сказать особо нечего) описанный во вводном конспекте *(см. главу 2)*. Ниже будет охарактеризован другой представитель "легкой" весовой категории — редактор lpe.

Разумеется, ee и lpe не исчерпывается список редакторов такого класса: в FreeBSD в качестве портов и пакетов доступны, например, pico (компонент почтовой программы pine) и его аналог nano, по возможностям (да и идеологически) близкие к ee.

С другой стороны, именно командными редакторами являются наиболее мощные инструменты комплексной обработки текста (а иногда — и не только текста), такие, как vi (вернее, его современная модификация — Vim) и emacs. Обоим им посвящено много литературы, и потому подробно рассматривать их я не буду. Однако наличие некоторых навыков работы с vi необходимо пользователю любой UNIX-системы, в том числе и FreeBSD. Так что некоторое внимание ему уделить придется.

Наконец, существует категория редакторов, занимающих промежуточное положение между мощными средствами типа Vim и emacs (последний както в одной из публикаций даже удостоился титула "операционная среда" — вспомним, что именно так величалась в свое время Windows 3.1) и простенькими редакторами. Это — joe и jed. Первый из них представляется мне оптимальным с точки зрения баланса между простотой и функциональностью, и потому ему будет посвящена отдельная (восемнадцатая) глава.

Редактор jed будет охарактеризован вкратце, как промежуточное звено между командными и меню-ориентированными редакторами. Из представителей же последней группы будет описан le — весьма развитый и удобный ee представитель.

Редактор lpe

Этот редактор назван его автором — Хрисом Смитом (Chris Smith), Lightweight programmer's editor. Что применительно к случаю я перевел бы как Легковесный Редактор для Программистов. Под легковесностью, вероятно, понимается его компактность и нетребовательность к ресурсам. Поскольку возможности его вполне достаточны для несложной работы.

При запуске редактора командой `lpe` без параметров перед глазами возникает почти пустой экран с предложением нажать <Ctrl>+<E> для получения помощи. Что отнюдь не лишне — никакого меню нет и в помине, все действия осуществляются только комбинациями клавиш. Узнать назначение которых, кроме как из Help, неоткуда. Хотя нет, имеется еще и info-файл, вызываемый командой

```
$info lpe
```

и содержащий еще более подробные сведения. Документации в формате man-страниц эта программа не имеет.

Так что знакомство с редактором наичнается с двух этих документов. Поскольку он требует практически автоматических навыков в обращении с управляющими комбинациями клавиш. Однако по приобретении таких навыков можно убедиться, что все действия по вводу и редактированию текстов в lpe выполняются легко, просто и быстро. А подсветка синтаксиса делает его применение довольно удобным при несложном редактировании html-страниц.

Так что, дабы не заниматься отсебятиной, я просто прокомментирую содержание info-файла, когда это покажется мне необходимым.

Итак, он может быть запущен следующим образом:

```
$lpe [ --help | --version ]
```

или

```
$lpe --mode значение -- file
```

В первом случае выдается справка по использованию lpe или сведения о его версии. Во втором — вызывается соответствующий файл; если таковой не существует — создается пустой документ.

Необязательная опция `mode` принимает значение соответствующего языкового режима, например, `htmlmode`. Хотя режим автоматически устанавливается для файлов соответствующих типов. По крайней мере, для html-страниц — безошибочно, с включением соответствующей подсветки.

Символ — являет собой не опцию, а указание на то, что следующим аргументом будет имя файла. Без всего этого вполне можно обойтись, вызывая редактор просто

```
$lpe имя_файла
```

В соответствии со своей классовой принадлежностью, все действия по навигации и редактированию в lpe осуществляются посредством соответствующих командных клавиш. Правда, и такие клавиши, как стрелки управления курсором, <Home>, <End>, <Backspace>, <Delete> действуют обычным образом, хотя и дублируются некоторыми комбинациями.

Модель командных клавиш в lpe отличается от распространенных моделей в стиле vi или emacs, однако очень проста: все комбинации состоят из клавиши <Ctrl> и литерной, нечувствительной к регистру. Рассмотрим их по порядку.

- <Ctrl>+<Q> — перемещение курсора в начало строки (альтернатива <Home>).

- <Ctrl>+<W> — перемещение курсора в конец строки (альтернатива <End>).
- <Ctrl>+<R> — прокрутка на один экран вверх (альтернатива <PgUp>).
- <Ctrl>+<T> — прокрутка на один экран вниз (альтернатива <PgDn>).
- <Ctrl>+<O> — перемещение к предыдущему слову.
- <Ctrl>+<P> — перемещение к следующему слову.
- <Ctrl>+<K> — уничтожение текущей строки.
- <Ctrl>+<Y> или <Ctrl>+<U> — вставка уничтоженной (командой <Ctrl>+<K>) линии в позицию курсора.
- <Ctrl>+<S> — поиск последовательности символов в файле.
- <Ctrl>+<A> — повторный поиск.
- <Ctrl>+<F>–<Ctrl>+<O> — открытие нового файла с заменой (закрытием) текущего.
- <Ctrl>+<F>–<Ctrl>+<S> — запись текущего файла.
- <Ctrl>+<F>–<Ctrl>+<R> — считывание файла и его вставка в редактируемый документ в позиции курсора.
- <Ctrl>+<B>–<Ctrl>+<S> — установка языкового режима текущего файла.
- <Ctrl>+<B>–<Ctrl>+<T> — переключение между жесткой и мягкой табуляцией.
- <Ctrl>+<B>–<Ctrl>+<A> — включение/выключение автоматического отступа.
- <Ctrl>+<G>–<Ctrl>+<A> — переход к первой строке текущего документа.
- <Ctrl>+<G>–<Ctrl>+<S> — переход к последней строке текущего документа.
- <Ctrl>+<G>–<Ctrl>+<G> — переход к указанной строке текущего документа.
- <Ctrl>+<N>–<Ctrl>+<R> — ввод значения для повторяющейся команды.
- <Ctrl>+<N>–<Ctrl>+<T> — многократное повторение команды.
- <Ctrl>+<N>–<Ctrl>+<O> — начало или конец протоколирования макроса.
- <Ctrl>+<N>–<Ctrl>+<P> — исполнение последнего запротоколированного макроса.
- <Ctrl>+<V>–<Ctrl>+<V> — исполнение редактируемой строки как команды оболочки.
- <Ctrl>+<V>–<Ctrl>+<A> — исполнение редактируемой строки как команды `awk`.

- <Ctrl>+<V>–<Ctrl>+<S> — исполнение редактируемой строки как команды sed.
- <Ctrl>+<V>–<Ctrl>+<B> — исполнение нескольких строк как последовательности команд оболочки.
- <Ctrl>+<V>–<Ctrl>+<D> — исполнение нескольких строк как последовательности команд awk.
- <Ctrl>+<V>–<Ctrl>+<F> — исполнение нескольких строк как последовательности команд sed.
- <Ctrl>+<X> — выход из редактора с записью сделанных изменений.
- <INTERRUPT> — выход из редактора без записи изменений. Под <INTERRUPT> понимается комбинация прерывания для конкретного терминала. В нашем случае это будет <Ctrl>+<C>, тогда как <Ctrl>+<Z> обеспечит временный выход в командную строку оболочки.

Из рассмотрения командных комбинаций можно составить практически полное представление о возможностях lpe. Казалось бы, они не очень богаты, но вполне достаточны для несложной работы. Кроме того, в lpe обнаруживаются функции, не всегда имеющиеся и в более "развитых" редакторах, например, возможность протоколирования макросов и их воспроизведения, а также подключения внешних модулей языковой поддержки.

И потому сфера применения lpe — быстрая правка кода несложных html-страниц, создание пользовательских сценариев оболочки, правка конфигурационных файлов. Компактность и быстродействие, подсветка синтаксиса, возможность автоматизации типовых действий делает его более чем пригодным для этих целей.

vi и Vim

Консольный редактор vi (или какой-либо из его клонов) — непременный атрибут любой UNIX-системы, обычно вызываемый по умолчанию при общесистемных настройках. И потому любой пользователь FreeBSD должен иметь о нем представление, хотя для повседневной работы какой-либо иной редактор может оказаться более подходящим.

Поначалу vi может показаться порождением больного ума с садомазохистскими наклонностями. Однако достаточно осознать внутреннюю его логику — и начинаешь понимать, что более быстрого инструмента для обработки текста человеческий разум еще не придумал. А многочисленные возможности для его настройки (причем — выполняемые вполне элементарно) обеспечивают должную функциональность такой обработки. Хотя, с другой сто-

роны, создание нарративных текстов — не самая сильная его сторона. Однако к этому вопросу я еще вернусь.

Существует несколько редакторов, основанных на vi, включающих дополнительные возможности, но полностью совместимых с ним по системе базовых команд. И потому знание vi обеспечит возможность работы с любым из его клонов. Более того, если в некоей UNIX-системе вместо vi используется какой-либо из редакторов-клонов, например, Vim или elvis, в каталоге /usr/bin всегда будет представлен файл vi, являющийся жесткой или символической ссылкой на реальный исполнимый файл этого редактора. Поэтому vi или любой его аналог всегда может быть запущен командой

```
$vi имя_файла
```

хотя реально при этом запускается, например, Vim. Правда, в FreeBSD представлены как классический vi, входящий в базовый комплект системы, так и Vim (Vi IMproved), который дополнительно следует установить из коллекции пакетов или через систему портов. Именно о последнем и пойдет речь в этом разделе. Подчеркнем, что ниже термины vi, vim и Vim используются как синонимы, но везде имеется в виду Vim. Хотя практически все описанные приемы относятся также к классическому vi.

Редактор Vim может быть запущен из командной строки оболочки с именем файла в качестве аргумента или без такового. В первом случае открывается файл, если он существует, или создается новый — в текущем каталоге, или в каталоге, определенном в пути к нему. Например, команда

```
$vim.zshrc
```

вызовет для редактирования конфигурационный файл оболочки zsh из домашнего каталога пользователя, а команда

```
$vim ~/mytext/newtext.txt
```

создаст новый текстовый файл в подкаталоге ~/mytetxt домашнего каталога пользователя. Конечно, при этом каталог ~/mytetxt должен уже существовать, иначе при записи файла последует сообщение об ошибке.

Команда `vim` без имени файла откроет редактор Vim и выведет заставку следующего содержания:

```
~    VIM - Vi IMproved
~
~     version 6.1b BETA
~       by Bram Moolenaar et al.
~  Vim is open source and freely distributable
~
~  Help poor children in Uganda!
```

```
~  type  :help iccf<ENTER>       for information
~
~  type  :q<ENTER>               to exit
~  type  :help<ENTER>  or  <F1>  for on-line help
~  type  :help version6<ENTER>   for version info
```

которая сразу подсказывает направление дальнейших действий — можно либо немедленно начать работу (правда, как это сделать — я скажу чуть ниже), либо получить справку по использованию редактора.

Настоятельно рекомендуем воспользоваться этой возможностью, т. к. работа в Vim покажется весьма непривычной для пользователя, привыкшего к текстовым редакторам DOS/Windows. В частности, попытка немедленно начать ввод текста успехом не увенчается. И потому следует сначала ознакомиться с режимами работы vi.

В vi существует три основных (и принципиально различных) режима работы — командный, или визуальный (visual command mode), именуемый также нормальным, режим ввода (edit mode) и т. н. ex-режим, или режим построчного редактирования (ex mode или colon mode), который по-русски именуется еще и режимом ввода команд. Кроме того, в Vim (но не в классическом vi) дополнительно выделяется еще режим визуального выделения, или выбора.

Уже само по себе изобилие режимов способно запутать неподготовленного пользователя. А без четкого понимания различий между режимами никакая, даже самая элементарная, работа в vi попросту невозможна: широкое хождение имеют легенды о пользователях, полагающих единственным способом выхода из этого редактора перезапуск машины. И разнобой переводной русскоязычной терминологии только усугубляет дело. Однако, если рассмотреть эти режимы последовательно, все оказывается не так страшно.

Начнем с того режима, который включается по умолчанию при загрузке vi, почему далее я и буду именовать его нормальным — именно он обеспечивает быстроту перемещения по тексту и его обработки. К тому же из него осуществляется переход во все остальные режимы и возврат из них. В нормальном режиме нажатия клавиш не приводят к вводу символов, а интерпретируются как внутренние команды навигации и редактирования, привязанные к различным алфавитно-цифровым или символьным клавишам и их комбинациям. Например, нажатие клавиши <h> вызывает перемещение курсора на один символ влево, клавиши <l> — на один символ вправо, <k> — на строку вверх, <j> — на строку вниз и т. п.).

Предопределенные клавиши нормального режима чувствительны к регистру (и это имеет глубокий смысл, как будет показано чуть ниже) и последовательности нажатий. Так, повторное нажатие клавиши <d>—<d> отнюдь не эквивалентно по действию двум ее одиночным нажатиям. Кроме того, они

чувствительны и к раскладке клавиатуры. В частности, при включении кириллической раскладки никакие клавиши нормального режима при настройках по умолчанию просто не оказывают никакого действия. Впрочем, методы борьбы с этим существуют, о чем я расскажу в заключение раздела.

Естественно, создание текста в командном режиме невозможно. Для этого следует перейти во второй режим (будем называть его режимом ввода), для чего служат разнообразные команды (нормального режима!), например, <a> (от append) и <i> (от insert). Здесь нажатия клавиш приводят к вводу обычных буквенно-цифровых символов (после текущей позиции курсора в случае первой команды и перед ней — в случае второй), позволяя создавать новый текст или редактировать имеющийся. Хотя последнее более эффективно в командном режиме, возврат в который осуществляется клавишей <Escape>.

Для операций с документами (т. е. файлами) предназначен третий режим, за которым резонно закрепить название командного. Он вызывается из нормального режима командой <:> (символ двоеточия). Вводимые после этого последовательности символов (или отдельные символы) воспринимаются уже как команды, не привязанные к фиксированным клавишам, но опознаваемые по своим именам (или их аббревиатурам). Здесь возможны следующие действия:

- открытие существующего файла (`:edit имя_файла`, здесь и далее символ : указывает на принадлежность команды к командному режиму); если какой-либо файл перед этим уже загружен, он будет закрыт и замещен новым; если же изменения в нем не были сохранены — последует сообщение об ошибке;
- вставка существующего файла в позицию курсора (`:read имя_файла`);
- запись файла (`:write`), в том числе под другим именем (`:write имя_файла`);
- выход из редактора (`:quit`), происходящий, если текущий файл не изменялся или был предварительно сохранен;
- выход из редактора с предварительным сохранением измененного файла (`:xit` — от exit).

Одна из возможных причин путаницы между нормальным и командным режимами — то, что в последнем допустимы почти любые аббревиатуры указанных команд, вплоть до односимвольных (каковые в обиходе, как правило, и используются): `:e` — для открытия файла, `:r` — для его вставки, `:w` — для записи, `:q` и `:x` — для выхода. Однако это именно сокращения команд, а не клавиши, предопределенные для неких действий, как в нормальном режиме. Возможно совмещение команд командного режима, например, `:wq` — выход с предварительным сохранением измененного файла (что аналогично команде `:x`).

Команды отправляются на исполнение нажатием клавиши <Enter>, после чего происходит возврат в нормальный режим. Однако попытка, например, закрыть редактор без сохранения изменений в редактируемом документе (командой `:q`) или загрузить новый файл (командой `:e`), не сохранив предыдущий, вызовет сообщение об ошибке. Для принудительного выполнения таких действий команды `:q` и `:e` должны сопровождаться символом `!` (восклицательного знака) без пробела. Например, команда `:q!` закроет редактор vi, не сохранив изменений в текущем файле.

Действия командного режима частично дублируются в режиме нормальном. Так, в последнем для закрытия же файла (с предварительным сохранением изменений) используется комбинация <Z>–<Z> (двойное нажатие клавиши <Z> — регистр важен!), что является эквивалентом сочетания команд `:wq` командного режима.

Четвертый режим Vim назовем режимом выбора, поскольку он служит именно для выбора фрагментов текста с целью последующего применения к ним команд нормального режима. Переход в него (как обычно, из нормального режима) осуществляется нажатием клавиш <v> или <V>. В обоих случаях нажатия клавиш управления курсором приводят не к его перемещению, а к выделению блока, в первом — начиная с текущего положения курсора в пределах строки, во втором — начиная с начала маркированной курсором строки. Повторное нажатие тех же клавиш вызывает возврат в нормальный режим. Однако если нажать какую-либо клавишу, обеспечивающую действие в нормальном режиме, оно будет распространено на весь выделенный блок. Ну и после выполнения этого действия, естественно, происходит возврат в нормальный режим.

В современных версиях Vim предусмотрено нечто вроде индикации режимов — при включении режима ввода в нижней части экрана появляется надпись

```
--INSERT--
```

при переходе в режим выбора — надпись

```
--VISUAL--
```

Командный режим включается только по наборе символа : (предполагается, что пользователь этого не забудет сразу же), а нормальный режим, по замыслу разработчиков, на то и нормальный, что в индикации не нуждается. Впрочем, находясь в нормальном режиме, легко убедиться, что это он и есть. Для этого достаточно нажать клавишу <Escape> и услышать звуковой сигнал.

Такая система работы может показаться начинающему пользователю запутанной. Однако она имеет глубокое внутреннее обоснование. Редактор vi

создавался изначально как кроссплатформенный, который обязан работать на любых типах реальных и виртуальных терминалов. И потому все действия в нем можно осуществить, не покидая основной, алфавитно-цифровой, части клавиатуры, без обращения к дополнительным клавишам — стрелкам управления курсором, <Home>, <End>, <PageUp>, <PageDown>, <Insert>, <Delete>, <Backspace>.

Это, с одной стороны, обеспечивает быстроту и эффективность работы (правда, только при наличии доведенных до автоматизма навыков). С другой стороны, такая особенность vi позволяет не заботиться о настройках виртуальных терминалов в системе X Window, которые существенно влияют на поведение дополнительных клавиш: внутренние команды навигации и редактирования всегда будут интерпретироваться идентично.

Поскольку нормальный режим vi является не только основным, но и наиболее характерным для этого редактора (и — непривычным для пользователя!), имеет смысл ознакомиться с ним подробнее. Все изобилие команд vi (а их насчитывается много десятков) можно разделить на три группы:

- команды навигации;
- команды редактирования;
- команды перехода (в режим ввода).

Команды навигации служат для перемещения курсора по тексту. В консольном режиме FreeBSD для этого могут быть использованы и обычные клавиши управления курсором (стрелки, <PageUp>/<PageDown>, <Home>/<End>). Однако уже в различных программах эмуляции терминала в графическом режиме их поведение неоднозначно (и зависит от настроек), а в других UNIX-системах может быть иным. К тому же внутренние команды дают больше возможностей для навигации по тексту, чем клавиши стандартных клавиатур.

Так, в vi предопределены командные клавиши <h> и <l>, <k> и <j>, действие которых эквивалентно нажатиям клавиш <Left> и <Right>, <Up> и <Down> соответственно. Но, кроме того, с помощью парных команд <w> и <W> можно переместиться вперед, соответственно, на т. н. "маленькое" слово (т. е. отделенное пробелом или любым знаком препинания, символами - или +) и на "большое" (т. е. обязательно отделенное пробелом) слово. Пара команд <b> и <B> выполняет аналогичное перемещение назад, а команды <e> и <E> перемещают курсор в конец следующего "маленького" и "большого" слова.

Вообще, для многих команд vi характерно наличие парных эквивалентов — в нижнем и верхнем регистрах одной клавиши (<w> и <W>, <e> и <E>); действие второй команды из пары как бы расширяет действие первой.

Возможны также перемещения в предыдущее (клавиша <(>, соответствующая открывающей кавычке) и последующее (клавиша <)>, отвечающая закрывающей кавычке) предложения, в начало (<H>) и конец (<L>) экрана, в начало (<0> — ноль) и конец (<$>) строки и т. п. — список навигационных команд приближается к 30. Иными словами, нажатием одной клавиши или, в крайнем случае, двухклавишной комбинации (<Ctrl>+<f> — на следующую экранную страницу, <Ctrl>+<b> — на предыдущую) можно переместиться в абсолютно любое заранее определенное место текущего документа.

В дополнение к этому, команды навигации vi могут использоваться с численными аргументами. Например, команда <5>−<h> переместит курсор на 5 символов влево (считая символ в позиции курсора), а команда <3>−<k> — на три строки вверх.

Далее, для навигации по тексту могут использоваться клавиши <−> (минус) для перемещения на одну строку вверх и <+> (плюс) для перемещения на одну строку вниз. Особенно эффективны они в сочетании с численными аргументами: командой <7>−<+> возможно перемещение на седьмую строку вперед, а командой <13>−<-> — на тринадцатую строку назад (в обоих случаях — включая строку в позиции курсора).

Команды редактирования предназначены для модификации существующего текста без перехода в режим ввода. Конечно, и в последнем возможно удаление и замена символов стандартными клавишами <Delete> или <Backspace>, но, как и в случае с навигацией, возможности командного режима в этом отношении много шире.

Так, наряду с удалением единичного символа в позиции курсора (<x>) или перед ней (<X>), возможно удаление слова (<d>−<w>), строки (<d>−<d>) или ее части перед (<d>−<D>) или после (<d>−<G>) курсора, предложения (<d>−<)>) или абзаца (<d>−<}>).

Как и навигационные команды, команды редактирования могут использоваться с численными аргументами. Так, команда <5>−<d>−<d> удалит текущую строку и еще четыре строки вниз. А с помощью команды <3>−<d>−<w> можно удалить три слова подряд (включая то, на котором находится курсор).

Не меньше команд отвечает за копирование фрагментов, их вставку и замену. Например, команда <y>−<w> копирует "маленькое" слово, <y>−<W> — "большое", <y>−<y> — строку, <y>−<)> — предложение, <y>−<}> — абзац. Командой же <p> удаленный или скопированный фрагмент вставляется в позицию курсора.

Действие ошибочно введенных команд редактирования может быть отменено командой <u> (от undo). Вторичный ввод этой команды приведет к отмене предыдущего действия, и т. д. Для возврата ошибочно отмененной операции используется команда <Ctrl>+<R>.

Описание всех команд редактирования заняло бы слишком много места. Подчеркну лишь, что, как и в случае с навигацией, нажатием одной-двух клавиш можно удалить, скопировать, вставить и переместить текстовый блок практически любого размера — от единичного символа до произвольного их количества (строки, предложения, абзаца, экранной страницы).

Команды перехода могут рассматриваться как подмножество команд редактирования нормального режима, после которых возможен ввод новых символов и их последовательностей. Кроме уже упомянутых <a> и <i> (ввод после курсора и в его позиции, соответственно) к ним относятся:

- <A> — ввод текста в конец строки;
- <I> — ввод в начало строки;
- <o> — создание новой строки под текущей с возможностью ввода в нее текста;
- <O> — создание новой строки над текущей, в которую также можно ввести текст.

Как и большинство прочих команд редактора vi, команды перехода могут использоваться с числовыми аргументами. Так, если дать команду <3>–<a> и после этого ввести некоторую последовательность символов, по выходе в командный режим (клавишей <Escape>) она будет повторена трижды. Аналогично, ввод текста после команды <#>–<O> даст # идентичных строк.

Редактор vi располагает средствами поиска и замены текстовых фрагментов, в том числе и с использованием регулярных выражений. Для этого предназначена команда командного режима `:s` (или `:substitute`). Она дается в форме

```
:#s/text1/text2/опция
```

где `#` — количество строк, в которых операции поиска и замены должны осуществляться (без указания его действие команды распространяется только на текущую строку). Если поиск и замену необходимо выполнить по всему тексту документа, дается команда `:%s`.

Следует подчеркнуть, что если не указать заменяющего текста (и опций замены), все текстовые фрагменты, указанные в качестве заменяемого текста, будут просто удалены. Во избежание этого используются опции команды `:s`. Так, опция `/c` предписывает запрос подтверждения на замену для каждого найденного фрагмента.

И поиск, и замена в vi возможны только для последовательности символов, составляющих одну строку (т. е. не содержащей *символа возврата каретки* — LF). Заменяющая последовательность символов также должна образовывать единую строку.

Описанным не исчерпываются возможности редактора vi. В частности, он (вернее, его клон Vim) поддерживает язык макрокоманд и допускает их протоколирование. В комплекте с редактором (в каталоге /usr/local/share/vim/vim###) имеется большое количество таких макросов. Там же — и детальная документация по их применению.

Функциональность редактора Vim в значительной мере зависит от его настроек. Пример конфигурационного файла (vimrc_example.vim) можно обнаружить в том же каталоге (/usr/local/share/vim/vim###). Его следует скопировать в свой домашний каталог под именем ~/.vimrc и отредактировать по потребностям.

В частности, именно в этом файле можно обеспечить работу клавиш нормального режима при кириллической раскладке клавиатуры. Для этого в него вносится строка

```
set langmap=
```

в которой далее перечисляются все символы кириллической раскладки (и в верхнем, и в нижнем регистрах) попарно с соответствующими символами раскладки латинской, разделяя пары запятой без пробела:

```
set langmap=ё,Ё~,йq,... Б<,ю.,Ю>
```

Правда, чтобы это сработало, Vim должен быть собран с поддержкой опции `langmap`. Вариант же, распространяемый с FreeBSD в качестве бинарного пакета, таковой не поддерживает. И потому для полноценного использования Vim в русскоязычном окружении его следует устанавливать из коллекции портов с внесением соответствующей правки перед компиляцией.

А вообще редактированием файла ~/.vimrc Vim может быть адаптирован для задач любого рода, например, работы с html-документами. В Интернете имеется большое количество примеров конфигурационных файлов такого рода. Некоторые из ссылок приведены в разделе о дополнительных источниках.

Редактор jed

Это — весьма своеобразный редактор, занимающий промежуточное положение между чисто командными и меню-ориентированными редакторами. Первый его запуск создает впечатление, что он объединил в себе все мыслимые недостатки обоих семейств. Однако более углубленное рассмотрение показывает, что это не так.

Начать с того, что jed способен эмулировать такие распространенные программы для обработки текста, как emacs, wordstar, Borland IDE (текстовый редактор из интегрированных сред разработки фирмы Borland) и некоторые

другие. Далее, он обеспечивает подсветку синтаксиса, имеет собственный макроязык и многочисленные возможности настройки. Последние осуществляются редактированием файла /usr/local/lib/jed/lib/jed.rc, который надлежит скопировать в свой домашний каталог под именем ~/.jedrc. Ну и возможность выбора между работой в командном режиме и действиями через меню обеспечивают ему достаточную гибкость.

Редактор le

Текстовый редактор le — программа производства нашего соотечественника, Александра Лукьянова. Конечно, это — не специфический инструмент FreeBSD. Однако в FreeBSD редактор этот прочно входит как в качестве порта, так и (в большинстве версий) пакета, тогда как, скажем, в дистрибутивах Linux он мне не попадался.

Запуск le (одноименной командой) в первый раз вызывает стандартное сообщение о свободном характере программы, отсутствии всяческих гарантий и прочего. С одновременным предложением ввести имя загружаемого файла. Буде задать имя файла отсутствующего, об этом сообщат, предложив заодно и создать такой файл. Ну, а для открытия существующего файла необходимо указать полный путь к нему, причем дополнение клавишей табуляции не работает.

После этого перед глазами возникает абсолютно пустой синий экран, где можно тут же начинать вводить текст. Правда, что делать с ним дальше, остается неясным, если только клавишей <F10> не активизировать меню. Через которое и производятся все действия. Хотя ряд из них сдублирован и клавишными комбинациями.

Поскольку обзор пунктов меню может дать практически полное представление о возможностях le, с него-то мы и начнем. Пункты эти следующие:

- **File** — Файл;
- **Block** — Блок;
- **Search** — Поиск;
- **Move** — Переместить;
- **Format** — Форматировать;
- **Others** — Прочее;
- **Options** — Опции;
- **Help** — Помощь.

Через пункт **File** (Файл) доступны следующие действия: открытие (**Load** (Считать), <F3>) и сохранение (**Save** (Сохранить), <F2>) файла, сохранение его под другим именем (**Save as...** (Сохранить как...), <Escape>–<F2>), пере-

ключение между открытыми файлами (<Escape>–<F3>), каковых может быть сколько угодно много, считывание открытого файла заново (**Reopen**), получение информации о файле (**Info**) — размер, число строк, полный путь до файла, владелец и т. п.

В пункте же **File** (Файл) имеются такие опции, как построение (**Make**), запуск (**Run**) и компиляция (**Compile**) программы, временный выход в командную среду (**Shell**) и приостановка работы редактора (**Suspend**).

Пункт **Block** (Блок) посвящен операциями с фрагментами текста. Здесь можно установить начало (<F5>) и конец (<F6>) выделяемого фрагмента, который затем может быть скопирован (**Copy** (Копировать), <F11>), перемещен (**Move** (Переместить), <F12>), удален (**Delete** (Удалить)<F4>–<D>), записан в виде файла (**Write** (Записать), <F4>–<W>). Здесь же в текущий файл может быт вставлен и иной документ (**Read** (Прочитать), <F4>–<R>).

Кроме того, над выделенными блоками доступны такие действия, как установка отступа, смена нижнего регистра на верхний и наоборот, а также конвертация регистров. Однако все это возможно только по отношению к фрагментам, выделенным с клавиатуры (клавишами <F5> и <F6>); фрагменты, выделенные мышью, могут быть только скопированы (щелчком средней ее клавиши) в текущую позицию курсора.

На которую, кстати говоря, перемещение мыши никакого влияния не оказывает — она может быть изменена только с помощью клавиатуры — стрелками управления курсором, клавишами <Home> (перемещение в начало строки), <End> (перемещение в конец строки), <PageUp> и <PageDown> (пролистывание на один экран назад и вперед соответственно).

В сочетании с клавишей <Escape> клавиши перемещения курсора оказывают более сложное действие. Так, <Escape>–<Left> (<Escape>–<Right>) перемещает курсор на одно слово влево (или, соответственно, вправо), <Escape>–<Home> (<Escape>–<End>) — в начало (конец) документа.

Пункт **Search** (Поиск), как нетрудно догадаться, посвящен поиску и замене внутри документа. Поиск возможен вперед (<Ctrl>+<F> или <F7>), назад (<Ctrl>+<B> или <Escape>–<Ctrl>+<F7>; при этом внизу экрана появляется строка с предложением ввести искомое выражение. Возможна, естественно, и замена (<Ctrl>+<R>), при этом сначала предлагается ввести заменяемое выражение, затем — выражение-заменитель.

Пункт **Move** (Переместить) — это перемещения по документу: к строке по ее номеру, в начало или конец файла или отмеченного блока, к последнему редактировавшемуся участку и т. д.

Пункт **Format** (Форматирование) позволяет отформатировать абзац (т. е. установить в нем длину строки в соответствии с принятой границей переноса слов, о чем чуть ниже) или весь текст (причем без возможности отмены), центрировать текущую строку или выровнять ее по правому краю.

В пункте **Others** (Прочее) собраны всякие дополнительные опции, как то:

- калькулятор;
- рисование таблиц с помощью символов псевдографики;
- вставка специальных символов из кодовой таблицы в позицию курсора (**Character Set**); очень полезная возможность, когда нужно вставить коды символов или esc-последовательности;
- конвертация текстового файла UNIX (использующего LF в качестве символа конца строки) в формат DOS (где таковым выступает LF+CR).

Пункт **Options** (Опции) посвящен настройкам редактора. А пункт **Help** (Помощь) — это, естественно, помощь, с которой я настоятельно советую ознакомиться. Здесь, в разделе **Help on keys** (Подсказка по ключам) (вызываемом традиционной для DOS/Windows клавишей <F1>), приведены все клавишные комбинации для действий, которые затруднительно было бы отыскать в меню. Так, только из **Help** (Помощь) удается узнать, как отменить неправильное действие — для этого служит комбинация <Ctrl>+<U>. А раздел **Help on word** (Подсказка по словам) по замыслу позволяет, установив курсор на какое-либо слово (например, имя команды), получить по нему некую справку, что достигается вызовом стандартной man-страницы.

В заключение — об особенностях работы с русским языком. При правильной русификации консоли никаких проблем со вводом кириллических символов и отображением их на экране не возникает. Более того, возможно переключение экранных кодировок — с альтернативной (CP866) на KOI8 и обратно, в зависимости от того, требуется ли отображение символов псевдографики или нет.

Кроме того, le позволяет переключаться с латиницы на кириллицу, используя меню **Options** (Опции) — **Editor** (Редактор) или комбинации клавиш <Ctrl>+<A>–<R>, хотя это не очень удобно и срабатывает не всегда. Кроме того, при кириллической раскладке редактор нормально реагирует на нажатие функциональных клавиш. Использование буквенных комбинаций требует дополнительного нажатия клавиши <Ctrl>: например, <F4>–<Ctrl>+<D> — для удаления блока или символа, <F4>–<Ctrl>+<P> — для конвертации в верхний регистр и т. д.

Дополнительные источники

Текстовым редакторам традиционно уделяется много внимания в любой книге, посвященной UNIX и UNIX-подобным операционным системам. В частности, обзор программ этого рода можно найти в книге: Федорчук А. Офис, графика, Web в Linux. — СПб.: БХВ-Петербург, 2001. Не смотря на

название, все описанные там редакторы работают и под FreeBSD. Хотя, кажусь, мнение о многих рассмотренных там редакторах я существенно изменил — и обычно в лучшую сторону.

Описанию редактора Vim, его возможностям и настройкам посвящена серия статей Владимира Попова:

- Vim. Быть может он действительно — лучший? — **http://www.softerra.ru/review/oses/linux/12402/**;
- Vim-2 или "что может быть проще?" — **http://www.softerra.ru/review/oses/linux/12607/**;
- Vim: уже шесть! — **http://www.softerra.ru/freeos/15764/**.

На многих сайтах (например, **http://linux.perm.ru**) можно обнаружить русский перевод документа под названием Vim Color Editor HOW-TO, который немало поспособствует в освоении этого редактора.

И, наконец, на сайте **http://www.vim.org** можно найти множество примеров настройки Vim под конкретные задачи. А на сайте **http://www.math.fu-berlin.de/~guckes/sven/** имеется файл vimrc.forall, созданный Свеном Гуккесом (Sven Guckes) и представляющий собой универсальный конфигуратор этого редактора практически на все случаи жизни.

Глава 18

Текстовый редактор joe

Текстовый редактор joe — типичный представитель консольных редакторов командного стиля, т. е. ориентированных не на действие через меню, а на управление с помощью прямых директив. Название его можно перевести примерно как "редактор дядюшки Джо". Он создан Джозефом Алленом (Joseph H. Allen) при участии Ларри Форда (Larry Foard) и Гари Грея (Gary Gray). Это — открытая и бесплатная программа, доступная в исходниках на сервере **ftp://ftp.std.com**. Она реализована для всех, насколько я знаю, UNIX-подобных систем. В поставку FreeBSD она входит в качестве порта и пакета, и представлена (в текущих ветках системы) двумя версиями — joe 2.8.x и joe-devel (2.9.x). Вторая из них обладает некоторыми дополнительными возможностями (в частности, улучшенной работой с кириллицей), но в некоторых версиях FreeBSD может работать нестабильно. Однако к моменту выхода этой книги, думаю, это будет исправлено, и потому ниже речь будет идти именно о последней.

Обзор возможностей

Запускается joe одноименной командой, можно — с именем файла, предназначенного для редактирования. В случае, если этого имени в природе не имеется, создается новый файл. Кроме этого, при запуске joe можно использовать ряд опций командной строки. Представление о них дает чтение страниц экранной документации (man joe).

Сразу после запуска joe выглядит весьма непрезентабельно: черный экран со светло-серым текстом — и все. Что делать дальше — остается пока неясным. Единственно, строка заголовка в верхней части экрана гласит, что с помощью комбинации клавиш <Ctrl>+<K>−<H> можно вывести на экран систему помощи. И далее пролистывать ее с помощью <Escape>−<.> (точка) — вперед или <Escape>−<,> (запятая) — назад.

Так вот, внимательное знакомство с системой помощи дает представление о возможностях редактора. Каковые неожиданно оказываются весьма богатыми.

Думаю, понятно, что текстовый редактор позволяет вводить текст (в том числе и кириллический) и обеспечивает навигацию по нему. Последняя осуществляется двумя способами: либо с помощью собственных клавишных комбинаций (как правило, это <Ctrl>+<литера>), о чем подробнее — в следующем разделе, либо — с помощью стандартных клавиш управления курсором (стрелок, <PageUp>, <PageDown>, <End>, <Home>). При этом клавиши эти введут себя обычным (с точки зрения пользователя DOS/Windows) образом, что отнюдь не само собой разумеющееся для консольных UNIX-редакторов). Исключение — клавиша <End>, которая на некоторых типах терминалов и в большинстве эмуляторов терминала графического режима работать отказывается.

Вообще, следует заметить, что основным средством навигации по тексту в joe являются именно собственные команды, а не клавиши управления курсором. Во-первых, при наличии некоторого навыка, они обеспечивают большую скорость работы за счет того, что не требуется перемещения пальцев за пределы основной клавиатуры (поверьте, это действительно быстрее!). Во-вторых, команды joe, вызываемые клавишными комбинациями, функционируют абсолютно одинаково в любых консолях (по крайней мере за Linux-консоль, syscons FreeBSD, pcvt из OpenBSD я ручаюсь — проверено лично) и эмуляторах терминалов. Тогда как для клавиш управления курсором возможно, как уже отмечено, аномальное поведение.

Мышь в joe поддерживается стандартным для UNIX-консоли образом. То есть она выступает не как указательное устройство, а как средство выделения и копирования текстовых блоков. Это относится как к текстовой консоли, так и к эмуляторам терминалов графического режима.

Разумеется, в соответствии со своим званием, joe позволяет и редактирование текстов, т. е.: выделение блоков и отдельных знаков, их копирование, перемещение и удаление, форматирование абзацев (центрирование, лево- и правостороннее выравнивание и т. п.), вставку существующих файлов в текущий документ и запись выделенных фрагментов в виде отдельных файлов.

Редактор joe имеет функцию многоуровневой отмены и возврата отмененных операций. Встроенной проверки орфографии нет, однако можно подключить внешнюю программу (такую, как ispell), в том числе и для русскоязычных текстов. Имеются достаточно развитые средства поиска и замены, в том числе с использованием шаблонов и регулярных выражений. Есть возможность создания закладок (Bookmarks) и перехода к ним, что необходимо при редактировании длинных структурированных документов.

В joe имитируется многооконный режим: поле текущего документа может быть разбито пополам, и далее каждое из них также может делиться сколь угодно дробно (правда, только по горизонтали). Обеспечена также одновременная работа со многими документами, каждый из которых может быть выведен в оконном или полноэкранном виде.

Количество одновременно открытых файлов, насколько мне удалось понять, не ограничено ничем. Более того, по сведениям моих корреспондентов, joe позволяет работать с документами, объем которых превышает объем ВСЕЙ доступной (т. е. физической + виртуальной) памяти! Правда, проверить это мне не удалось...

В joe поддерживается собственный макроязык с достаточно прозрачным синтаксисом. Кроме того, имеется режим протоколирования макросов, что позволяет быстро наращивать его возможности.

И вообще, joe — очень настраиваемый редактор. Во-первых, имеется система интерактивной настройки ряда параметров, таких как перенос слов, абзацный отступ и т. п. Правда, установки эти действуют только в текущем сеансе. Для перманентных изменений необходимо редактирование конфигурационного файла. Однако здесь, учитывая возможность встраивания макрокоманд, возможности настроек становятся поистине безграничными.

И наконец, что немаловажно в наших условиях, joe корректно работает с кириллицей. При корректно русифицированной консоли не возникает никаких проблем ни с выводом, ни с вводом символов кириллицы. Более того, все клавишные комбинации работают и при латинской, и при русской раскладке клавиатуры. Правда, в последнем случае часто требуется дополнительное нажатие управляющей клавиши.

Иными словами, joe — вполне развитый и полнофункциональный текстовый редактор общего назначения. Он в раной мере пригоден для эпизодической работы по написанию скриптов, правке конфигурационных файлов и т. п., и для систематического использования при создании длинных структурированных текстов нарративного характера. А наличие языка макрокоманд допускает эффективно применять его и в специальных целях — для разметки html-страниц, верстки документов в TeX, не говоря уже о собственно программировании. Единственный недостаток joe в этом плане — отсутствие подсветки синтаксиса как в консоли, так и в эмуляторе терминала.

Система помощи

Ознакомиться практически со всеми возможностями редактора joe можно посредством его системы помощи. Как уже говорилось, она выводится на экран нажатием комбинации клавиш <Ctrl>+<K>−<H> и насчитывает семь секций, каждая из которых занимает собственный экран, перемещение между которыми осуществляется комбинациями клавиш <Meta>+<.> (вперед) и <Meta>+<,> (назад).

Первая секция, **Basic**, (рис. 18.1) описывает действия наиболее общего плана: перемещения курсора (субсекция **CURSOR**), переходы по тексту (субсекция **GO TO**), операции с текстовыми блоками (субсекция **BLOCK**),

команды удаления символов и текстовых фрагментов (субсекция **DELETE**), команды поиска, проверки орфографии, форматирования (субсекции **SEARCH, SPELL, MISC**), операции с файлами (субсекция **FILE**), а также выход из редактора.

Рис. 18.1. Система помощи редактора joe: первая, базовая, секция

Вторая секция посвящена описанию манипуляций с окнами — расщеплению (Split) экрана, скрытию и показу открытых окон, переходу между окнами, изменению их размера (рис. 18.2).

Рис. 18.2. Система помощи joe: операции с окнами

В третьей секции (рис. 18.3) собрано описание расширенных возможностей для редактирования текстов, как то:

- работа по протоколированию, записи и воспроизведению макросов (субсекция **MACROS**);
- команды для прокрутки текста (субсекция **SCROLL**);
- команды для взаимодействия с оболочкой (субсекция **SHELL**);
- средства установки закладок, наращиваемого поиска, ввода спецсимволов (секции **BOOKMARKS, I-SEARCH, QUOTE**) и т. д.

Четвертая секция — расширенные возможности для программистов (команды перехода к регулярным выражениям, компилирования и отладки). В пятой секции дано описание сложных регулярных выражений. И, наконец, шестая секция — операции с командной строкой встроенной в редактор командной оболочки (рис. 18.4).

Рис. 18.3. Система помощи joe: расширенные возможности

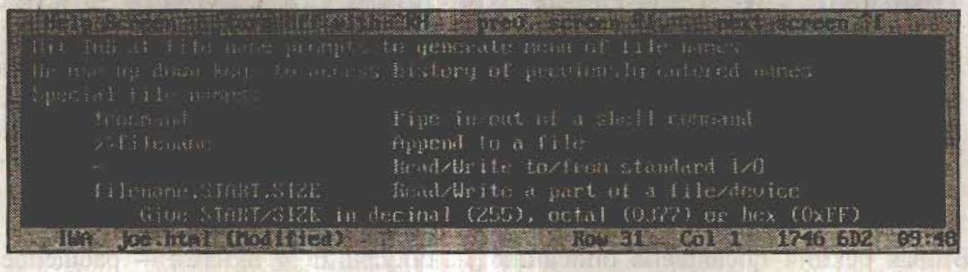


Рис. 18.4. Система помощи joe: операции с командной строкой встроенной оболочки

Таковы возможности справочной системы joe по умолчанию. Как будет показано ниже, пользователь может не только изменять имеющиеся секции, но и создавать собственные, в любом количестве и для любых целей.

Если мне удалось убедить читателя, что joe — вещь стоящая, имеет смысл подробнее ознакомиться с его характерными особенностями.

Характерные особенности

Для эффективного использования joe следует четко уяснить, что это — командный редактор в чистом виде. То есть все действия по редактированию текста осуществляются соответствующими встроенными командами, к которым привязаны комбинации клавиш. В сущности, как будет показано ниже, это — макросы на собственном языке joe. Из чего следует, что, с одной стороны, система команд может быть сколь угодно наращена, с другой — что клавишные комбинации для них могут быть переопределены произвольным образом.

Впрочем, необходимости в последнем я не вижу. Структура предопределенных по умолчанию клавишных комбинаций проста и очень логична. За самыми простыми и частыми действиями для навигации и редактирования закреплены двухклавишные комбинации. Это, как правило, сочетание одновременно нажатых клавиш <Ctrl> и буквенной. Последняя имеет, обычно, мнемонический смысл, хотя и не всегда прозрачный.

Полный список встроенных команд и привязанных к ним клавишных комбинаций дан в заключительном разделе этой главы. Здесь же я приведу только основные примеры. Так, комбинация <Ctrl>+<B> (от backward) перемещает курсор на один знак влево, <Ctrl>+<F> (от forward) — на один знак вправо, <Ctrl>+<Z> — переход к предыдущему слову, <Ctrl>+<X> — к последующему слову, и т. д.

В некоторых случаях в качестве управляющей клавиши используется клавиша <Meta> (напомню, что ее эквивалент — нажатие и отпускание клавиши <Escape>): если в комбинации с ней набрать литеру <W> — курсор переместится на строку вверх, литеру <Z> — на строку вниз. Кроме того, <Meta> служит для вызова проверки правописания для слова (<Meta>+<N>) и всего файла (<Meta>+<L>). Нажатие клавиши <Meta> два раза подряд приводит к установке закладки (bookmark), которая маркируется произвольной цифрой, а <Meta>+<#> (где # — эта самая цифра) вызывает переход к установленной закладке. Правда, очевидно, что закладок не может быть больше 10; и к тому же по завершении сеанса они не сохраняются.

Все клавишные комбинации в joe не чувствительны к регистру, причем не только для буквенных, но и символьных клавиш. Так, для отмены последней операции (как уже говорилось, многоуровневой) зарезервирована комбинация <Ctrl>+<_> (знак подчеркивания), а для возврата отмененного действия — <Ctrl>+<^>; однако в первом случае работает также комбинация <Ctrl>+<-> (дефис или минус), во втором — <Ctrl>+<6>.

Кроме того, все двухклавишные комбинации не чувствительны и к раскладке клавиатуры: сочетание клавиш, например, <Ctrl>+<T> (латинское) будет вызывать систему настройки joe и при кириллической раскладке. Интересно, что для пролистывания страниц помощи (показанных, например, на рис. 18.1—18.4) вперед и назад при кириллической раскладке следует нажимать <Escape> и точку (или, соответственно, запятую) также в ее положении на русифицированной клавиатуре (т. е. в нижнем правом углу для Windows-клавиатур и на верхнем регистре цифр 5 и 7, если мне не изменяет память, — для клавиатур с DOS-маркировкой).

Следует заметить, что некоторые команды навигации по тексту, связанные с перемещением по словам и удалением слов (такие, как <Ctrl>+<X>, <Ctrl>+<Z>, <Ctrl>+<W> и <Ctrl>+<O>), обнаруживают аномалию при применении к русскоязычным текстам. В версии joe 2.8.x клавиатуры они оказывают то же действие, что и комбинации смещения и удаления отдельных символов — <Ctrl>+<F>, <Ctrl>+<B>, <Ctrl>+<D> и <Ctrl>+<O>. Это связано с тем, что joe не в состоянии определить границы слов, набранных символами, отличными от латиницы. В версии 2.9.x это исправлено, и почти все такие комбинации клавиш манипулируют целыми кириллическими словами, за исключением комбинации <Ctrl>+<W> — она по-прежнему удаляет лишь единичный символ в позиции курсора.

Для более сложных или редких действий используются трехклавишные комбинации. Это почти исключительно одновременно нажатые <Ctrl>+<K>, после чего нажимается литерная клавиша. Так, операции с блоками осуществляются следующим образом:

- <Ctrl>+<K>–<B> — отмечает начало выделяемого блока;
- <Ctrl>+<K>–<K> — его конец;
- <Ctrl>+<K>–<C> — копирует;
- <Ctrl>+<K>–<M> — перемещает выделенный блок в позицию курсора

и т. д.

Трехклавишные комбинации также не чувствительны к регистру. И работают также и при кириллической раскладке клавиатуры. В этом случае только необходимо нажимать вторую литерную клавишу вместе с той же клавишей <Ctrl>: т. е. запись текущего файла при включении кириллической раскладки потребует комбинации <Ctrl>+<K>–<Ctrl>+<D>, вызов нового файла — <Ctrl>+<K>–<Ctrl>+<E> и т. д.

В `joe` нет отдельной функции для переименования файла. Но при любой записи текущего документа следует запрос на подтверждение имени файла, изменить которое при этом никто не запрещает. Следует только помнить, что дальнейшая работа происходит с исходным, а не переименованным файлом.

Кроме того, в joe доступны еще некоторые действия с файлами. Так, комбинация <Ctrl>+<K>–<R> вставляет текст из существующего файла в позицию курсора, <Ctrl>+<K>–<W> — записывает выделенный блок в виде нового файла (разумеется, запросив предварительно его имя). С помощью комбинации <Ctrl>+<K>–<E> можно открыть для редактирования другой существующий файл. При этом следует предложение ввести путь и имя, причем и для того, и для другого работает режим дополнения клавиш табуляции, как в командных средах bash или tcsh.

Между одновременно открытыми файлами возможен обмен данными: или с помощью команд joe (т. е. выделением блока в первом файле и его копированием или перемещением во второй), или с помощью мыши — стандартным выделением и вставкой щелчком средней ее кнопки. Второй способ, естественно, может применяться и для обмена между разными копиями joe, запущенными на отдельных виртуальных консолях.

Одновременно открытые файлы могут быть представлены как в полноэкранном виде, так и каждый в своем окне. Для переключения между одноооконным и многооконным режимами служит комбинация <Ctrl>+<K>–<I>. Размер каждого из выведенных окон может быть увеличен или уменьшен (<Ctrl>+<K>–<G> и <Ctrl>+<K>–<T> соответственно), правда, только с шагом в одну экранную строку. Переключение между открытыми документами, вне зависимости от режима, осуществляется комбинациями <Ctrl>+ +<K>–<N> (вперед или вниз) и <Ctrl>+<K>–<P> (назад или вверх).

К слову сказать, в joe возможен и независимый просмотр разных частей документа в отдельных окнах, для чего предназначена функция расщепления окна (<Ctrl>+<K>−<O>). Ну и, конечно же, фрагменты из одной части файла могут быть легко перенесены в другую.

Универсальной комбинацией для окончания любой операции в joe является <Ctrl>+<C>. С ее помощью закрывается окно с текущим документом; если он был единственным в данном сеансе, одновременно происходит и выход из редактора. В обоих случаях следует запрос на сохранение изменений, буде таковые имелись. Отказаться от выхода или закрытия файла можно повторным нажатием той же комбинации <Ctrl>+<C>. Она же используется для прекращения любой длящейся во времени (спеллинг, поиск) или требующей подтверждения операции.

Кроме этого, непосредственно из joe, без выхода, можно обращаться к командам оболочки (Shell), причем — различными способами. Так, комбинация <Ctrl>+<K>−<Z> обеспечивает временный выход в оболочку, где можно вводить любые ее команды. А по завершении операций — вернуться в среду joe можно командой `fg`. То есть в данном случае мы имеем дело с обычной приостановкой текущей задачи.

Кроме этого, есть и более интересная возможность: открытие внутри joe, посредством комбинации <Ctrl>+<K>−<'> (апостроф), самостоятельного окна с собственной командной оболочкой. Здесь можно выполнять любые команды с выводом их результатов на экран. После чего стандартной командой `exit` осуществляется выход из среды, а все результаты сохраняются обычным для joe образом в виде текстового файла: возможность неоценимая при создании и редактировании всякого рода скриптов.

Следует подчеркнуть, что в данном случае запускается именно встроенная в joe оболочка, не имеющая никакого отношения к пользовательской (login Shell). Она не порождает нового процесса (в чем легко убедиться командой `ps`) и не наследует никаких переменных окружения от пользовательской оболочки. Интерактивные ее возможности достаточно слабы (не поддерживаются ни автодополнение, ни история команд). Однако она дает возможность интеграции возможностей joe по редактированию и команд оболочки для обработки текста.

Единичная команда оболочки может быть выполнена после нажатия комбинации клавиш <Meta>+<!>. В этом случае внизу экрана появляется приглашение командной строки в форме:

```
Program to run:
```

представляющее собой разновидность встроенной оболочки joe.

Наконец, в joe обеспечивается ввод специальных символов. Так, нажатие клавиши <`> (обратный апостроф) приводит к предложению ввести первый знак десятичного (0-9), шестнадцатеричного (x) или восьмеричного (o) кода

символа. Если же вместо кода нажать <Escape> — можно ввести любую esc-последовательность. То же самое можно сделать и после нажатия комбинации <Ctrl>+<\>.

Специальные символы могут вводиться не только непосредственно в редактируемом тексте, но и в строке поиска. Это позволяет легко находить и глобально заменять лишние разрывы строк и тому подобные нарушения структуры.

Таковы основные возможности joe для редактирования текстов общего характера. Кроме этого, имеется ряд команд специального назначения, используемых при программировании (поиск ошибок, компилирование и пр.), однако их я за ненадобностью (мне) не изучал. Однако и сказанного достаточно для представления возможностей редактора joe.

Макрокоманды

Если же штатных команд редактора joe оказывается недостаточно, можно прибегнуть к их самостоятельному конструированию. Для чего предназначен внутренний язык макрокоманд. К сожалению, он не описан в экранной документации. Однако получить представление о его синтаксисе и возможностях можно, изучив внимательно конфигурационный файл joerc (о чем подробнее — в следующем разделе).

Кроме того, есть еще один простой и эффективный способ изучения макроязыка, совмещающий приятное с полезным, — режим протоколирования макрокоманд (см. рис. 18.3). Включается он комбинацией клавиш <Ctrl>+<K>−<[> (открывающая квадратная скобка), вслед за чем следует ввести номер макроса (от 0 до 9), выступающий как в качестве его имени, так и в роли запускающей клавиши. Далее просто выполняются необходимые действия (например, вводится требуемый тэг HTML), после чего запись макроса останавливается комбинацией <Ctrl>+<K>−<]> (закрывающая квадратная скобка). Для воспроизведения запротоколированного макроса используется комбинация <Ctrl>+<K>−<#> (где # — указанный при записи номер макрокоманды).

С помощью протоколирования макросов можно автоматизировать ввод наиболее нужных для конкретной задачи символов и их наборов, не предусмотренных штатным образом. Например, основных тэгов HTML для разметки Web-страниц, таких как параграф, разрыв строки, заголовки нескольких уровней, таблицы и списки. А заодно — и изучить синтаксис языка. Поскольку в joe предусмотрена возможность помещения запротоколированных в данном сеансе макросов в тело существующего или нового документа (комбинацией клавиш <Meta>+<D).

Например, записанные мной макросы для ввода HTML-тэгов выглядят следующим образом:

```
General Structure
insf,"~/blank.html",rtn  ^K.k1    HTML Page
"<H1></H1>",ltarw,ltarw,ltarw,ltarw,ltarw      .k1     H1
rtn,rtn,"<H2></H2>",ltarw,ltarw,ltarw,ltarw,ltarw      .k2     H2
rtn,rtn,"\x09","<H3></H3>",ltarw,...,ltarw       .k3     H3
rtn,rtn,"\x09","\x09","<H4></H4>",ltarw,...,ltarw.k4     H4

Body Text
rtn,rtn,"<P>","</P>",ltarw,ltarw,ltarw,ltarw   .k5     Paragraph
rtn,"<BR>"      .k6  Break
rtn,rtn,"<PRE>",rtn,rtn,"</PRE>",uparw .k7     Preformat
"<STRONG></STRONG>",ltarw,...,ltarw,ltarw .k8     Strong
"<EM></EM>",ltarw,ltarw,ltarw,ltarw,ltarw      .k9     Emphasis
rtn,rtn,"<DIV>",rtn,rtn,"</DIV>",uparw  ^K.k8  Division

Lists and Tables
rtn,rtn,"<UL>",rtn,rtn,"</UL>",uparw    ^K.k2  Unordered List
rtn,rtn,"<OL>",rtn,rtn,"</OL>",uparw    ^K.k3  Ordered List
"<LI>",ltarw,ltarw,ltarw,ltarw,ltarw       ^K.k4  List Item
rtn,rtn,"<TABLE>",rtn,rtn,"</TABLE>",uparw      ^K.k5  Table
"<TR>",rtn,rtn,"</TR>",uparw    ^K.k6
```

Из чего вполне можно составить представление о командах языка: в первой колонке следуют разделяемые запятыми (без пробелов!) зарезервированные команды (`rtn` — <Enter>, `ltarw` — <←>, `uparw` — <↑> и т. п.) и вводимые символьные значения в парных кавычках ("<p>"). Далее идут разделенные табулятором клавишные комбинации (`^K 0` — <Ctrl>+<K>−<0>), закрепленные за макросами, и имена макросов. Последние в оригинале представлены в виде Macro 0, Macro 1 и т. п. Но ничего не запрещает при редактировании придать им осмысленные имена. Да и сами макросы могут быть отредактированы должным образом в текстовом редакторе (том же joe, например).

Легко подсчитать, что за один прием можно запротоколировать не более 10 макросов (маркированных цифрами от 0 до 9). Более того, они будут действовать только в течение данного сеанса: по выходе из редактора записанные макросы сами собой не сохраняются. Что, казалось бы, напрочь обесценивает данную возможность.

К счастью, это не так. И один раз запротоколированные макрокоманды можно сохранить для дальнейшего использования. Да и количество их не обязано ограничиваться десятью. Как этого добиться?

Выясняется, что за воспроизведение макросов отвечает конфигурационный файл ~/.joerc, о котором подробнее будет сказано в следующем разделе. И потому достаточно поместить (с помощью <Meta>+<D или из текстового файла) в соответствующую его секцию (какую — также скажу позднее) созданные команды, чтобы обеспечить их исполнение во всех последующих сеансах.

Более того, назначенные по умолчанию клавишные комбинации не являются обязательными. И их можно вручную заменить на любые другие, из числа не использованных ранее. После чего можно начать протоколирование команд сначала, нумеруя их от 0 до 9. Затем — повторить процедуру встраивания их ~/.joerc, и т. д., сколько потребуется (или до исчерпания комбинаторики клавиш). А поскольку количество незадействованных в joe клавишных комбинаций очень велико, то реально число созданных пользователем макрокоманд ограничивается только его фантазией и потребностями.

Таким образом можно легко автоматизировать процесс ввода тэгов HTML или XML, конструкций JavaScript, скриптов командной среды, разметки документов TeX, а также все, что потребуется впредь. Превратив joe в специализированный инструмент для решения почти любых задач.

Настройка joe

Как я уже говорил, некоторые настройки joe можно выполнить интерактивно (вызвав их комбинацией <Ctrl>+<T>). Однако они весьма ограничены и к тому же будут иметь силу только в текущем сеансе. Более интересные и богатые возможности открываются при редактировании конфигурационного файла joerc.

Пример такого файла обнаруживается в каталоге /usr/local/etc/joerc. Первонаперво его надлежит скопировать с свой домашний каталог и переименовать в ~/.joerc — именно этот файл ищется в первую очередь при загрузке редактора. А затем он открывается в любом текстовом редакторе — лучше всего в самом же joe и, после соответствующего изучения, правится вручную.

В экранной документации синтаксис конфигурационного файла не описан, что мотивируется его простотой и понятностью. Это действительно почти так — смысл практически всех настроек можно понять из контекста (и комментариев).

Файл joerc разбит на четыре секции. Первая — это глобальные опции редактора, бóльшая часть которых может быть задана также параметрами командной строки. Все они — односложные и имеют вид -имя_опции (установить данную опцию) или --имя_опции (отменить ее). Опция является установ-

ленной (или, напротив, отмененной), если ею начинается строка (первая колонка, в терминологии программы — это относится и ко всем остальным секциям .joerc). Если строка начинается с пробела или табуляции, дальнейшее ее содержание рассматривается как комментарий и не учитывается. Комментарием является и все, отделенное пробелом от имени опции.

Описание всех опций заняло бы слишком много места, поэтому обращу внимание только на некоторые ключевые моменты. Обязательно следует включить (т. е. удалить пробел в начале строки, если он имеет место быть) опцию

```
-asis
```

Это необходимо для правильного отображения символов кириллицы — иначе они будут показаны латинской транслитерацией.

Полезным представляется также установка опций:

- `-lightoff`

  обеспечивающей выключение подсветки выделенного блока после его перемещения или копирования, — иначе выделение это будет маячить перед глазами;

- `-marking`

  дающей подсветку текста между началом выделяемого блока и текущей позицией курсора.

Можно отменить также создание страховых копий или, напротив, определить место для их помещения, отличное от исходных файлов. Это достигается включением опций

```
-nobackups
-backpath path
```

соответственно.

Интересная возможность — задание количества строк и колонок (знаков в строке) на экране, отличное от определенных для текущего терминала (виртуальной консоли) в целом, что задается опциями

```
-lines #
-columns #
```

где # — количество строк и знаков соответственно.

В этой же секции настраивается вид статусной строки (вывод которой, впрочем, можно и отключить опцией `-nosta`). Он определяется двумя опциями `-lmsg` и `-rmsg`. Первая определяет компоненты, выводимые в левой части строки, вторая — в правой. В любой из них можно вывести индикацию режимов (забивки или вставки, переноса слов, автоотступов), имя файла и показатель его изменения, текущее положение курсора (в строках, колонках или знаках) и т. п. Первый знак левой части статусной строки —

Escape-последовательность, определяющая ее общий вид: инвертирование цветов, выделение подчеркнутым или полужирным начертанием, мерцание.

Например, строка вида

```
-lmsg \i%k%T%W%I%X %n %m%R %M
```

указывает, что в левой части статусной строки в инвертированном виде (черным по белому, `\i`) должны быть выведены:

- префиксные ключи (`%k`), маркирующие включение режимов вставки/замены (`%T`), переноса слов (`%W`), автоотступа (`%I`), прямоугольного выделения (`%X`);
- имя редактируемого файла (`%n`);
- указание на модификацию файла (`%m`) и на режим "только для чтения" (`%R`);
- индикатор включения протоколирования макросов (`%M`).

Строка же вида

```
-rmsg  Row %r Col %c %o %O %u
```

выводит в правой части статусной линии номер текущей строки (`Row %r`) и колонки (`Col %c`) файла, смещение от начала в байтах (в десятичной, `%o`, и шестнадцатеричной, `%O`, формах, и системное время в 24-часовом формате(`%u`).

Возможен вывод и иной информации, как то:

- системного времени в 12-часовом формате (`%t`);
- индикации измененного файла символом `*`;
- ASCII-кода символа под курсором в десятичной (`%a`) или шестнадцатеричной (`%A`) форме;
- процента просмотра файла до позиции курсора (`%p`);
- общего количества строк в файле (`%l`);
- индикации запуска встроенной оболочки (`%S`).

Кроме того, внешний вид статусной строки можно изменить, придав ему, вместо или помимо инверсии, атрибуты подчеркивания (`\u`), полужирного начертания (`\b`), мерцания (`\f`).

Вторая секция конфигурационного файла — это локальные опции, которые можно определить отдельно для файлов различных типов. Для этого в ее составе создаются субсекции вида:

- `*  все файлы`;
- `*.html  документы HTML`;
- `*.c  программы на C`;
- `*rc  конфигурационные файлы`

и т. д. Напомню здесь, что знак маски (*) должен обязательно начинаться с начала строки, т. е. занимать первую колонку.

Для каждой субсекции можно независимо задать такие параметры, как перенос слов (или его отключение), автоматический отступ, величину табуляции и т. п., а также, что интересно, назначить собственную раскладку клавиатуры, отличную от определенной для консоли в целом. Кроме того, с файлами любого типа можно связать макросы, выполняемые при их загрузке или записи.

Третья секция описывает вид экранов помощи, выводимых клавишной комбинацией <Ctrl>+<K>−<H>. Экраны эти могут быть изменены как с позиций внешнего вида (инверсия цветов, выделение или подчеркивание и т. п.), так и по существу. В частности, здесь можно задать специальные экраны помощи для собственных макрокоманд. Более того, редактированием этой секции можно выводить и подсказки на русском (или каком-либо еще) языке. В частности, для макросов ввода тэгов HTML, описанных в одном из предыдущих разделов, можно создать экран помощи вида:

```
{HTML
\i   Help Screen    turn off with ^KH    more help with ESC. (^[.)
\i \i\uGeneral\u    \uBody Text\u                \uList&Table\u
\u
\i \i^KF1 Blank  F5 P  F7 Pre ^KF8 Div  ^KF2 UL ^KF3 OL ^KF4 LI  F10 Href
^KF9
\i \iF1-F4 H1-H4 F6 Br F8 Strong F9 Em  ^KF5 Tab ^KF6 TR ^KF7 TD ^KF10
Name
}
```

Каждый такой экран заключается в фигурные скобки, предваряется собственным идентификатором (в примере — HTML), ему придаются атрибуты внешнего вида (в примере — инвертирование рамки, \i), задается строка подсказки, после чего следует просто перечисление клавишных комбинаций и их описание. В результате соответствующий экран помощи примет вид, как на рис. 18.5.

Рис. 18.5. Экран помощи для ввода HTML-тэгов

Четвертая секция — разного рода ключевые последовательности, или связки (key bindings), в том числе и макрокоманды. Они могут быть определены

отдельно для всех окон (субсекция `:windows`, опять же начиная с первой колонки), окна редактируемого текста (субсекция `:main`), и т. д.

Внимательное рассмотрение секции показывает, что все описанные выше (и перечисленные в приложении) клавишные комбинации joe представляют собой макрокоманды, именно здесь и определенные. Из чего следует два вывода.

Первый — возможность переопределения клавиатурных комбинаций, назначенных для штатных команд joe по умолчанию. Например, если вам не нравится, что пролистывание экрана осуществляется комбинациями <Ctrl>+<U> (назад) и <Ctrl>+<V> (вперед), можно присвоить им иные значения (из числа свободных, разумеется).

Второй вывод — штатные команды joe могут быть (почти) неограниченно наращены собственными макрокомандами. Для чего достаточно их запротоколировать и поместить (напомню, посредством <Escape>−<D>) в соответствующее место четвертой секции (скорее всего, в раздел `:windows` или `:main`) файла ~/joerc. Более того, здесь же им можно присвоить и членораздельные имена (вместо данных по умолчанию Macro 1, Macro 2 и т. п.), во-первых, и изменить закрепленные клавишные комбинации (по умолчанию — <Ctrl>+<K>−<0>, <Ctrl>+<K>−<1> и т. п.) — во-вторых. То есть полностью индивидуализировать редактор, не прибегая ни к каким сильнодействующим средствам (вроде перекомпиляции исходных текстов).

Пользовательские макросы следует помещать в одну из субсекций — `:window` или `:main`. В последней, кроме этого, целесообразно отредактировать имеющиеся там по умолчанию макросы проверки орфографии для работы с русским словарем. По умолчанию эти макросы описываются строками вида:

```
:def spellfile
:def spellword
```

для спеллинга всего текста и отдельных слов соответственно. Они просто описывают параметры вызова внешней программы проверки орфографии (по умолчанию — ispell). И подключение русского словаря при этом, разумеется, не предусмотрено. Чтобы осуществить это, следует просто добавить к обоим строкам после вызова исполнимой команды `ispell` опцию `-d russian`.

Заключительные соображения

Итак, чем же может привлечь пользователя редактор joe? На мой взгляд, многим. Разумеется, если он, т. е. пользователь, вообще испытывает потребность в консольном редакторе.

Во-первых, в его пользу — относительная (по сравнению, скажем, с vi) простота освоения и использования. Немаловажно, что элементарные действия по вводу и редактированию текста в нем могут осуществляться (в том числе и) привычным для выходца из мира DOS/Windows способом.

Второй плюс — единообразность модели выполнения команд, особенно отчетливо проступающая в сравнении с emacs. Модель эта логична и легко запоминается, в том числе и благодаря мнемоническому характеру литерных клавиш, сочетающихся с управляющими.

Если же сравнивать joe с меню-ориентированными редакторами, такими как le, то его отличают а) очень высокая степень настраиваемости (практически не уступающая классическим редакторам командного стиля) и б) быстрота выполнения основных операций по вводу и редактированию. Впрочем, конечно, быстрота эта (как и во всех командных редакторах) достижима только при наработке определенного минимума практических навыков, желательно — доведенных до рефлекторного уровня.

Однако подчеркну, что такие навыки появляются достаточно быстро. И, кроме того, имеются альтернативные им, традиционные (для DOS/Windows) приемы навигации по тексту и пр. Что приближает joe по простоте использования к меню-ориентированным редакторам и делает его пригодным и для эпизодического использования. Чего, в общем случае, нельзя сказать ни о vi, ни о emacs — эффективное их использование возможно только при постоянной практике.

Весьма удобными представляются средства одновременной работы с большим количеством документов и обмена данными между ними. Возможность независимого просмотра различных частей одного файла в отдельных окнах также следует отнести к числу достоинств (коими не обладает, например, le).

Достаточно просто реализована возможность вставки специальных символов, Escape-последовательностей и тому подобных вещей. Что гармонично дополняется возможностью определения для joe клавиатурной раскладки, отличной от используемой в консоли по умолчанию.

Очень эффективно применение joe для составления и редактирования пользовательских сценариев командой оболочки — благодаря возможностям временного выхода в командную строку — раз, запуску единичной команды внутри редактора — два и открытию почти полноценного сеанса командной оболочки изнутри него же (с возможностью записи в виде файла) — три. Судя по документации и конфигурационным файлам, есть средства и для более сложных программистских упражнений, но об их эффективности судить я не компетентен.

Наконец, главное достоинство joe — простота адаптации к специальным задачам, не требующая ни программирования на LISP, ни иных сильнодейст-

вующих средств. Она вполне достижима элементарным протоколированием макрокоманд и несложным их редактированием. Вероятно, такие возможности покажутся смешными записному программисту, но пользователю, профессионально связанному с подготовкой нарративных текстов, они в большинстве случаев более чем достаточны.

К принципиальным упущениям joe можно отнести, пожалуй, только отсутствие подсветки синтаксиса. Что, впрочем, имеет свой плюс — строгий бело-черный интерфейс не вызывает ряби в глазах и действует успокаивающе. Нет также и самостоятельных языковых режимов, хотя в зачаточной форме языково-зависимые привязки можно определить в файле инициализации.

Иными словами, joe — достаточно мощный и удобный инструмент для работы с текстами. Он может быть использован и при периодических работах (вроде правки конфигурационных файлов или составления небольших документов). Но наиболее эффективно применять joe при повседневной работе с большими и структурированными нарративными текстами, особенно — предназначенными для публикации в Сети. Среди всех консольных редакторов joe отличается близким к оптимальному соотношением простоты, мощности и настраиваемости. А посему беру на себя смелость рекомендовать его всем любителям работы в текстовом режиме, буде до сего времени они не приобрели иных пристрастий.

Штатные команды joe и соответствующие им комбинации клавиш

В заключение приведу более-менее подробную справку по штатным командам редактора joe, примерно в соответствии с системой его встроенной помощи.

Перемещения курсора:

- <Ctrl>+<B> — на один знак влево (эквивалентно <←>);
- <Ctrl>+<F> — на один знак вправо (эквивалентно <→>);
- <Ctrl>+<P> — на один знак вверх (в той же колонке по вертикали, эквивалентно <↑>);
- <Ctrl>+<N> — на один знак вниз (в той же колонке по вертикали, эквивалентно <↓>);
- <Ctrl>+<Z> — к предыдущему слову;
- <Ctrl>+<X> — к следующему слову.

Переходы по тексту и скроллинг:

- <Ctrl>+<U> — к предыдущей экранной странице (эквивалентно <PageUp>);
- <Ctrl>+<V> — к следующей экранной странице (эквивалентно <PageDown>);
- <Ctrl>+<A> — к началу строки (эквивалентно <Home>);
- <Ctrl>+<E> — к концу строки (эквивалентно <End>);
- <Ctrl>+<K>–<U> — к началу файла;
- <Ctrl>+<K>–<V> — к концу файла;
- <Ctrl>+<K>–<L> — к строке по номеру;
- <Meta>+ <0>–<9> — переход к нумерованной закладке;
- <Meta>+<Ctrl>+<[> — установка нумерованной закладки;
- <Meta>+<B> — к началу отмеченного (через <Ctrl>+<K>–<B>) блока;
- <Meta>+<K> — к концу отмеченного (через <Ctrl>+<K>–<K>) блока;
- <Meta>+<W> — смещение на одну строку вверх без изменения позиции курсора;
- <Meta>+<Z> — смещение на одну строку вниз без изменения позиции курсора;
- <Ctrl>+<K–<<> — смещение на один экран влево;
- <Ctrl>+<K–<>> — смещение на один экран вправо.

Действия с блоками:

- <Ctrl>+<K>–<B> — начало выделения блока;
- <Ctrl>+<K>–<K> — конец выделения блока;
- <Ctrl>+<K>–<M> — перемещение выделенного блока в позицию курсора;
- <Ctrl>+<K>–<C> — копирование выделенного блока в позицию курсора;
- <Ctrl>+<K>–<W> — сохранение выделенного блока в виде файла;
- <Ctrl>+<K>–<Y> — удаление блока;
- <Ctrl>+<K>–</> — выделение всего текста для применения команд — фильтров оболочки, например, сквозной сортировки (sort), поиска фрагмента (grep) и т. п.

Удаление и вставка:

- <Ctrl>+<D> — знака в позиции курсора (эквивалентно <Delete>);
- <Ctrl>+<H> — знака перед курсором (эквивалентно <Backspace>);

- <Ctrl>+<Y> — строки целиком;
- <Ctrl>+<W> — следующего слова (или пробела);
- <Ctrl>+<O> — предыдущего слова (или пробела);
- <Ctrl>+<J> — части линии справа от курсора до конца;
- <Meta>+<Y> — разрыва строки;
- <Meta>+<O> — линии от позиции курсора до начала;
- <`>–<#> — вставка кода символа, # — номер кода в десятичной (начиная с 0), восьмеричной (начиная с o) или шестнадцатеричной (начиная с x) форме;
- <Ctrl>+<\> — вставка Escape-последовательности;
- <Ctrl>+<_> — отмена последней операции;
- <Ctrl>+<^> — возврат отмененной операции.

Поиск и спеллинг:

- <Ctrl>+<K>–<F> — поиск текстового фрагмента;
- <Ctrl>+<L> — следующего вхождения;
- <Meta>+<R> — наращиваемый поиск назад;
- <Meta>+<S> — наращиваемый поиск вперед;
- <Meta>+<N> — проверка орфографии по словам;
- <Meta>+<L> — проверка орфографии для всего файла.

Выход из редактора и команды оболочки:

- <Ctrl>+<K>–<X> — с записью;
- <Ctrl>+<C> — без записи, с запросом на подтверждение утраты сделанных изменений, буде таковые имелись; также закрытие текущего окна и прерывание операции;
- <Ctrl>+<K>–<Z> — временный выход в командную оболочку с переводом работы редактора в фоновый режим;
- <Ctrl>+<K>–<'> — открытие окна встроенной командной оболочки;
- <Meta>+<!> — запуск одиночной команды во встроенной оболочке.

Файловые операции:

- <Ctrl>+<K>–<E> — вызов файла для редактирования (в новом окне);
- <Ctrl>+<K>–<R> — вставка существующего файла в текущий (в позицию курсора);
- <Ctrl>+<K>–<D> — запись текущего файла с запросом имени.

Управление окнами:

- <Ctrl>+<K>–<O> — разделение окна пополам (только по горизонтали);
- <Ctrl>+<K>–<E> — открытие файла в новом окне;
- <Ctrl>+<K>–<G> — увеличение размера текущего окна (с шагом в одну строку);
- <Ctrl>+<K>–<T> — уменьшение размера текущего окна (с шагом в одну строку);
- <Ctrl>+<K>–<N> — переход к нижнему (или предыдущему) окну;
- <Ctrl>+<K>–<P> — переход к верхнему (или последующему) окну;
- <Ctrl>+<C> — закрытие текущего окна;
- <Ctrl>+<K>–<I> — показать или скрыть все окна, кроме текущего (переключатель).

Протоколирование макрокоманд:

- <Ctrl>+<K>–<[> <0>–<9> — включение режима протоколирования макроса;
- <Ctrl>+<K>–<]> — выключение режима протоколирования макроса;
- <Ctrl>+<K>–<0>–<9> — воспроизведение запротоколированного в текущем сеансе макроса по его номеру;
- <Meta>+<D> — вставка записанных в данном сеансе макросов в текущий файл.

Разное:

- <Ctrl>+<K>–<J> — форматирование абзаца;
- <Ctrl>+<K>–<A> — центрирование строки;
- <Ctrl>+<K>–<,> — уменьшение отступа;
- <Ctrl>+<K>–<.> — увеличение отступа;
- <Ctrl>+<T> — вызов настроек (имеющих силу только в текущем сеансе);
- <Ctrl>+<R> — обновление экрана;
- <Ctrl>+<K>–<Spacebar> — вывод строки статуса в нижней части экрана (номер текущей строки и колонки, смещение от начала файла, ASCII-код символа под курсором);
- <Ctrl>+<K>–<\> — повторение команды;
- <Meta>+<M> — ввод регулярного выражения;
- <Meta>+<H> — отправка сообщения (нечто вроде команды `write`, но на одном и том же терминале).

Дополнительные источники

- В отличие от vi/Vim и emacs, редактор joe очень слабо освещен в русскоязычной литературе. Единственное исключение — переводная книга Петцке К. Linux. От понимания к применению. — М.: ДМК, 2000, где ему посвящена специальная глава.
- Подробную информацию о joe в Интернете найти также нелегко. Наибольшее количество сведений о нем обнаруживается на странице Норма Марлоффа (Norm Matloff) — **http://heather.cs.ucdavis.edu/~matloff/joe.html**, где содержится нечто вроде руководства пользователя (на английском, разумеется, языке).

Глава 19

Прочие консольные приложения

Текстовыми редакторами, разумеется, не исчерпывается список доступных в FreeBSD приложений. В этой главе будут рассмотрены программы консольного режима, призванные не только облегчить работу пользователя и скрасить его досуг. Это — инструменты для обработки изображений и мультимедийные приложения. В заключение будут рассмотрены средства работы с кириллицей и всякого рода языково-зависимые программы.

Графика в консоли

Казалось бы, текстовый режим системной консоли и работа с графикой суть вещи несовместные. И действительно, консоль — не самое подходящее место для редактирования полноцветных фотореалистических изображений или сложной векторной графики. Однако кое-какие манипуляции с изображениями можно производить и здесь.

На протяжении всего времени работы с Linux меня угнетала невозможность изготовления истинных скриншотов консольных программ. Пока в FreeBSD не обнаружилось неожиданное решение. Это была утилитка под названием scr2png, входящая в систему портов и доступная также как бинарный пакет. Именно для создания консольных скриншотов и приспособленная. Принцип ее действия основывался на двух особенностях системной консоли FreeBSD: возможности подгрузки шрифтов и создания дампа экранного буфера. И за ту, и за другую отвечает команда `vidcontrol`, описанная в *главе 12*. Соответственно, scr2png способна функционировать только в связке с последней. Но зато — вполне справно.

Как уже говорилось, сам по себе дамп экрана текущей консоли создается командой

```
$ vidcontrol -p
```

в бинарном формате, или

```
$ vidcontrol -P
```

в текстовом, с игнорированием непечатных символов и текстовых атрибутов. Разумеется, такой дамп может быть изготовлен и для любой иной, отличной от текущей, виртуальной консоли — следует только перенаправить ее на стандартный ввод команды `vidcontrol`:

```
$ vidcontrol -p < /dev/ttyv#
```

И в любом случае вывод команды может быть перенаправлен в файл, однако само по себе радости это доставит мало — нужно обладать изрядной фантазией, чтобы принять такой dump за изображение чего либо, имеющего быть на консоли.

Программа же scr2png просто-напросто конвертирует такой экранный дамп в графический файл — как не трудно догадаться из ее названия, формата именно PNG, а не какого-либо другого. Но не просто, а с учетом экранного шрифта. Который в принципе не обязан совпадать с загруженным в настоящий момент. Отсюда синтаксис команды —

```
scr2png -f имя_шрифта
```

где в качестве аргумента можно взять любой шрифт из каталога /usr/share/syscons/fonts/ (причем путь к шрифтовому файлу должен быть указан в явном виде). Разумеется, если есть желание воспроизвести русский текст, шрифт этот должен содержать символы кириллицы в должной кодировке. А для воспроизведения именно копии экрана консольный шрифт и шрифт для scr2png должны совпадать — правда, в этом случае аргумент можно не указывать вообще. Хотя, с другой стороны, это — один из способов проверить, как будет выглядеть экран при другом консольном шрифте, не загружая последний на самом деле.

Продолжим, однако, об изготовлении скриншотов. И указанная выше команда не приведет ни к какому результату (даже к возврату приглашения командной строки). Потому что все описанное хозяйство нужно объединить в командную конструкцию с соответствующим перенаправлением ввода и вывода. Сделать это можно двояким способом. Во-первых, через промежуточный scr-файл:

```
vidcontrol -p > shot.scr; \
scr2png < shot.scr > shot.png
```

Во-вторых, конвейеризацией команд:

```
vidcontrol -p | scr2png > shot.png
```

В обоих случаях в итоге будет образован графический слепок экрана текущей консоли (файл shot.png). Однако такой слепок может быть получен и для любой другой доступной консоли, для чего она должна быть указана в качестве устройства ввода команды `vidcontrol`:

```
vidcontrol -p < /dev/ttyv# > shot.scr;
scr2png < shot.scr > shot.png
```

или

```
vidcontrol -p < /dev/ttyv# | scr2png > shot.png
```

Наконец, если есть желание посмотреть на вид консоли при каком-либо другом шрифте, его следует задать явно:

```
vidcontrol -p < /dev/ttyv0 | scr2png \
/usr/share/syscons/fonts/866-sans-8x16.fnt > \
~/work/images/shot.png
```

Именно посредством программы scr2png сделаны все скриншоты консольных программ в этой книге. Возможно, качество их не всегда соответствует идеалу. Но какая другая система позволяет в принципе делать графические слепки программ текстового режима?

Графические возможности консоли не исчерпываются возможностью изготовления скриншотов. Так, в состав коллекции портов входит ImageMagik — набор многофункциональных утилит для манипулирования с растровыми изображениями. Некоторые из них, такие, как display (общий интерфейс для всех программ пакета), или import (средство создания скриншотов), непременно требуют графического режима, предоставляемого системой X Window. Однако другие вполне способны функционировать в режиме консольном.

Среди них — утилита convert, центральный компонент пакета, обеспечивающий преобразование растровых изображений из одного формата в другой, причем — в пакетном режиме, без открытия исходных файлов. Формат команды —

```
$ convert исходный_файл целевой_файл
```

Команда автоматически опознает формат исходного файла и преобразует его в формат, соответствующий расширению имени целевого файла (один из немногих в UNIX случаев, когда расширение файла играет роль).

Список распознаваемых программой convert графических форматов поистине необъятен. Сосчитать их количество я, честно говоря, поленился, но среди них заведомо присутствуют все распространенные форматы — TIFF, GIF, JPEG, PNG, BMP, PCX и множество других. Программа способна работать и с форматами, которые могут содержать в одном файле несколько изображений, такими, как анимированный GIF или MPEG.

Многочисленные опции команды convert позволяют объединять серию изображений в единую последовательность (или — наоборот, разбивать последовательность на части), добавлять комментарии, указывать степень и алгоритм сжатия, изменять разрешение и глубину цвета и многое другое (вплоть до наложения фильтров).

С помощью программы convert, например, можно легко перевести сотни отсканированных в формате TIFF картинок в формат JPEG для представле-

ния в Web, причем с одновременным изменением расширений файлов для их корректного распознавания Web-сервером. Для этого достаточно написать простенький сценарий:

```
for i in *.tif
do
convert $i jpg: 'basename $i.tif'.jpg
done
```

Следует учесть только, что образуемые в процессе преобразования временные файлы помещаются в каталог /var. И если последний выделен в отдельную файловую систему (как это рекомендуется программой установки FreeBSD, о чем говорилось в *главе 4*) — необходимо, чтобы места в ней было вдоволь...

Из других утилит пакета ImageMagik стоит упомянуть программы combine и montage. Первая из них создает одно изображение из нескольких файлов путем наложения друг на друга. Вторая же выполняет монтирование нескольких изображений в единое панно, с подписями в виде имен файлов (по умолчанию) или без оных.

Консольный Интернет

С момента появления браузера Netscape Communicator интернет-серфинг прочно ассоциируется с графическим режимом и ОС Windows. Что лишь усугубилось набором популярности Internet Explorer (IE) — неотъемлемой, по утверждению создателей, части этой системы. Однако не следует забывать, что сама по себе Сеть Сетей — в значительной мере порождение UNIX-сообщества. И задолго до Netscape Communicator и IE средства просмотра Web-страниц в сети существовали — и использовали традиционную текстовую среду.

Я не буду углубляться в седую древность — расскажу лишь о двух браузерах текстового режима, и по сию пору представляющих не только исторический интерес. Почему стоит помнить о них в век Web-страниц, насыщенных статической и динамической графикой и прочими технологическими из... хотел сказать извращениями, но решил воздержаться? Во-первых, они отличаются исключительным быстродействием, во-вторых, обеспечивают столь же комфортное зрительное восприятие при просмотре текстовых материалов, как и консольные редакторы — при их наборе. А в-третьих, текстовые браузеры — традиционный для FreeBSD инструмент просмотра не только Web-страниц, но и документации в html-формате.

Первым в ряду текстовых браузеров следует упомянуть lynx — мы ведь помним по *главе* 3, что именно он был предложен по умолчанию при определении опций установки системы. Запускается он очень просто командой `lynx`,

в качестве аргумента которой могут быть указаны имя локального файла, URL или численный IP-адрес. После чего на экране возникает соответствующая html-страница в чисто текстовом исполнении — графические элементы отображаются по умолчанию ссылками на соответствующие файлы-источники.

Отличительная особенность браузера lynx — очень строгий подход к интерпретации html-кода: он требует весьма точного его соответствия спецификациям консорциума W3C (WWW Consortium), иначе он может просто отказаться читать страницу. И потому lynx незаменим как одно из средств проверки валидности собственных Web-творений.

Основная слабость браузера lynx — непонимание им фреймов и неправильное отображение таблиц html-кода. Этих недостатков лишена программа links — также текстовый браузер, очень сходный с lynx, по способный нормально работать с таблицами и фреймами. А также имеющий удобное меню, которое, при правильной установке локали и соответствующих настройках, может быть русскоязычным.

Мультимедийная консоль

> Денег нет, в стране бардак, в воде холера,
> На душе ненужные сомнения.
> Лишь портвейн да музыка Малера
> Успокаивают мне пищеваренье.
>
> *Тимур Шаов*

Думаю, не ошибусь, если скажу, что для многих наличие звукового фона — одно из необходимых условий эффективной работы за компьютером. Что же касаемо технического осуществления такой потребности, она реализуется обычно посредством либо аудио-CD, либо воспроизведением mpeg-файлов с любого носителя; реже прибегают к RealAudio, еще реже — к прочим форматам оцифрованного звука.

Немаловажный момент околопроизводственной мультимедиа — возможность "граббить" (не от ограбления, а от grabbing, хотя последнее тоже могут полагать аморальным) понравившиеся аудио-CD: во-первых, высвобождения привода ради, во-вторых, сохранности его для, в-третьих, во имя экономии усилий по смене дисков. Ну а на-граббленное следует конвертировать во что-то компактное (тот же MPEG, например) — иначе на WAV винчестеров не напасешься.

Как же в этом отношении смотрится консоль FreeBSD? Как выясняется, неплохо. Конечно, о RealAudio в консоли придется забыть — на сей предмет существует (насколько мне известно) лишь стандартный RealPlayer, функ-

ционирующий только под "иксами" (да и то имеющий лишь Linux-версию). Остальное же — можно.

Конечно, прежде чем музыку слушать, следует позаботиться о том, чтобы было, чем. То есть — настроить звуковую карту. Однако для большинства современных моделей для шины PCI (в том числе и встроенных в материнские платы звуковых чипов) это — достаточно простая задача. Достаточно перекомпилировать ядро, внеся предварительно в его конфигурационный файл строку вида

```
device      pcm
```

и дело в шляпе. Старые ISA-модели, особенно не имеющие функций Plug-and-Play, могут потребовать несколько большего количества действий, однако непреодолимых трудностей, как будто, и здесь не предвидится (за деталями отсылаю к дополнительным источникам). Судя по всему, и недо-звук программных кодеков AC'97 также должен воспроизводиться нормально.

Победив тем или иным способом звуковую карту, можно перейти к обсуждению вопроса звукопрослушивания. Начнем с воспроизведения аудио-CD. Самая простая утилита на сей предмет называется, как ни странно, cdplay. Чтобы прибегнуть к ней, требуется: а) вставить компакт в привод и б) набрать в командной строке вышеозначенное заклинание. Музыка с диска мгновенно начинает звучать, а приглашение командной строки — столь же мгновенно возвращается обратно. Вследствие чего в данной консоли можно продолжать общественно-полезную деятельность, но уже с аудиосопровождением. Правда, управлять воспроизводством диска не получится, но ведь для фонового аккомпанемента работы — это не так и важно, не так ли?

Главное средство для воспроизведения mpeg-музыки также просто, как грабли. Да и название у него соответствующее — mpg123. Призванное сказать, что mpeg-музыку крутить — это все равно, что... Lotus 1-2-3 изучить, скажем.

Программа эта так и запускается — набором `mpg123` с указанием имени mpeg-файла в качестве аргумента, после чего этот файл и начинает воспроизводиться. Аргументом может служить и путь к каталогу с mpeg-файлами и вездесущим символом звездочки (*) — тогда будут проигрываться все наличествующие здесь mpeg (непрерывно, в порядке ASCII-кода первого символа имен файлов), даже озадачиваться указанием маски не требуется. Конечно, как и любая уважающая себя UNIX-программа, mpg123 имеет вдоволь опций командной строки — при желании ознакомиться с ними можно посредством `mpg123 -?` или `mpg123 --help`.

Со средствами управления и здесь не густо, имеющиеся достижимы только указанием соответствующих опций командной строки. Однако для фонового сопровождения и этого хватает — сгрузил любимые, успокаивающие нервы, песенки, в один каталог, задал путь к нему, и трудись далее в свое удовольствие.

Другое дело, что, если проигрывать через mpg123 более одного файла, консоль, с которой он был запущен, оказывается занятой до окончания процесса. Чтобы избежать этого, следует запускать эту программу в фоновом режиме, с указанием символа амперсанда (&, через пробел) после указания аргумента. Интересно, что при этом самого по себе приглашения командной строки не появляется, но на введенные с клавиатуры команды система реагирует как обычно. Хотя некоторые из введенных позднее команд могут оборвать проигрывание. В общем, надежнее запускать mpg123 в собственной консоли — благо, как неоднократно говорилось ранее, их можно расплодить сколько душе угодно...

Еще один интересный момент — в фоновом режиме можно запустить несколько копий mpg123, каждая — со своим набором файлов в качестве аргументов. Чем любители сюрреализма могут создать весьма забавные эффекты. Конечно, одновременно с mpeg-файлами можно воспроизводить и треки с аудио-CD, но это не столь интересно.

Ни один из помянутых выше плееров не позволяет делать простой вещи — регулировать уровень звука, его баланс и тому подобные вещи, представляющиеся абсолютно необходимыми — тянуться каждый раз к регуляторам на колонках в процессе творческой работы обычно лениво. Однако тут на помощь приходит специально предназначенная для этого утилита aumix. Она может быть запущена одновременно с плеером CD и (или) MPEG, в том числе и в любой из свободных виртуальных консолей. Средства управления, предоставляемые ей, могут показаться несколько непривычными, но вполне элементарны в обращении.

Все описанные здесь средства могут показаться донельзя примитивными по сравнению со всякого рода WinAmp или XMMS, работающими в графическом режиме. Однако для создания звукового фона при работе они, на мой взгляд, вполне достаточны. Отличаясь предельной простотой использования, не отвлекающей (в том числе и внешними эффектами типа скиннов) от течения творческой мысли. Тем не менее, и в консоли можно прибегнуть к более изощренным средствам воспроизведения mpeg-музыки.

Одно из них — mpeg-проигрыватель mp3blaster (рис. 19.1). Он обладает, во-первых, возможностью управления воспроизведением файлов (остановки, паузы, перехода), осуществляемой простыми нажатиями цифровых клавиш. Во-вторых, в нем предусмотрены средства регулирования громкости, баланса и прочего, делающие ненужным обращение к внешним микшерам. И, наконец, в mp3blaster можно легко создавать плей-листы из произвольных наборов файлов, как для текущего сеанса, так и сохраняемые на диске (в простом текстовом формате).

В последнее время определенную популярность приобретает новый формат аудиофайлов — OGG, являющий собой аналог MPEG, однако — открытый и свободно распространяемый. Разумеется, и FreeBSD не прошла мимо этой

тенденции, обзаведясь (в коллекции портов) соответствующим инструментарием — vorbis-tools. В него включены программы — ogg123 (средство воспроизведения ogg-файлов), oggenc (конвертер wav-файлов в формат OGG) и ogginfo (средство для получения информации об ogg-файлах). Обращение с ними настолько элементарно, что задерживаться на этом не буду.

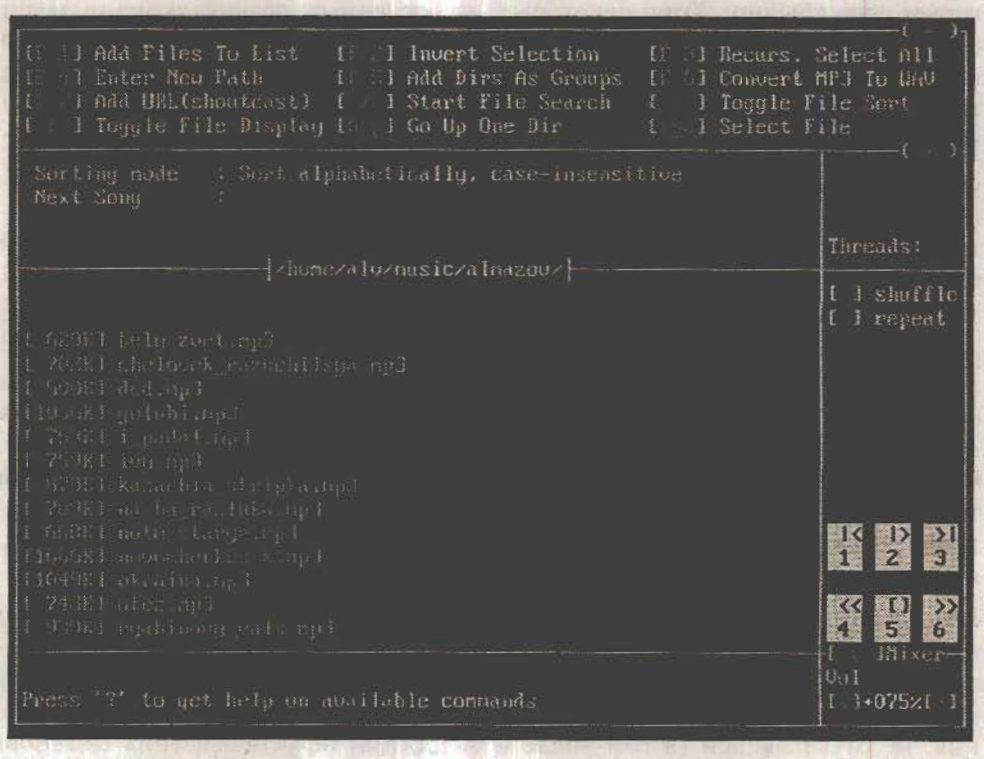


Рис. 19.1. Программа mp3blaster для воспроизведения mpeg-аудиофайлов

Потому что пора поговорить о том, как получить то, что мы желаем воспроизводить нашими плеерами.

Один из моих бывших воинских начальников любил говаривать, что для того, чтобы сварить суп из курицы, нужно как минимум иметь курицу. Злые языки добавляли, что для этого довольно и кошки, но это — искажение догмы: ведь не все любят кошек, и еще меньше тех, кто умеет их готовить. Соответственно, чтобы проигрывать mpeg-файлы (а именно этим занимается в ходе работы большинство из нас, не правда ли?), нужно эти самые файлы иметь.

Каким же образом их получить? Разумеется, на-граббить с аудио-CD. И не будем лицемерно восклицать, что тем самым нарушаются чьи-то там автор-

ские права. Поскольку нынче речь идет не о морали, а о технике дела. Так вот, с технической стороны для о-грабления аудио-CD в коллекции портов FreeBSD существует программа dagrab. Для использования ее требуется вставить CD-диск в привод (разумеется, не монтируя его) и дать одноименную команду с указанием необходимых опций и, возможно, аргументов.

```
$ dagrab [oprions] ##_tracks
```

А из опций абсолютно необходима по меньшей мере одна: `-d имя_устройства`. По умолчанию в программе dagrab качестве такового выступает /dev/cdrom (широко используемая во многих UNIX-системах ссылка на имя реального устройства чтения CD-дисков), однако в FreeBSD потребуется указать именно реальное устройство — скорее всего, это будет /dev/acd0.

Далее, опция `-i` выведет список трэков текущего CD-диска. Если в качестве аргумента команды `dagrab` указать один из этих номеров, соответствующий трэк немедленно будет за-грабблен и записан на диск. Указав через пробелы несколько номеров трэков, можно захватить их в один прием. А если вместо этого задать опцию `-a`, о-грабблению будет подвергнут весь диск.

По умолчанию `dagrab` присваивает создаваемым аудиофайлам имена вида track0#.wav. Однако опцией `-f` можно задать произвольную маску для создаваемых имен файлов. Например, при о-грабблении диска автора эпиграфа к данному разделу она может выглядеть как shaov%02.

Таким образом, форма использования программы, скорее всего, будет такой:

```
$ dagrab -d /dev/acd0 -f name%02d.wav 1 2... #
```

для захвата нескольких трэков, и

```
$ dagrab -d /dev/acd0 -f name%02d.wav -a
```

для о-грабления целого диска.

Команда `dagrab` имеет и дополнительные опции, позволяющие задавать условия записи аудиофайлов (например, в монофонической форме), создавать базы данных аудиозаписей и т. д. Ознакомиться с ними можно, как всегда, на соответствующей man-странице.

В открытых UNIX-системах существует и другой инструмент для той же цели — программа под забавным названием cdparanoia. Хотя как раз ничего параноидального в ней нет: набираем это слово в командной строке (не забыв вставить диск в привод, конечно), в качестве первого аргумента указываем номер трека (просто цифрой, например, 1), а в качестве второго — желаемое имя выходного wav-файла. Если не сделать последнего, он автоматически получит имя типа *cdda.wav*.

И cdparanoia имеет вдоволь опций командной строки, о которых можно узнать стандартным образом. Я же задержусь только на одной, наиболее упот-

ребимой — опции -B (от batch), позволяющей о-граббить за один присест сколько угодно треков, хоть весь диск сразу. Для этого после данной опции указываются либо номера треков через дефис, либо (для о-грабления всего диска) два дефиса подряд; в последнем случае, указав дополнительно опцию -#, можно исключить из процесса трек за номером #. То есть команда

```
$ cdparanoia -B -- -3
```

за-граббит весь аудио-CD, за исключением третьего трека. В случае пакетного о-грабления указывать имена файлов не нужно — они получат их автоматически, в виде track01.cdda.wav, track02.cdda.wav и т. п.

Полученные wav-файлы могут быть прослушаны программами типа waveplay или wavplay — последняя позволяет, кроме этого, производить аудиозапись с микрофона или линейного ввода звуковой карты.

Однако о-грабление диска — лишь первый этап, второй же — преобразование полученных wav-файлов в формат MPEG. Вот уж для этого консольный режим предоставляет массу инструментов. Один из них — программа lame (реализованная, кстати, практически для всех платформ). В элементарном исполнении она требует только указания двух аргументов — имени исходного WAV и целевого MPEG. Впрочем, последний не обязателен — файл на выходе автоматически получит имя вида *.wav.mp3. А в неэлементарном виде многочисленные опции дадут возможность выбирать уровень mpeg-кодирования (от 1-го по 3-й), уровень компрессии (и качество звучания соответственно), создавать моно- или стереофонические записи, задавать bitrate и прочие параметры, каковые можно изучить посредством `lame -?`.

Программу lame можно использовать в паре со средствами захвата аудиотреков, объединив их в конвейер конструкцией типа

```
$ dagrab [options] # | lame - имя_файла.mp3
$ cdparanoia # | lame - имя_файла.mp3
```

В любом из этих случаев сначала будет захвачен третий трек с диска, а потом получившийся wav-файл будет конвертирован в формат MPEG. Символ - (дефиса) в качестве первого аргумента команды `lame` указывает, что входные данные она получает со стандартного ввода, каковым в данном случае оказывается wav-файл, получаемый на выводе программы аудиограббинга.

Еще один удобный инструмент для пакетной конвертации в mpeg-формат — программа bladeenc. Разумеется, она умеет конвертировать и единичные файлы —

```
$ bladeenc input.wav output.mp3
```

и некоторое количество их —

```
$ bladeenc input1.wav output1.mp3... input#.wav output#.mp3
```

Однако главная ее способность — конвертировать wav-файлы (предварительно на-грабленные, например, с аудио-CD) целыми каталогами, с записью в любой иной каталог:

```
$ bladeenc -outdir=path path/*.wav
```

К тому же она имеет множество дополнительных опций конвертации, ознакомиться с которыми можно, запустив bladeenc без опций и аргументов (man-страницы к дистрибутиву программы не прилагается).

Возвращаясь к формату OGG, можно заметить, что есть и средства конвертации в него mpeg-файлов — программа mp32ogg, работающая по аналогичному принципу:

```
$ mp32ogg input.mp3 output.ogg
```

Однако это, скорее всего, приведет к некоторой потере качества записи. И потому логично воспользоваться средством для непосредственного преобразования wav-файла в ogg-формат — программой oggenc. Она требует в качестве аргумента имени входного файла (или файлов). А в виде опций могут задаваться параметры конвертации, имена выходных файлов или маски их генерации, комментарии к записям и прочие вещи, позволяющие создавать полноценные альбомы. Однако в текущих версиях oggenc работает не всегда корректно — в частности, она иногда отказывается воспринимать вполне нормальные wav-файлы, утверждая, что они коррумпированы.

Средства работы с кириллицей

Для эффективного использования практически всех приложений в наших условиях необходима одна мелочь — полноценная русификация системы. Эта тема неоднократно затрагивалась по ходу повествования, однако некоторые аспекты проблемы остались неосвещенными. Среди них — кодировки Великого и Могучего.

Своеобразие исторической судьбы России проявилось, в частности, и в том, что русский язык — самый многокодировочный язык в мире. Это создает, конечно, некоторые трудности в повседневной жизни. Но зато — какой простор для проявления исконно народной смекалки...

Перечисление ныне существующих (т. е. используемых) кодировок кириллических символов — уже требует изрядного напряжения интеллекта, особенно если учесть все синонимы, псевдонимы и эпитеты. Здесь

- и классическая кодировка DOS, она же — CP866, IBM866, альтернативная (на полноту перечня не претендую); по сию пору она есть стандарт междокументного обмена — так, РФФИ принимает электронные версии проектов только в ней (и в чисто текстовом формате);
- и не менее традиционная KOI8-R — кодировка электронной почты и, до недавнего времени, Рунета;

- и ISO8859-5, кодировка Sun, по не вполне понятным причинам сподобившаяся канонизации в качестве кодировки ГОСТ;
- и CP1251, она же — кодировка Windows, не только добившаяся почти безраздельного господства на пользовательских десктопах, но и теснящая KOI8 в ее исконной вотчине — интернет-серверах;
- и кодировка Mac OS, правда, слава богу, так и не вышедшая за пределы своей платформы.

А еще где-то за кадром остаются "болгарская" кодировка (как легко понять из названия, к Болгарии почти никакого отношения не имеющая), и древняя кодировка того же Mac (как помнится, даже не одна), и апокалиптический Unicode...

Легка жизнь пользователей, закосневших в чистом "черном" DOS или присягнувших на верность OS/2: как бы то ни было, их CP866 признается всеми — нет, наверное, текстового процессора, не имеющего функции сохранения документа в этом формате. Не сложнее — и мучителям Windows (или мучимым оной): они могут позволить себе просто не знать о существовании иных кодировок, кроме CP1251. Ну, а эзотерикам от Mac OS — им, подозреваю, ничего, кроме собственной кодировки и не нужно. Не знаю уж, как обходятся Solaris — но и от них сетований как будто не слышно.

А вот пользователи открытых и свободных UNIX-систем, в том числе и FreeBSD, обязаны учитывать все изобилие кириллических кодировок. Хотя бы потому, что большинство из них даже не поднимаясь из-за своего десктопа, используют две из них — KOI8 для клавиатурного ввода и CP866 для экранного вывода.

Кроме того, пользователи FreeBSD живут не в безвоздушном пространстве — время от времени им требуется читать документы, созданные на иных платформах (пользователи которых, как уже говорилось, подчас даже не подозревают, что кроме их кодировок существуют какие-либо иные). Причем — чем дальше, тем больше: как я уже говорил, даже в русскоязычное Web-пространство проникло тлетворное влияние Microsoft.

Первое напрашивающееся решение при этом — использование перекодировщиков. И действительно, таковых создано немало. Самый простой из них — программка iconv. Она являет собой компонент общесистемной библиотеки libc (glibc) и потому гарантированно присутствует в любой UNIX-системе (в том числе и в FreeBSD). Формат ее предельно прост:

```
iconv -f [исходная_кодировка] -t [целевая кодировка] имя_файла
```

Она выводит результат своей работы на стандартный вывод, поэтому для сохранения его в файле следует прибегнуть к операции перенаправления вывода:

```
iconv -f enc1 -t enc2 source_file > target_file
```

Есть и более развитые средства, позволяющие в один присест перекодировать несколько файлов (и даже целый каталог). Среди них — rusconv Олега Паращенко. Правда, в штатный комплект портов или пакетов FreeBSD она не входит, но легко собирается в этой системе. Она позволяет осуществить перекодировку произвольного количества файлов (в том числе и по маске) с сохранением в текущем или любом другом каталоге. И при этом не только между большинством распространенных кодировок кириллицы (DOS, Windows, KOI, Mac), но и с превращением латинской транслитерации в читабельный русский текст (или наоборот). Большое достоинство программы — ее прекрасная документированность, руководство пользователя для нее (на русском языке) составляет несколько десятков страниц. Это избавляет меня от описания ее установки и работы — интересующихся отсылаю к сайту автора (см. дополнительные источники).

Еще более удобной представляется мне программа d1489 (автор — Андрей Чернов, известный, помимо прочего, и своими трудами по русификации UNIX), доступная в FreeBSD в виде порта и пакета (в дистрибутивах Linux она мне не попадалась, может быть, просто не обращал внимания). В отличие от rusconv, она не документирована вообще — к ней нет ни man-, ни info-страниц, лишь очень краткая справка по использованию. Впрочем — ни в чем более она и не нуждается, предоставляя, однако, вполне достаточные возможности.

Пакет d1489 включает в себя две пары утилит — fromwin/towin и fromdos/todos, назначение которых вполне ясно из их имен: это перекодирование между KOI8 и Windows/DOS. Есть в ней и еще одна утилитка — a2kfcnv, для трансформации экранных консольных шрифтов из CP866 в KOI8, что может оказаться отнюдь не лишним.

Используется каждая из парных утилит одинаково, хотя и разнообразно (о чем мы узнаем посредством запуска любой команды с опцией `-h` или `-?`). Можно дать команду

```
$ fromwin имя_файла
```

в результате чего исходный файл в кодировке CP1251 будет замещен им же, но в KOI8. Можно перенаправить результат перекодировки в другой файл —

```
$ fromwin < Win-файл > KOI-файл
```

А можно перекодировать и множество файлов, в том числе и по маске (например, все текстовые или html-документы каталога) с записью результатов в другой каталог, скажем

```
$ fromwin -o output_dir ~/*.html
```

По умолчанию при перекодировании из KOI8 в DOS или Windows и обратно обе пары программ (fromwin/towin и fromdos/todos) заодно преобразуют и символы конца строк в вид, принятый в целевой системе. Что при желании

можно подавить опцией `-b`. Ну, а опция `-p` по возможности пытается сохранить такие атрибуты файлов, как время модификации (насколько это осуществимо при переносе между DOS/Windows и UNIX). Наконец, если перекодируемый в KOI8 файл находится на носителе с файловой системой FATxx, а целевой файл помещается на BSD-раздел диска, с него автоматически снимается атрибут исполнения (каковой появляется при обычном копировании файлов с FAT- на UFS-разделы).

Короче говоря, пакет очень удобен, если требуется регулярно переносить большие объемы данных на фиксированные носители (например, Zip-диск или сменный винчестер) и обратно, позволяя делать это парой-тройкой несложных скриптов. Правда, она не позволяет рекурсивного перекодирования вложенных подкаталогов — но такой возможности я пока не видел ни в одной программе-перекодировщике.

Однако перекодировщики не решают всех проблем обмена документами — достаточно скучно перекодировать все, что потребуется впредь (или, напротив, нужно только сиюминутно). А уж если нужно поправить пару символов в Windows или DOS-документе...

Так что очень желательной видится возможность переключения кодировок терминала, что называется, на лету (или на бегу?). К счастью, и такое не запрещается — благодаря механизму загрузки клавиатурных раскладок и экранных шрифтов (да и карт соответствия, буде потребуется — также). В FreeBSD эти функции возложены на штатные программы из пакета системной консоли (syscons) — vidcontrol и kbdcontrol. Посредством первой можно подгрузить шрифт в любой кодировке, например:

```
vidcontrol -f 8x16 cp1251-8x16.fnt
```

А второй — задать соответствующую ему клавиатурную раскладку:

```
kbdcontrol -l ru.cp1251.kbd
```

Разумеется, для этого нужно иметь соответствующие экранные шрифты и клавиатурные раскладки. Но эта проблема уже рассматривалась — в рассказе о системной консоли *(см. главу 12)*. Если нет желания менять экранные шрифты, той же командой `vidcontrol` в форме

```
vidcontrol -l *.scm
```

можно подгрузить и требуемую карту соответствия. Опять же, если таковая имеется — а штатно она есть только для KOI8 -> CP866. Но и тут принципиальных сложностей не предвидится — с помощью толики терпения нужную карту можно склепать вручную.

Однако средства syscons — не панацея от всех бед. Главный их недостаток — переключение шрифта, раскладки и таблицы соответствия между ними действует на все виртуальные консоли сразу. И потому мечта — сочинять текст в KOI8, время от времени просматривая документы, сохраненные в

CP1251, — имеет шанс так и остаться мечтой. А уж о том, чтобы таскать между ними фрагменты, казалось бы, можно забыть.

Можно — если бы не замечательная программа mapchan, позволяющая (цитирую файл README из ее комплекта) "произвольным образом преобразовывать терминальный ввод/вывод". И дающей, таким образом, возможность работать на различных виртуальных консолях с документами в любых кодировках (в общем случае — не только кириллицы, и не только на локальной машине, но и с удаленных терминалов, но этот аспект я затрагивать не буду). Правда, и она не входит в коллекцию портов, но легко может быть получена с сайта фирмы Крон (см. дополнительные источники).

Программа mapchan (автор ее представляется как Юрий Калиниченко) является собой модернизацию одноименной утилиты из SCO UNIX и написана, судя по всему, под Linux. Однако ничего Linux-специфичного не содержит и потому должна работать под любым UNIX. За любой — не скажу, но под FreeBSD она устанавливается без проблем и работает прекрасно.

Установка ее — стандартна, распаковывается архив и от имени root даются команды

```
$ make
$ make install
```

Правда, при этом свой исполнимый файл (mapchan же) записывается непосредственно в каталог /bin, а дополнительные компоненты (в частности, собственно таблицы перекодировки ввода/вывода — map-файлы) предлагается руками положить в каталог /etc/mapchan. Если это почему-либо не устраивает — можно обойтись без `make install` и вручную расписать их куда угодно (например, как $HOME/bin/mapchan и т. п.). Далее программа запускается следующим образом

```
$ mapchan -s [Shell] -f /path/file.map
```

после чего в текущей консоли грузится указанная опцией `-s` оболочка (по умолчанию, если эта опция отсутствует — /bin/sh), в которой можно наслаждаться чтением и писанием русских текстов в той кодировке, которая определена заданным через опцию `-f` map-файлом, вне зависимости от того, какая кодировка определена для системы в целом. При этом не затрагиваются ни клавиатурные раскладки (все буквы находятся на тех же местах, что и раньше), ни их переключатели (кириллица/латиница переключаются тем же способом), ни экранные шрифты (вид их остается неизменным). Более того, все прочие консоли сохраняют общесистемную кодировку. И выделенный мышью фрагмент в консоли с KOI8 в правильном виде вставляется на консоль с Win- или DOS-кодировкой.

Красота, да и только... Дело за малым — обзавестись достаточным количеством map-файлов для всех требуемых случаев. В комплекте таковые имеются для работы с DOS-кодировкой в KOI-терминале, и с кодировкой KOI8-U

при терминалах с альтернативным DOS (он же — CP866) и с кодировкой ГОСТ. Есть и вариант для некоего терминала CM7238 — я с таким не сталкивался.

А вот самый актуальный ныне случай — для работы с CP1251 при KOI-терминале — не представлен. Изготовлением соответствующего map-файла следует озадачиться самостоятельно. Благо, вследствие формата map-файла это просто: определяются две секции — input и output, которые заполняются соответствиями символов "кто -> во что преобразуется" (очевидно, что содержание секций представляет собой зеркальное отражение). При этом и исходный, и преобразуемый символы могут быть заданы почти любым образом — восьмеричными, шестнадцатеричными или десятичными их кодами, а то и просто символами, заключенными в одинарные кавычки (и даже без оных). Для удобочитаемости символы могут разделяться пробелами (а могут и не разделяться).

Еще одно из языково-зависимых средств, о котором необходимо упомянуть — проверка орфографии. Этой цели в FreeBSD служит стандартная программа ispell, устанавливаемая из портов или пакетов. В дополнение к ней в коллекции портов можно обнаружить и словари для проверки русских текстов — даже в двух экземплярах: rispell (известный словарь Константина Книжника) и rus-ispell (Александра Лебедева), как будто более новый (и активно обновляемый). Трудно сказать, какой из них лучше — я пользуюсь вторым вариантом.

В любом случае, оба порта устанавливаются стандартным образом, после чего не требуется никаких дополнительных настроек. Достаточно дать команду

```
$ ispell -d russian file
```

Замечу, однако, что, скомпилированные по умолчанию из системы портов, оба словаря позволяют выполнять проверку орфографии только для русскоязычных текстов в кодировке KOI8-R. Если необходимо проверять тексты и в иной кодировке, например, CP1251, словари следует пересобрать с соответствующими опциями. Не поручусь за rispell, но в документации к rus-ispell содержатся исчерпывающие сведения о том, как именно это сделать.

Программа ispell может быть встроена в текстовые редакторы (как это сделать для joe, было описано в предыдущей главе) или текстовые процессоры (например, в Lyx, о чем речь еще впереди).

Многообразие консольных программ, доступных в качестве портов FreeBSD, разумеется, не исчерпывается описанными. За чертой рассмотрения остались многочисленные средства форматирования и печати текстов (например, семейство утилит groff), и профессиональная система подготовки публикаций TeX, и электронные таблицы (типа sc, slsc, ss), и многое другое. В оправдание только опять скажу, что нельзя объять необъятное. И к тому

же это — именно те задачи, которые, если не заниматься ими систематически и профессионально, более эффективно решаются средствами графического режима системы X Window. Рассмотрение которых начинается со следующей главы.

Дополнительные источники

- Большинство затронутых в этой главе вопросов практически слабо освещены в литературе. И потому замечу только, что программу rusconv вместе с полной русской документацией для нее можно найти по адресу: **http://uucode.com/rusconv/**.
- А программа mapchan располагается на сайте фирмы Крон — **http://www.kron.vinnica.ua/free/download/**.
- Кроме этого, ссылки на все упомянутые программы, не входящие в систему портов и пакетов FreeBSD, можно найти на моем сайте — **http://freebsd.by.ru**, где я взял на себя смелость также разместить и исходные тексты труднодоступных программ, а также результаты своих упражнений в области консольных шрифтов, клавиатурных раскладок и карт соответствия.

Глава 20

Оконная система X и менеджеры окон

Функционирование приложений графического режима в FreeBSD обеспечивается оконной системой X (X Window System), вернее, ее открытой и свободной реализацией для Intel-совместимых платформ — XFree86. Строго говоря, это — не составная часть ОС FreeBSD, хотя она и входит в состав базового комплекта, именуемого Distributions: точно та же система XFree86 входит в состав любого дистрибутива Linux. Однако графические приложения играют ныне большую роль в FreeBSD, и потому целесообразно остановиться на устройстве системы XFree86 и программных средств, обеспечивающих управление программами графического режима — т. н. оконными менеджерами и интегрированными графическими средами. Именно это и составит предмет настоящей главы.

Оконная система X

Система XFree86 представляет собой одну из реализаций оконной системы X (X Window System, система Window System, или, в просторечии, просто "иксов"). Отличительные ее особенности ясны из названия — во-первых, она распространяется свободно и открыто, во-вторых — ориентирована на Intel-совместимые платформы. Сама же система X развивается на протяжении уже почти двадцати лет. У истоков ее — трое разработчиков, Роберт Шейфлер (Robert Sheifler), Джим Геттис (Jim Gettys) и Рон Ньюмен (Ron Newman). Современная реализация оконной системы X представлена 11-й версией в 6-й ее реализации, почему сокращенно и называется X11R6 — это имя фигурирует в структуре файловой системы FreeBSD (каталог /usr/X11R6).

К слову сказать, в названии системы X нет ничего таинственного. Просто ее разработчики на первом этапе развивали ее в русле одной из предшествующих графических систем, носившей название системы W. А литера X, как известно, в латинском алфавите следует непосредственно за W...

Что же представляет собой система X? Одно из определений (заимствованное из статьи Андрея Зубинского — **http://www.i2r.ru/static/278/out_9441.shtml**) гласит, что это — не зависящая от платформы растровая графическая оконная система, обеспечивающая прозрачное использование ее ресурсов с помощью сетевых соединений. Определение верное, но вызывающее больше вопросов, чем дающее ответов. Которые мы и попробуем наметить.

Независимость от платформы, пожалуй, в объяснениях не нуждается — как говорят в народе, "иксы" работают на всем, что загружается, и немного — на том, что загружаться не способно. По крайней мере, на любой машине, имеющей графическую видеосистему. И поверх почти любой ОС из ныне живущих — от UNIX любого рода чуть ли не до Windows. Однако именно в UNIX-системах (и FreeBSD здесь не исключение) "иксы" приняты в качестве стандартного графического интерфейса. Вернее, метаинтерфейса, на котором основываются собственно пользовательские интерфейсы.

Растровый характер системы X определяется тем, что она работает только с устройствами вывода, обеспечивающими попиксельное отображение информации, а не символьное, как текстовая консоль. Оконная же она потому, что имеет более высокий, нежели пиксел, уровень абстракции — "окно" (window), лежащий в основе всех интерфейсных элементов: и собственно окон, и кнопок, и ниспадающих меню, и строк ввода и даже обрамления вокруг окон.

Сетевой характер системы X Window обусловлен ее моделью — клиент-серверной, в которой взаимодействие между сервером и клиентами осуществляется (даже на локальной машине) за счет специального сетевого X-протокола (подмножества протоколов TCP/IP — именно поэтому поддержка последнего на уровне ядра необходима для функционирования системы X Window.

Именно сервер и клиент — первое из базовых понятий системы X. При этом понимание их кажется обратным общепринятому: в качестве сервера выступает аппаратно-зависимая система ввода/вывода (видеоподсистема, клавиатура и мышь), предоставляющая свои ресурсы клиентам — прикладным программам, обеспечивающим остальные функции. То есть сервером в общем случае является локальный терминал, тогда как клиентская часть может исполняться и на удаленной машине.

Впрочем, для персонального компьютера это не принципиально. Более важным здесь оказывается другое базовое понятие — дисплей. Под которым в данном случае понимается отнюдь не экран монитора — вернее, не только он один, но его комбинация с устройствами ввода и позиционирования, сиречь клавиатурой и мышью. В случае персонального компьютера такой X-дисплей целиком занимает одну виртуальную консоль — именно поэтому для запуска системы X Window необходимо, чтобы хотя бы одна из таковых

была доступна (т. е. на ней должен быть запущен какой-либо процесс, конкретно — процесс getty) и свободна (т. е. статус ее должен быть определен в как off), о чем говорилось в главе о стартовых файлах.

Один и тот же X-дисплей в состоянии поддерживать некоторое количество виртуальных экранов (отдаленный аналог виртуальных консолей текстового режима). Количество их ограничено максимальным значением целого числа, поддерживаемым системой (что для 32-разрядных компьютеров составит 231). Каждый же экран может содержать множество окон — отображаемых областей растровой памяти. Окна образуют строгие иерархии на основании простого правила — в каждом экране по умолчанию существует одно уникальное окно-родитель (корневое или root-окно), каждое окно должно иметь "родителя" и само может быть "родителем" для других окон. Окна могут создаваться, перерисовываться, удаляться, в них может выводиться текстовая и графическая информация.

Чтобы получить представление о базовых возможностях системы X, достаточно запустить их в чистом виде. Для чего достаточно дать следующую команду в консоли:

```
$ X
```

Файл /usr/X11R6/bin/X представляет собой ссылку на конкретный X-сервер, определяемый при конфигурировании системы в файле /etc/X11/XF86Config (для нынешних видеокарт это, с большой долей вероятности, будет XSVGA-сервер). После чего X-сервер запускается на ближайшей (из существующих и свободных) виртуальной консоли, а консоль, с которой X-сеанс был запущен, становится недоступной для текстового режима. При желании (и наличии свободных консолей) можно запустить несколько X-сеансов. Для этого достаточно дать ту же команду с указанием номера дисплея в качестве аргумента. Так, если первый сеанс по умолчанию займет нулевой дисплей, то для второго сеанса команда запуска будет выглядеть как

```
$ X :1
```

и т. д. Впрочем, много радости это не доставит, т. к. никаких действий в таком сеансе выполнить нельзя, кроме как наблюдать за перемещением крестообразного курсора по серо-решетчатому фону. Правда, есть еще и возможность прервать X-сеанс — комбинацией клавиш <Alt>+<Ctrl>+ +<Backspace>; к слову сказать, это универсальный способ выхода из графического режима, к которому приходится прибегать, если штатные средства выхода (о которых я скажу дальше) почему-либо не работают.

Чтобы осуществлять какие-либо действия в X-сеансе, он должен быть запущен совместно с какой-нибудь клиентской программой. Достигается это с помощью стартового сценария /usr/X11R6/bin/startx, который исполняет директивы, записанные в файле инициализации системы X — /usr/X11R6/lib/X11/xinit/xinitrc. Инициализационный файл аналогичного назначения

может быть создан и в домашнем каталоге пользователя — ~/.xinitrc: как обычно, общесистемные установки в этом случае "забиваются" пользовательскими.

Самый простой способ выполнить в X-сеансе что-либо полезное — это запустить совместно с ним программу эмуляции терминала. Таковых достаточно много, стандартной (и всегда присутствующей в любой системе) является xterm. Для его запуска создаем в домашнем каталоге вышеозначенный файл и вписываем в него одну-единственную строчку:

```
xterm
```

После этого даем команду

```
$ startx
```

и на том же решетчатом фоне наблюдаем терминальное окно. Это — не что иное, как эмулятор текстовой консоли в графическом режиме с запущенной в ней командной оболочкой. Как и в консоли, здесь можно давать любые команды и командные конструкции, запускать консольные приложения, а кроме того — и приложения графического режима. В частности, можно запустить программу GIMP и сделать скриншот получившейся картины (рис. 20.1).

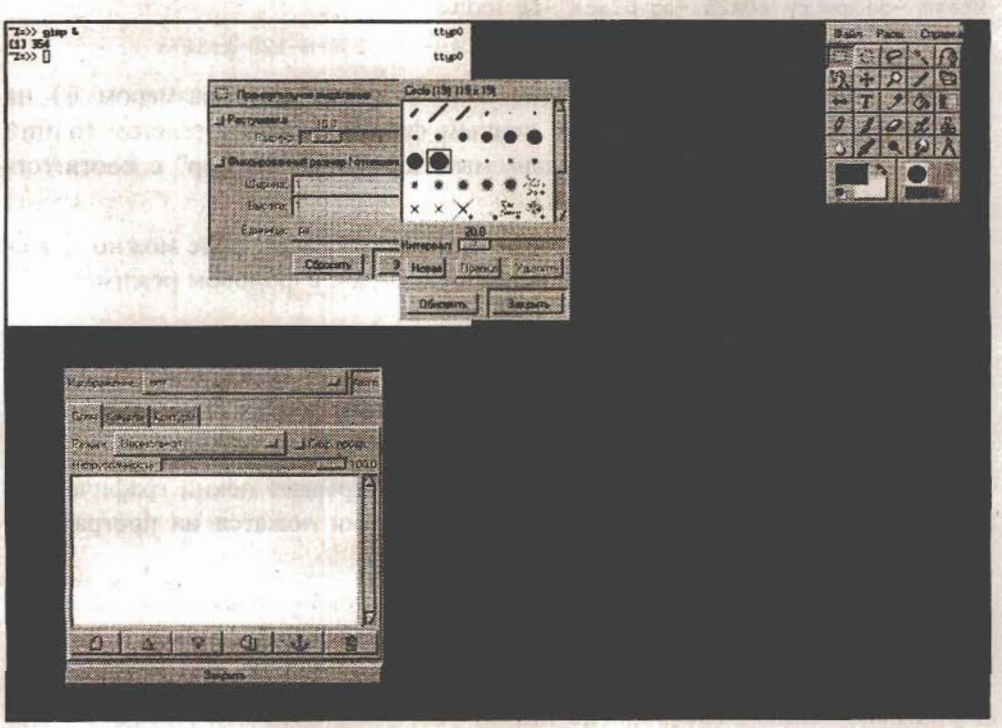


Рис. 20.1. X-сеанс с окном эмулятора терминала

Далее, мы получаем возможность закрытия X-сеанса, не прибегая к клавишным комбинациям. Для этого достаточно дать в командной строке команду

```
$ exit
```

которая вместе с выходом из оболочки вызовет и выход из системы X Window. Чего, впрочем, можно избежать, определив в файле ~/.xinitrc запуск терминальной программы в фоновом режиме:

```
xterm &
```

Впрочем, после выхода из терминальной программы все равно никаких действий выполнить не удастся, но такая возможность пригодится нам в дальнейшем.

Терминальному окну можно посредством соответствующих опций придать различные атрибуты: задать горизонтальный и вертикальный размер, определить цвет текста и фона, гарнитуру и размер шрифта и т. п. Например, если указанную выше строку файла ~/.xinitrc модифицировать следующим образом:

```
xterm -geometry 60x25 -bg black -fg white \
-fn -cronyx-courier-bold-r-normal--20-140-100-100-m-120-koi8-r
```

то при старте X-сеанса будет (рис. 20.2) открыто окно размером 60 на 25 символов (опция `-geometry`) с черным фоном и белым текстом (опции `-bg` и `-fg` соответственно), передаваемым шрифтом "Курьер" с соответствующими атрибутами (опция `-fn`).

Впрочем, и это доставит мало радости. Конечно, в таком окне можно запустить любое количество программ, используя запуск в фоновом режиме:

```
$ cmd &
```

Однако окно каждой последующей программы будет перекрывать предыдущие (как это видно на рис. 20.1) и получить к ним доступ станет невозможно. Не удастся также переместить окна или изменить их размер.

Таким образом, сам по себе X-сервер просто открывает некий графический дисплей. Все же функции по управлению окнами ложатся на программы-клиенты.

К этой теме я вернусь в следующем разделе, а пока — пару слов о еще одном важном компоненте системы X — разделяемых библиотеках и инструментальных наборах (в просторечии — тулкитах, toolkits), без которых построение и работа клиентских приложений невозможны.

Рис. 20.2. Окно терминала с атрибутами, определенными в файле ~/.xinitrc

Базовая библиотека, обязательно входящая в комплект любой реализации системы X Window — Xlib. Она обеспечивает открытие дисплея и взаимодействие с ним любой клиентской программы, создание и уничтожение окон, а также их отображение вместе с некоторыми простыми атрибутами.

Теоретически рассуждая, программы для системы X могут создаваться (и подчас создаются на практике) с использованием только базовой библиотеки. Однако, по-видимому, это требует значительных усилий со стороны разработчиков и применяется редко. Большинство распространенных программ создается с использованием высокоуровневых библиотек (и сопровождающих их тулкитов), из которых главными являются:

- Motif, до недавнего времени имевшая статус коммерческой, но ныне реализованная в открытых (OpenMotif и Lesstif) вариантах;
- Qt, статус которой не вполне определен (она бесплатна при некоммерческом использовании и весьма даже платна — при применении в коммерческих целях);
- Gtk — истинно открытая и бесплатная библиотека, разработанная в рамках проекта GNU.

Дискуссию о сравнительных достоинствах и недостатках этих библиотек я развивать не буду. Однако пользователю необходимо понимать, что наличие этих библиотек в системе — почти непременное условие для установки (и — функционирования) основанных на них приложений. Правда, Motif-приложения иногда собираются в статически связанном виде (т. е. библиотечные функции встраиваются при компиляции в исполняемые модули) — в этом случае необходимости в самой по себе библиотеке нет. Однако для библиотек Qt и Gtk такого, насколько мне известно, почти не практикуется — а ведь на них основаны такие популярные интегрированные среды, как, соответственно, KDE и GNOME, а также все их приложения.

Об оконных менеджерах

Как было сказано чуть выше, чтобы получить возможность управления окнами, над X-сервером следует запустить одну из клиентских программ, именуемых оконными менеджерами, менеджерами окон или, менее точно, их диспетчерами (Window Manager). Именно они, как не трудно догадаться из названия, отвечают за манипуляции над открытыми окнами — их закрытие, масштабирование, минимизацию, перемещение, а также переключение между окнами. Кроме того, многие оконные менеджеры обеспечивают и дополнительные возможности — вызов контекстных меню, различные инструментальные панели и пиктограммы запуска программ на рабочем столе, а также поддержку тех самых виртуальных экранов (рабочих столов), о которых говорилось ранее. В компетенции менеджера окон — и средства для выхода из графического режима штатным образом, без использования пресловутых клавишных комбинаций.

Вообще, именно от оконного менеджера зависит функциональность рабочей среды в X-сеансе, почему вопросу выбора такой программы следует уделить достаточное внимание. Однако прежде надо ознакомиться с возможностями разных менеджеров окон. А для этого — перво-наперво разобраться, как их запускать. Сделать это можно в командной строке эмулятора терминала отдачей соответствующей команды, обычно совпадающей с именем оконного менеджера. Так, командой

```
$ fvwm2
```

будет запущен оконный менеджер с этим названием (цифра 2 в имени указывает на номер версии), как это показано на рис. 20.3. Можно видеть, что родительское окно терминала (в левом верхнем углу рабочего стола) после запуска оконного менеджера волшебным образом преображается: на нем появляются строка заголовка, управляющие пиктограммы (максимизации, минимизации, закрытия и т. п.), линейки прокрутки и прочие атрибуты. С рабочего стола по щелчку кнопок мыши становятся доступными контекстные меню. А самих рабочих столов оказывается четыре, переключение

между ними обеспечивается инструментальной панелью (в нижнем правом углу). Если же выйти (посредством соответствующего пункта контекстного меню) из оконного менеджера, внешний вид X-сеанса возвращается к первозданному состоянию.

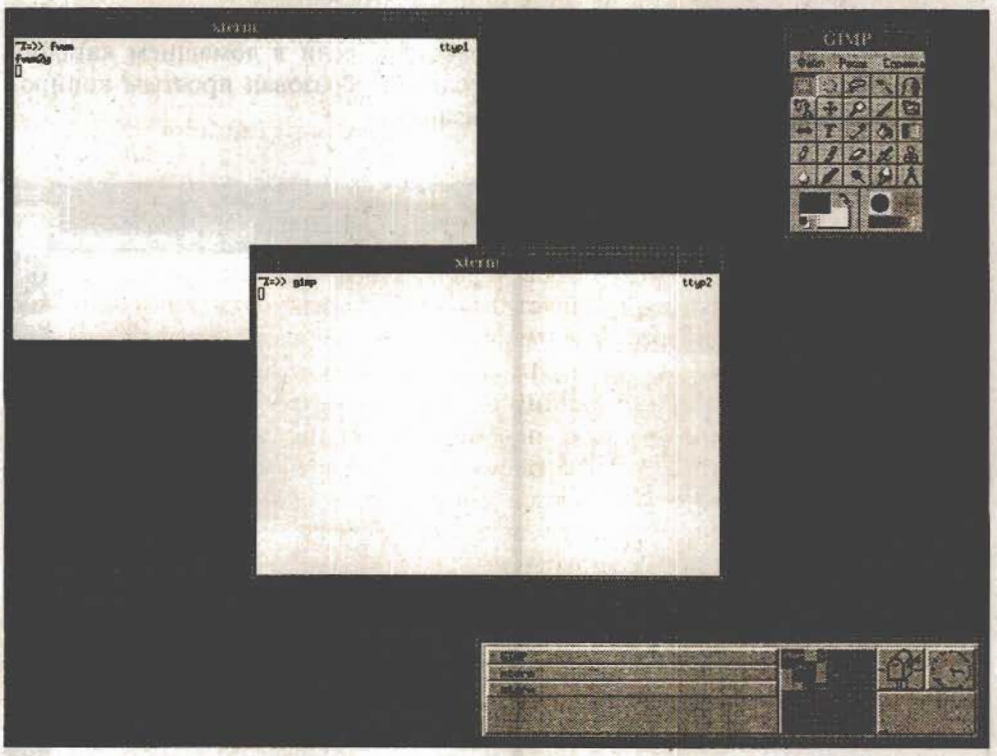


Рис. 20.3. Запуск менеджера окон из командной сроки эмулятора терминала

Есть, однако, более простой способ запуска менеджера окон — вписать соответствующую команду в один из файлов инициализации, общесистемный или пользовательский. Именно так сделано по умолчанию в FreeBSD: если устанавливать систему X Window в качестве компонента Distributions (а не из пакетов или системы портов), одновременно с ней устанавливается и оконный менеджер twm (старейший из менеджеров окон для системы X Window), а в общесистемный файл инициализации /usr/X11R6/lib/X11/xinit/xinitrc добавляются строки, обеспечивающие его запуск:

```
twm &
xclock -geometry 50x50-1+1 &
xterm -geometry 80x50+494+51 &
```

```
xterm -geometry 80x20+494-0 &
exec xterm -geometry 80x66+0+0 -name login
```

Можно видеть, что вместе с менеджером twm в фоновом режиме запускается также ряд терминальных окон и часы (рис. 20.4). Последняя из команд — `exec` — запускает в активном режиме еще одно терминальное окно. Закрытие его (командой `exit` в командной строке) приводит к выходу из оконного менеджера. Разумеется, все это сработает, если в домашнем каталоге пользователя нет файла ~/.xinitrc (или если он образован простым копированием общесистемного файла инициализации).

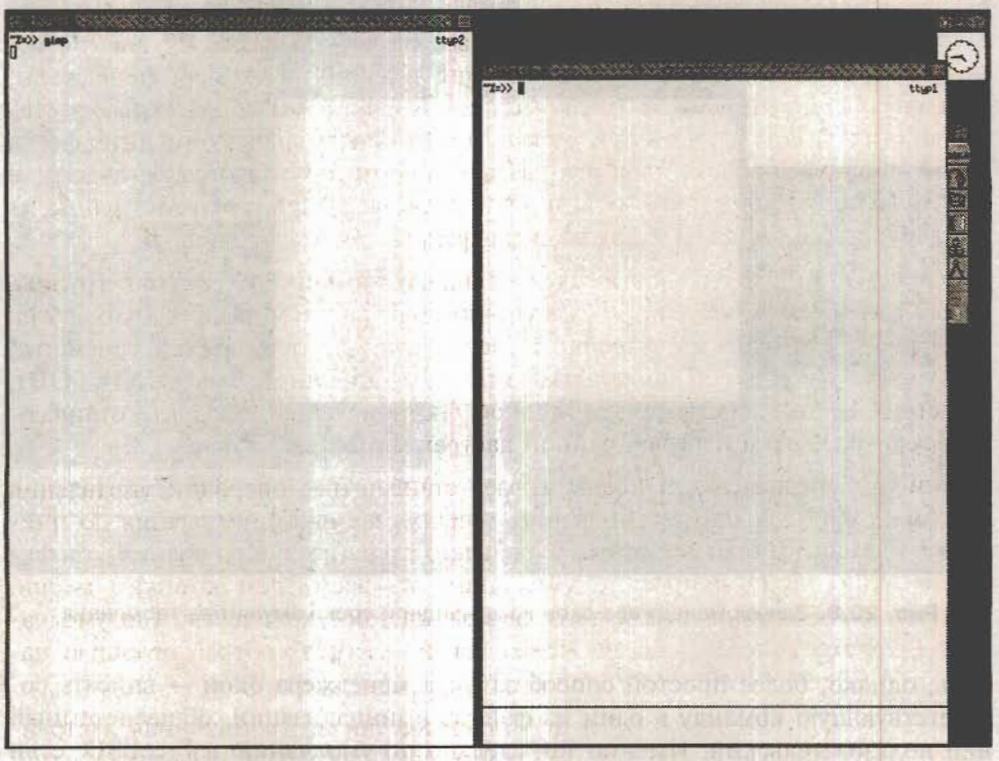


Рис. 20.4. Менеджер окон twm — вариант по умолчанию в FreeBSD

Вообще говоря, форма запуска оконных менеджеров может быть различной. Так, в своем "домашнем" .xinitrc пользователь может прописать команду:

```
exec имя_менеджера
```

предварив ее требуемыми директивами для запуска терминальных окон, программы, обеспечивающей работу скроллирующего колеса мыши, и т. п. А на первых порах, для знакомства с возможностями различных окноуправ-

ляющих программ, можно поместить их все, закрыв комментариями, которые последовательно снимать по мере изучения отдельных экземпляров этого класса.

Потому что FreeBSD — не тюрьма народов и не обязывает всех и каждого пользоваться оконным менеджером по умолчанию. Тем паче, что штатный twm а) достаточно стар, б) функционально ограничен и в) далек от эстетического идеала (по крайней мере, моего). И потому в виде портов и пакетов в FreeBSD доступно множество менеджеров окон — одно перечисление их заняло бы немало страниц, а уж на их описание не хватило бы ни сил, ни времени. Вместо этого я попробую дать общий обзор менеджеров окон с характеристикой наиболее ярких представителей.

С точки зрения подхода к интерфейсу все их можно разделить на три основные группы: менеджеры, развивающие традиции NextStep, менеджеры, имитирующие интерфейсы различных иных систем, и менеджеры, реализующие идеи True UNIX GUI, сиречь истинного графического интерфейса UNIX-систем. Впрочем, идеи эти разработчиками понимаются по-своему, и потому единственное, что объединяет представителей третьей группы, — непохожесть как на любые другие интерфейсы, так и друг на друга.

Впрочем, по справедливости следует начать именно с третьей группы. И здесь первым в очереди — уже упомянутый ранее менеджер окон fvwm. Во-первых, это один из старейших (после twm) представителей семейства, во-вторых, именно он полагается многими эталоном True UNIX GUI, в-третьих — это действительно полнофункциональный продукт, отличающийся практически неограниченной настраиваемостью.

Оконный менеджер fvwm поддерживает стандартные операции управления окнами, такие как масштабирование, максимизация, минимизация до пиктограммы или строки заголовка. С рабочего стола доступны разные контекстные меню, привязанные по умолчанию ко всем трем кнопкам мыши. Самих рабочих столов может быть неограниченное количество (по умолчанию — четыре), переключение между ними — через соответствующую панель или клавишными комбинациями.

Главная отличительная черта fvwm — практически неограниченная настраиваемость, осуществляемая, правда, почти исключительно редактированием конфигурационных файлов; настройке поддаются более 200 параметров. Кроме того, он позволяет подключать внешние модули, большое количество которых доступно в Интернете.

К вопросу о True UNIX GUI я еще вернусь. А пока — о менеджерах линии NextStep. Наиболее яркий их представитель — WindowMaker. Перед первым его запуском требуется настройка — путем запуска файла /usr/X11R6/bin/wmaker.inst, который создает в рабочем каталоге пользователя подкаталог ~/GNUstep/ с основными конфигурационными файлами.

Первое, что привлекает внимание после запуска WindowMaker, — крупные трехмерные кнопки вдоль правого края экрана, образующие управляющую панель, именуемую доком (**Dock**). Двойной щелчок левой кнопкой мыши на большинстве входящих в панель кнопок запускает приложения либо конфигурационные утилиты. Одинокая кнопка слева вверху (**Clip**) выполняет двойную функцию: переключения виртуальных экранов (**Workspace**) и прикрепления приложений.

Все остальное пространство экрана свободно и представляет собой рабочий стол. Щелчок на нем правой кнопкой мыши вызывает контекстное меню, а правой — список запущенных приложений.

Из контекстного меню создаются (в любом количестве) и удаляются новые виртуальные экраны, между которыми потом можно переключаться уголками на кнопке **Clip** (Клип).

Для запуска приложений, кроме кнопок на панели и стандартного терминала, используется также контекстное меню рабочего стола. Для чего в нем присутствует пункт **Run**, запускающий мини-терминал.

Базовые настройки осуществляются через кнопку **WMPefs** (Предпочтения). Она вызывает окно конфигурирования WindowMaker, где можно настроить позицию открываемых окон, фокусировку окна, расположение минимизированных окон и их выравнивание, размер экранных кнопок, пути поиска графических файлов для пиктограмм и фоновых изображений, навигацию по виртуальным экранам и т. п. Имеются также средства настройки внешнего вида иконок, окон, "горячих" клавиш, свойств мыши и пр. А главное — настройка меню приложений.

В меню приложений (т. е. **Applications** (Приложения), вызываемые из контекстного меню рабочего стола правой кнопкой мыши) можно добавить (и, разумеется, удалить) пункты первого уровня, которые могут содержать подменю любого уровня вложенности. И с любым количеством пунктов, которым приписываются любые команды — достаточно указать ее имя (или полный путь).

Дополнительные настройки внешнего вида WindowMaker можно выполнить из всплывающего меню. Здесь определяются темы из фиксированного набора, стили окон и меню, фоновое оформление — сплошная или градиентная цветовая заливка, изображения в любом распространенном растровом формате. Отсутствует только возможность изменения базового шрифта — это делается исключительно вручную.

WindowMaker позволяет точную ручную настройку путем правки конфигурационных файлов. Они, в количестве пяти штук, автоматически копируются в каталог ~/GNUstep/Defaults при первичной настройке.

В файле ~/GNUstep/Defaults/WMGLOBAL, как явствует из его названия, определяются наиболее общие параметры. В частности, только здесь можно переопределить шрифт для элементов рабочего стола.

В файле ~/GNUstep/Defaults/WindowMaker указываются стили меню и иконок, цветовая палитра, пути для иконок и фоновых рисунков и многие другие параметры, определяемые через **WMPrefs** (Предпочтения). Через который их и лучше изменять при необходимости.

Файл ~/GNUstep/Defaults/WMRootMenu описывает содержание меню **Applications** (Приложения), вызываемого с рабочего стола и также определяемого через файл ~/GNUstep/Defaults/WMPrefs. Содержание файла ~/GNUstep/Defaults/WMWindowAttributes ясно из названия. Ну, а файл ~/GNUstep/Defaults/WMState — описание панели **WMDock** (Док) и рабочих мест. Именно его легко отредактировать для внесения новых приложений в **WMDock** (Док).

В целом возможных настроек почти достаточно для удовлетворения личных эстетических идеалов и для внесения разнообразия в производственный процесс.

Наконец, группа менеджеров-имитаторов. Среди них — программы, более или менее точно воспроизводящие интерфейс AmigaOS, Mac OS, системы Plan9. Но наиболее многочисленны, разумеется, кальки Windows 9*x*. Из которых наиболее функциональным менеджером окон видится IceWm.

Внешний вид IceWM — весьма Windows-образный: в нижней части экрана панель запуска приложений, она же — переключатель виртуальных экранов и панель задач с аналогом кнопки **Start** (**Пуск** из Windows 9*x*), правее которой — кнопки для переключения задач и запуска приложений и кнопки переключения виртуальных экранов (по умолчанию четырех). Кроме того, в панели можно открыть строку мини-терминала.

Щелчок левой кнопкой мыши на рабочем столе вызывает список запущенных задач и виртуальных экранов, правой — меню, аналогичное по содержанию таковому стартовой кнопки; к средней кнопке привязан список открытых окон.

Средства визуального конфигурирования системы сводятся к выбору из фиксированного набора тем. Основные же настройки выполняются редактированием четырех конфигурационных файлов — preferences, menu, toolbar и winoptions, которые следует скопировать из каталога /usr/X11R6/share/icewm/ в домашний каталог пользователя — ~/.icewm.

В файле /usr/X11R6/share/icewm/preferences все пункты (а их там чуть ли не две сотни) по умолчанию отключены. Снимая значок комментария и устанавливая переключатели (1/0) в соответствующие значения, можно настроить массу параметров активизации окон, определить шрифты и цвета всех интерфейсных элементов, установить состав иконок на панели и многое другое. Очень интересная опция — возможность установки двойной высоты управляющей панели.

В файле /usr/X11R6/share/icewm/menu, как не сложно догадаться, можно настроить состав меню, вызываемого по стартовой кнопке. Пункты меню вписываются (в текстовом редакторе) в следующем формате: название приложения, имя иконки и имя исполняемого файла. Чтобы объединить несколько программ в одну папку (т. е. сделать иерархическое меню), указывается слово `menu`, после которого, в фигурных скобках, перечисляются программные компоненты.

В файле /usr/X11R6/share/icewm/toolbar, естественно, можно вывести иконки запуска приложений непосредственно с управляющей панели. Что делается абсолютно так же, как и настройка меню. Наконец, в файле /usr/X11R6/share/icewm/winoptions можно настроить параметры всяческих приложений.

Таким образом, IceWM сочетает в себе эффективные средства запуска программ, достаточное количество виртуальных экранов с удобными способами перемещения между ними и гибкость настроек. Замечу также, что в дополнение к нему в коллекции портов имеется утилита icepref для интерактивной настройки этого оконного менеджера.

В заключение раздела — несколько слов о так называемых "легких" (или "детских") оконных менеджерах. Основная их особенность — предельная нетребовательность к ресурсам, из чего обычно вытекает и ограниченная (хотя часто — вполне достаточная) функциональность. Программ этого класса среди портов FreeBSD достаточно много, задержусь лишь на одной — flwm.

Этот оконный менеджер не поддерживает ни пиктограмм запуска, ни управляющих панелей. Единственное средство запуска приложений — контекстное меню, вызываемое щелчком любой из кнопок мыши. Содержание его определяется через каталог ~/.wmx/, содержащий подкаталоги, соответствующие пунктам меню, с файлами, отвечающими подпунктам. Файлы эти представляют собой символические ссылки на исполняемые модули запуска соответствующих приложений — правда, создавать их нужно вручную. Вследствие такой простоты flwm может быть полезен при ограниченных ресурсах.

Более подробные сведения об оконных менеджерах можно получить из дополнительных источников информации. От рекомендаций по выбору я воздержусь — это дело слишком субъективное, можно даже сказать, интимное. Вместо этого в заключение этой главы будут изложены мои представления об истинном графическом интерфейсе UNIX.

Однако сначала нужно сказать, что, кроме оконных менеджеров, функциональность X-сеанса может быть достигнута посредством т. н. интегрированных сред.

Об интегрированных средах

Пожалуй, наиболее интенсивно развивающееся направление в системе открытого софта — это интегрированные визуальные (называемые также не совсем точно графическими) среды — KDE и GNOME, — и приложения для них. Роль этих сред, особенно KDE, для роста популярности UNIX-подобных систем переоценить трудно. Именно они помогли и помогают недавнему Windows-мигранту преодолеть барьер отчуждения от мира командных строк и текстовых редакторов.

В отличие от оконных менеджеров, функция которых — в первую очередь (а иногда и исключительно) управление окнами, а все прочие возможности (средства запуска программ, управление виртуальными рабочими столами и т. п.) являются как бы дополнительными, интегрированные среды изначально создавались как комплексные средства обеспечения работы в графическом режиме. И потому их неотъемлемые компоненты — не только средства управления всего и всем, но и некоторые наборы приложений. Причем менеджер окон — не всегда обязательный компонент интегрированной среды: GNOME, например, собственного менеджера окон не имеет, позволяя использовать любой из списка совместимых (по умолчанию — Sawfish).

Подробная характеристика KDE и GNOME не входит в мои задачи — это могло бы составить предмет отдельной книги. Замечу только, что в текущих своих версиях и KDE, и GNOME примерно равны по функциональности. Обе эти среды имеют в своем составе интегрированные средства настройки, многочисленные утилиты общесистемного и пользовательского назначения, обширный (и постоянно растущий) список прикладных программ широкого применения (на некоторых из них я остановлюсь в *главе 22*). Примерно равны у них и требования к аппаратным ресурсам (далеко превосходящие таковые самого "навороченного" менеджера окон). На мой взгляд, KDE несколько быстрее и устойчивее GNOME, а также (но это уж совсем субъективно) — много удобнее и эстетически приятнее (но это уж совсем субъективно).

Так что выбор между KDE и GNOME — дело вкуса и случая. Однако укажу на некоторые факты, которые косвенно на такой выбор могут повлиять.

Первое — и KDE, и GNOME для установки требуют обязательного наличия в системе "родной" библиотеки — Qt и Gtk соответственно. Необходимы они и для приложений этих сред. Из которых в KDE имеется по меньшей мере одна уникальная по функциональности программа — Konqueror, представляющая собой файловый менеджер, интегрированный не только с браузером, но и с полноценным эмулятором терминала. Иными словами — это универсальное средство для навигации и управления файловыми системами, как локальными, так и удаленными. И если у пользователя есть потребность в таком средстве — выбор в пользу KDE становится однозначным.

С другой стороны, та же библиотека Gtk, на которой основана среда GNOME, используется программой GIMP (собственно, для нее-то она и была первоначально разработана). А GIMP — это не имеющий даже близких (в UNIX-системах, конечно) аналогов растровый графический редактор, абсолютно незаменимый при любых работах с изображениями, выходящими за пределы элементарной правки яркости/контрастности. И потому если такие задачи перед пользователем стоят — общность библиотеки может склонить его в пользу GNOME.

Кроме того, GNOME в последнее время обзавелся (почти) функционально полными и работоспособными офисными приложениями — текстовым процессором AbiWord и электронной таблицей Gnumeric, также не способными работать без библиотеки Gtk. Интегрированный же офисный пакет из среды KDE — KOffice, несмотря на теоретически большую функциональность, работоспособным (в текущих версиях) может считаться с большой натяжкой: редкий пользователь может проработать с ним более половины часа до падения (пакета, разумеется). Хотя, повторяю, сама по себе среда KDE более устойчива, чем GNOME.

Конечно, никто не запрещает иметь на машине обе библиотеки и обе среды с их приложениями. Тем более, что подавляющее большинство этих приложений могут быть запущены и из любого оконного менеджера (достаточно только факта наличия среды и библиотеки в системе). Однако это ведет не только к трате дискового пространства (при современных объемах дисков сие — грех невеликий), но и предъявляет дополнительные требования к памяти, а они и у KDE, и у GNOME немалы даже по отдельности. Однако в ряде случаев иного выбора не остается — тем более, повторюсь, необходимости в запуске любой из этих сред для использования их приложений нет.

Однако возможность выбора интегрированной среды не исчерпывается KDE и GNOME. Потому что существует еще один претендент на этот титул — среда XFce. По своим аппаратным запросам она существенно ближе к оконным менеджерам, а интегрированность ее выражена в наличии ограниченного (но продуманного и функционального) набора встроенных утилит. Именно на этой среде я и хотел бы задержать внимание читателей.

Интегрированная среда XFce (The Cholesterol Free Desktop Environment, что применительно к случаю я перевел бы как Настольная Среда для Холериков) показывает ее соответствие названию: все работает весьма быстро и производит впечатление импульсивности. Среда включает в себя следующие компоненты:

- XFwm — собственно оконный менеджер;
- *Xfce-Toolbar — панель управления;*
- XFTree — файловый менеджер;
- xfterm — эмулятор терминала;

- XFPager — средство управления виртуальными рабочими столами;
- XFMouse — утилита для настройки мыши;
- XFSound — утилита для воспроизведения звуковых сигналов;
- XFbd — средство управления фоновыми изображениями;
- xfclock — часы.

Менеджер окон полностью встроен в окружение среды, предоставляя однородный интерфейс для всех ее приложений. Доступ к его функциям осуществляется через контекстные меню, вызываемые щелчками кнопок мыши. Назначение и состав этих меню зависит от версии и сборки, однако в целом они позволяют управлять окнами и иконками, настройкой мыши, фона и звука, перемещаться между окнами запущенных приложений, изменять атрибуты существующих окон и т. д. Средств запуска программ с рабочего стола (типа мини-терминала) не обнаруживается. Хотя такое средство включено в состав файлового менеджера. Так что, помимо главной панели, для запуска программ можно использовать только какой-либо стандартный эмулятор терминала (по умолчанию в нее включен xterm).

На панели управления можно видеть серию кнопок — от системных часов до **Help**. В центре — переключатель виртуальных экранов, обрамленный кнопками защиты (выбор его вызывает черный экран с предложением ввести пароль пользователя), информации о программе, настройки и выхода. Прочие же кнопки (количество их варьирует от сборки к сборке) вызывают серии приложений — и совсем не обязательно штатных для XFce. Стрелка над каждой из этих кнопок вызывает дополнительные меню, уже с кнопками вызова непосредственно программ; в каждом из вторичных меню непременно присутствует кнопка с предложением добавить новую кнопку. Процесс этот очень прост: щелчок правой кнопкой вызывает панель, в которой вводятся (путем просмотра дерева каталогов или просто вручную) пути до исполняемого файла и файла иконки, а также название (метка) программы.

Среди включенных в главную панель обычных приложений и утилит обычно обнаруживаются — какой-либо эмулятор терминала, браузер, калькулятор (совсем не обязательно имеющие место в системе), помянутые ранее специфичные для XFce программы. Из последних наибольшее внимание привлекает файловый менеджер XFTree — штатное средство для управления файлами из интегрированной среды XFce. Он имеет древовидную структуру, которая может разворачиваться и сворачиваться, несколько напоминая XTree Gold (был встарь такой файловый менеджер — соперник командира Нортона). Для выделения файла или каталога требуется фиксация и щелчок левой кнопкой. После этого щелчок правой кнопкой вызывает обширное контекстное меню, позволяющее: открыть каталог в новом окне; создать каталог или файл, удалить и переименовать их; выделить элемент или все, а также развыделить (так в оригинале) их; найти файл, правда, только в ката-

логе, на котором зафиксирован курсор; просмотреть свойства файла или каталога, в том числе и права доступа, и при наличии необходимости и полномочий — изменить их. А также несколько действий по переходу — к закладке, на уровень выше и т. д.

Копирования и перемещения файлов в меню не предусмотрено. Первое осуществляется просто перетаскиванием выделенных элементов (множественное выделение — как в Windows, мышью при нажатых <Shift> или <Ctrl>), в том числе и между разными окнами XFTree (их можно открыть сколько угодно); для перемещения нужно сначала выделить файл или их группу, а затем прибегнуть к перетаскиванию при нажатой клавише <Shift>. Здесь же — и пункт запуска программ, вызывающий мини-терминал, который хранит историю команд, правда, только в текущем сеансе и в пределах текущего дисплея. Кроме того, если выделить одиночный файл, то по щелчку правой кнопкой доступен пункт **Открыть с**, вызывающий ту же панель запуска программ с той же их историей.

В целом XFTree показался мне соответствующим стилю среды XFce (хотя, при наличии установленной библиотеки Gtk может функционировать и в любой другой оконной среде). Он производит впечатление очень быстродействующего: разворачивание многоуровневых каталогов или поиск от корневого каталога осуществляется практически мгновенно. Однако для повседневной работы не очень удобен, нет никаких интерактивных настроек цвета, шрифтов и прочего (все это устанавливается через конфигурирование XFce в целом), кроме невозможности перемещения файлов, нет также встроенного или подключаемого архиватора.

Оконный менеджер Blackbox

По мере обвыкания в UNIX-мире интегрированные среды превращаются в своего рода раздражающий фактор. Этому способствует и изобилие навязываемых для комплекта приложений, далеко не все из которых — лучшие в своем классе, и сложность и непрозрачность ручных настроек, и не всегда оправданная функционально требовательность к ресурсам. А главное, пожалуй, — излишне громоздкий интерфейс, снижающий не столько даже объективное (т. е. аппаратно-обусловленное) быстродействие, сколько замедление работы вследствие пресловутого человеческого фактора (по простому это можно назвать — "глаза разбегаются").

Кроме того, если KDE или GNOME воспринимается как органичный компонент ориентированных на конечного пользователя дистрибутивов Linux, то на фоне строгой простоты BSD-систем они выглядят чужеродными включениями. И возникает естественное желание вернуться к истокам, т. е. к тому, что было удачно названо Виктором Вагнером True UNIX GUI — Истинному графическому интерфейсу UNIX-систем.

Правда, понятие это само по себе неоднозначно и требует некоторого освещения. Действительно, какова основная цель использования GUI любого рода? Бытует мнение, что их главное (и чуть не единственное) предназначение — открыть много-много терминальных окон для работы в командной строке. Что, несмотря на некоторый экстремизм, не лишено резона. Ведь в UNIX-мире именно командная оболочка — наиболее эффективный инструмент для решения 90% пользовательских задач.

Однако остается еще 10% задач, которые требуют какого-никакого графического режима. Ведь пока еще никто не додумался, как ретушировать фотоизображения в окне терминала. И если в последнем можно строить векторные чертежи или 3D-объекты, то для визуализации результатов графический режим все равно потребуется. Как и соответствующая программа, которую, очевидно, придется запускать из графической среды. Из чего и следует главная функция GUI — обеспечить запуск программ, управление открытыми приложениями и взаимодействие между ними. Все же остальное может рассматриваться как "бантики".

Хотя роль "бантиков" преуменьшать также не следует. Если текстовую консоль при всем желании невозможно изуродовать, то с графическими интерфейсами это — плевое дело. И потому эстетика GUI — далеко не последнее, что требуется для эффективного использования его функций. А поскольку эстетика для каждого своя, настраиваемость среды приобретает самоценность. Причем настраиваемость, осуществляемая простыми и понятными средствами.

Что же такое True UNIX GUI с изложенных позиций? Как уже говорилось, в качестве такового часто рассматривается fvwm. Однако, на мой взгляд, есть не менее законный претендент на этот титул — оконный менеджер Blackbox.

Это — предельно простая по устройству и очень компактная программа (размер исполняемого файла менее 200 Кбайт, в полтора-два раза меньше так же не особо "тяжелых" IceWM или WindowMaker). Для своей установки и работы она не требует никаких специальных высокоуровневых библиотек типа Qt или Gtk. Тем не менее, возможностей для пользователя она предоставляет вдоволь. Здесь и (рис. 20.5):

- неограниченное количество виртуальных рабочих столов (**Workspaces**), создаваемых при необходимости легким движением мыши;
- удобные средства навигации между рабочими столами и окнами открытых приложений; таковыми являются, во-первых, небольшая (и к тому же способная скрываться автоматически) панель управления (**Toolbar**), во-вторых — контекстное меню, доступное по щелчку средней кнопкой мыши;
- легко доступное (по щелчку правой кнопкой мыши) и элементарно настраиваемое контекстное меню для запуска приложений;

- возможность минимизации, максимизации и закрытия окон приложений, сворачивания их до строки заголовка; при этом соответствующие управляющие элементы расположены удобно и привычно.

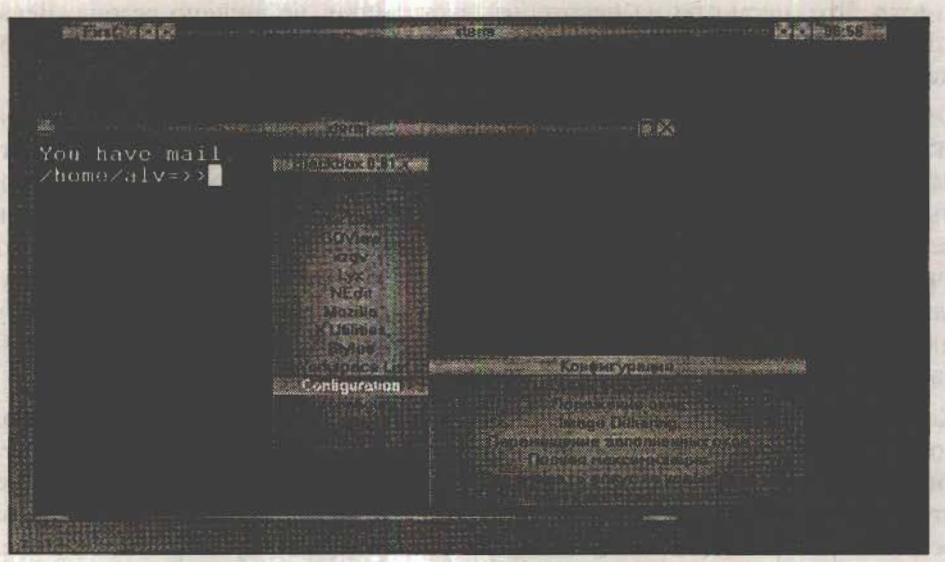


Рис. 20.5. Оконный менеджер Blackbox — общий вид

О возможностях настройки Blackbox следует сказать особо. Элементарное его конфигурирование выполняется непосредственно с рабочего стола. Здесь можно изменить модели фокусировки окон (по щелчку мышью, по перемещению курсора, автовсплытие) и их открытия (каскадное, "умное", упорядоченное в любом направлении), положение панели управления и некоторые другие параметры (рис. 20.6, 20.7).

Рис. 20.6. Настройка панели управления

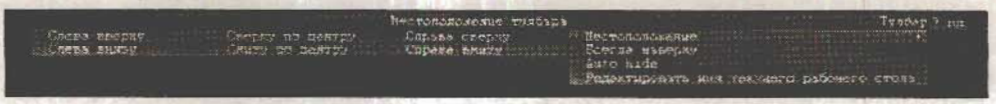


Рис. 20.7. Определение положения панели управления

Через главное же контекстное меню настраивается и внешний вид Blackbox — с помощью выбора одного из имеющихся в комплекте стилей (рис. 20.8). В это понятие входят цветовая гамма и характер заливки (сплошная, градиентная) рабочего стола и прочих интерфейсных элементов, шрифт меню, оконных титулов и пр. Правда, свойства эти — сквозные, приписать индивидуальный стиль отдельному рабочему столу не получается. Как нельзя и использовать фоновые изображения для них.

Рис. 20.8. Выбор стилей из главного контекстного меню

Более тонкие настройки выполняются традиционным способом — правкой конфигурационных файлов. Главным из них является ~/.blackboxrc, который при первом же запуске Blackbox автоматически образуется в домашнем каталоге пользователя. Именно в нем прописаны модели фокусировки и открытия окон, положение панели управления, количество и имена рабочих столов.

Кроме того, в ~/.blackboxrc указываются пути к файлу, определяющему состав главного меню, и к стилевому файлу, ответственному за внешний вид. По умолчанию это /usr/X11R6/share/Blackbox/menu и /usr/X11R6/share/Blackbox/styles/имя_стиля соответственно. Редактированию пользователем они, естественно, не поддаются. Однако ничто не мешает скопировать целиком /usr/X11R6/share/Blackbox/ в свой пользовательский каталог

```
$ cp -R /usr/X11R6/share/Blackbox/ ~/.Blackbox/
```

и развлекаться с ним в свое удовольствие. В частности, переопределить состав главного меню можно просто в текстовом редакторе. Для этого достаточно внести в него, после строки

```
[begin] (Blackbox 0.61.x)
```

требуемые приложения в следующем формате:

```
[exec] (rxvt)      {rxvt}
```

где компонент в круглых скобках — метка пункта меню, а в фигурных — собственно имя исполняемого файла. Можно организовать и субменю любой степени вложенности:

```
[submenu] (Graphics)
  [exec] (XV)            {xv}
  [exec] (The GIMP)      {gimp}
  [exec] (Image Magick) {display}
[end]
```

Ничуть не сложнее и переопределить стили внешнего вида — взяв за основу любой подходящий или создав свой собственный с нуля по образу и подобию имеющихся. Каковые, к слову сказать, сделаны очень строго и с большим вкусом — я, например, заменил в используемых мной стилях только шрифты — на более подходящие моему зрению.

Ну и, конечно, нужно не забыть изменить пути к соответствующим файлам в ~/.blackboxrc:

```
session.styleFile:  ~/.blackbox/Styles/currentstyle
session.menuFile:   ~/.Blackbox/menu
```

После этого вопрос с настройками Blackbox можно считать решенным.

При всех многочисленных достоинствах Blackbox (компактности, близком к оптимальному соотношению между возможностями настройки и их простотой, удобстве использования и исключительной элегантности), нельзя не отметить некоторой его функциональной ограниченности, выраженной в отсутствии мини-терминала для быстрого запуска программ, сложности клавиатурной навигации по открытым окнам и рабочим столам (клавишные комбинации, описанные в документации к программе, далеко не всегда работают). Ну, и эстетическое впечатление ослабляет отсутствие возможности определить фоновые изображения для рабочих столов.

Однако эти ограничения полностью компенсируются многочисленными дополнительными компонентами, доступными в FreeBSD в качестве самостоятельных портов. Среди них:

- bbrun, удобная строка мини-терминала, подобная mini-cli из KDE, поддерживающая историю команд в виде всплывающего меню (рис. 20.9);

- bbpager — средство визуального управления окнами, открытыми на виртуальных рабочих столах;
- bbkeys — система определения "горячих" клавиш;
- bbrb — менеджер фоновых изображений.

Рис. 20.9. Мини-терминал и менеджер виртуальных десктопов — дополнительные модули Blackbox

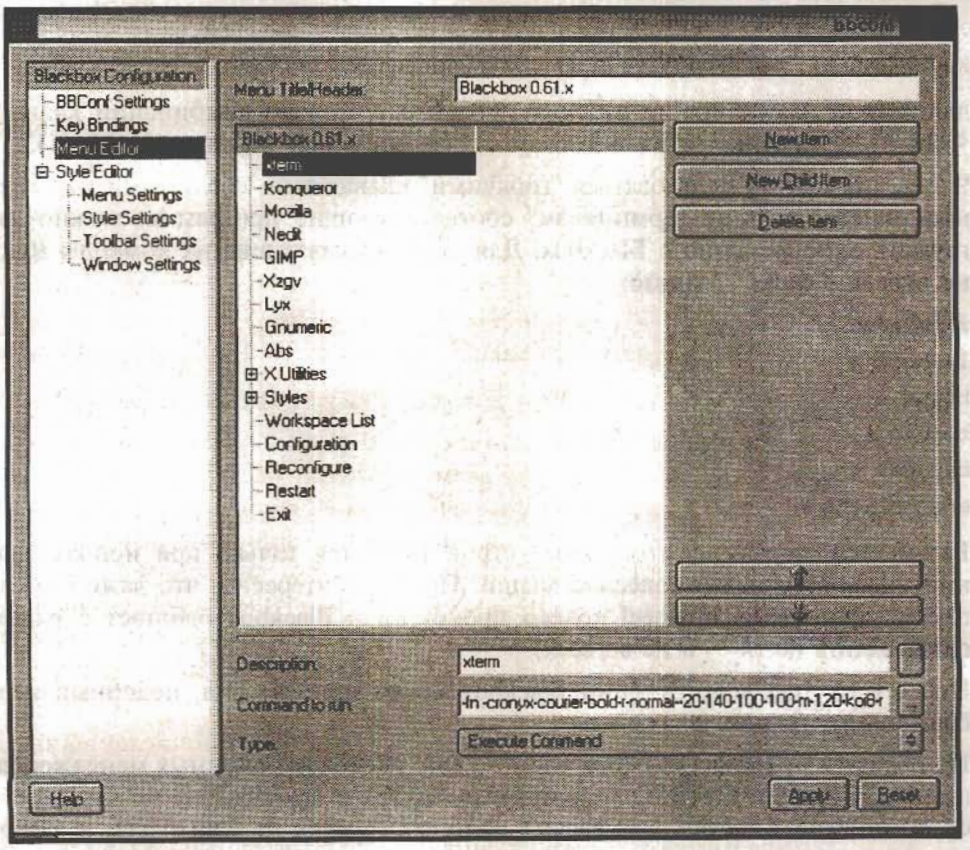


Рис. 20.10. Интерактивный конфигуратор менеджера окон Blackbox

Не обошлось и без интерактивного средства конфигурирования Blackbox — программы bbconf, позволяющей в интерактивном режиме изменять меню этого оконного менеджера, переопределять "горячие" клавиши и т. п. (рис. 20.10). Аналогичное назначение имеет и пакет toolbox — комплексное средство для настройки Blackbox. Впрочем, мне все равно кажется, что настройка вручную более удобна и трудностей не доставляет...

Определение "горячих" клавиш не показалось мне очень уж удобным. Оно требует сначала запуска программы bbkeysConfigC, в которой (в терминальном режиме) нужно определить все клавишные комбинации, которые предполагается задействовать в качестве "горячих", а уж потом посредством bbconf приписать им некоторые действия (перемещения по рабочим столам, например). Однако в результате этого создается новый конфигурационный файл — ~/.bbkeysrc, который легко отредактировать вручную, например, таким образом:

```
KeyToGrab(1), WithModifier(Mod1), WithAction(Workspace1)
...
KeyToGrab(#), WithModifier(Mod1), WithAction(Workspace#)
```

для легкой навигации по рабочим столам с помощью комбинации клавиш <Alt>+<#> (именно <Alt> в данном случае выступает под именем Mod1).

Чтобы постоянно пользоваться "горячими" клавишами, средствами управления окнами и мини-терминалом, соответствующие программы можно загружать одновременно с Blackbox. Для этого вызывающие их команды нужно внести в файл ~/.xinitrc:

```
#!/bin/sh
imwheel &
bbrun &
bbkeys &
bbpager &
exec blackbox
```

Разумеется, первая из командных строк требуется только при использовании скроллирующего колесика мыши. Причем интересно, что даже без загрузки программы imwheel колесо прокрутки в Blackbox работает с рядом приложений (хотя — и не во всех).

После всех этих манипуляций Blackbox может принять вид, подобный изображенному на рис. 20.11.

По образу и подобию Blackbox сделано еще несколько оконных менеджеров, портированных в FreeBSD — например, fluxbox и hakedbox. И по интерфейсу, и по возможностям они весьма мало отличаются от оригинала, и задерживаться на их описании я не буду.

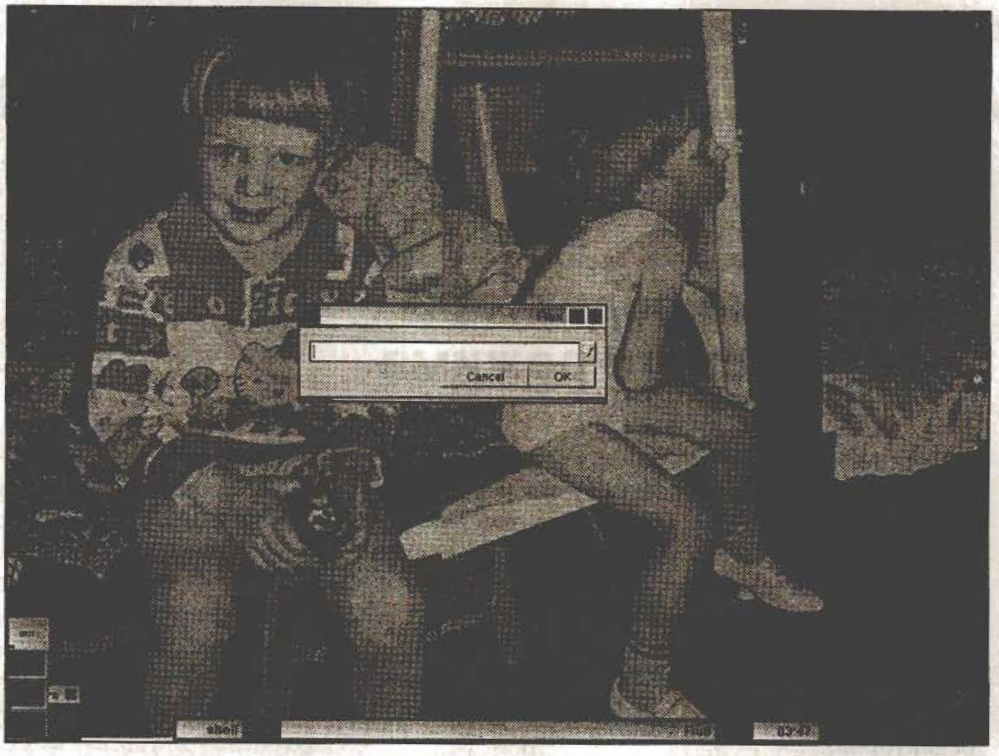


Рис. 20.11. Blackbox с фоновым изображением на рабочем столе и загруженными дополнительными модулями

Дополнительные источники

- Наиболее обстоятельное описание системы X Window на русском языке дано в серии статей Андрея Зубинского под общим названием X Window — восполняя пробелы: часть 1 — **http://www.i2r.ru/static/278/out_9441.shtml**, часть 2 — **http://www.i2r.ru/static/278/out_9569.shtml**, часть 3 — **http://www.itc.ua/article.phtml?ID=9078**.
- Обзор некоторых распространенных менеджеров окон дан в неоднократно упоминавшейся книге Алексея Федорчука. Описания отдельных оконных менеджеров и интегрированных сред можно найти в подборках документации на сайтах **http://linux.perm.ru** и **http://linux.yaroslavl.ru**. То, что посвящены эти сайты преимущественно ОС Linux, не должно смущать — и система X Window, и менеджеры окон для нее никаких отличий в FreeBSD не имеют.

Глава 21

Текстовые редакторы графического режима

Значение текстов и программ для их обработки в UNIX-системах сказывается и на приложениях графического режима. Именно для системы X Window реализовано огромное количество текстовых редакторов, самых разных по функциональности и интерфейсу. И практически все они доступны в FreeBSD в качестве портов, а многие — и как бинарные пакеты. В первую очередь здесь следует отметить штатные средства интегрированных сред KDE и GNOME — каждая из них сочла своим долгом обзавестись несколькими текстовыми редакторами общего назначения (не считая специализированных, например, редакторов шестнадцатеричных кодов или html-редакторов).

В среде KDE к этому классу относятся такие программы, как Kedit — очень примитивный редактор уровня Notepad из Windows, Kwrite — его более развитый собрат, и Kate — очень удобное, хотя и функционально ограниченное (по сравнению, впрочем, только с такими монстрами консольного режима, как Vim и emacs) средство создания и редактирования текстов. Имеется в составе KDE и специальный редактор html-кода — Quanta.

В среде GNOME число текстовых редакторов подбирается к полудюжине. Здесь мы находим и Dedit, и Gedit, и Gnotepad, и Glimmer, и Scintilla, не говоря о более специализированных средствах типа Bluefish (весьма развитого редактора html-кода).

Существует также немало текстовых редакторов, не привязанных к определенной интегрированной среде или набору специальных библиотек. Среди них на первое место следует поставить fte — мощный полнофункциональный редактор, ориентированный в первую очередь на программистов. Сюда же примыкает и Xemacs — воспроизведение функциональности emacs для графического режима (его не следует путать с emacs, запущенным из сеанса X Window — это разные программы).

Описывать все это многообразие в подробностях — не ни сил, ни возможностей, вкратце же — не имеет смысла. И потому я остановлюсь только на

одном редакторе для системы X Window — программе NEdit, единственной, которая по своей функциональности и настраиваемости может состязаться с самыми мощными редакторами консольного режима. И если титул лучшего редактора для консоли — предмет длительных, чуть ни религиозных дискуссий, то в графическом режиме пальма первенства, по моему глубокому убеждению, — несомненно за NEdit.

NEdit — общая характеристика

Редактор NEdit реализован для всех UNIX и UNIX-подобных систем, как коммерческих, так и свободных. В FreeBSD он доступен в качестве порта, а для большинства актуальных версий его можно отыскать в коллекции пакетов. Официальный сайт проекта — **http://www.nedit.org**. В числе разработчиков — большой коллектив, включающий Марка Эдела (Mark Edel), Джо Кирьякопулоса (Joy Kyriakopulos), Кристофера Конрада (Christopher Conrad) и многих других.

NEdit базируется на библиотеке Motif, имеющей статус коммерческой. Однако для FreeBSD он собирается с использованием свободного аналога этой библиотеки — OpenMotif. Наличие последнего — непременное условие успешной установки, вне зависимости, из портов или из пакетов.

По назначению NEdit — текстовый редактор для программистов. Однако реализация его такова, что, кроме этого, он может быть использован для множества целей — от элементарного набора текстов до разработки Web-страниц и верстки оригинал-макетов в TeX.

В чем величие и мощь этого редактора? Во-первых, он позволяет осуществлять все действия по вводу и редактированию текстов, в том числе весьма сложные операции по поиску и замене фрагментов введенного текста.

Во-вторых, NEdit — редактор, в котором настройке поддается практически все от гарнитуры экранного шрифта, его цвета и цвета фона до контекстного меню, вызываемого, как обычно, правой кнопкой мыши.

В-третьих, для NEdit характерна практически неограниченная наращиваемость, осуществляемая путем подключения сценариев командной оболочки и макрокоманд, создаваемых на встроенном языке макросов.

Как и большинство приложений, основанных на библиотеке Motif, NEdit управляется исключительно через меню: инструментальная панель отсутствует, что, однако, компенсируется достаточным количеством комбинаций "горячих" клавиш и, главное, возможностью создавать собственные макрокоманды.

Итак, основными элементами интерфейса редактора NEdit являются строка заголовка, главное меню, строка состояния и строка поиска (Statistic Line и

Incremental Search Line, соответственно, по умолчанию отключены) и собственно рабочее поле (рис. 21.1).

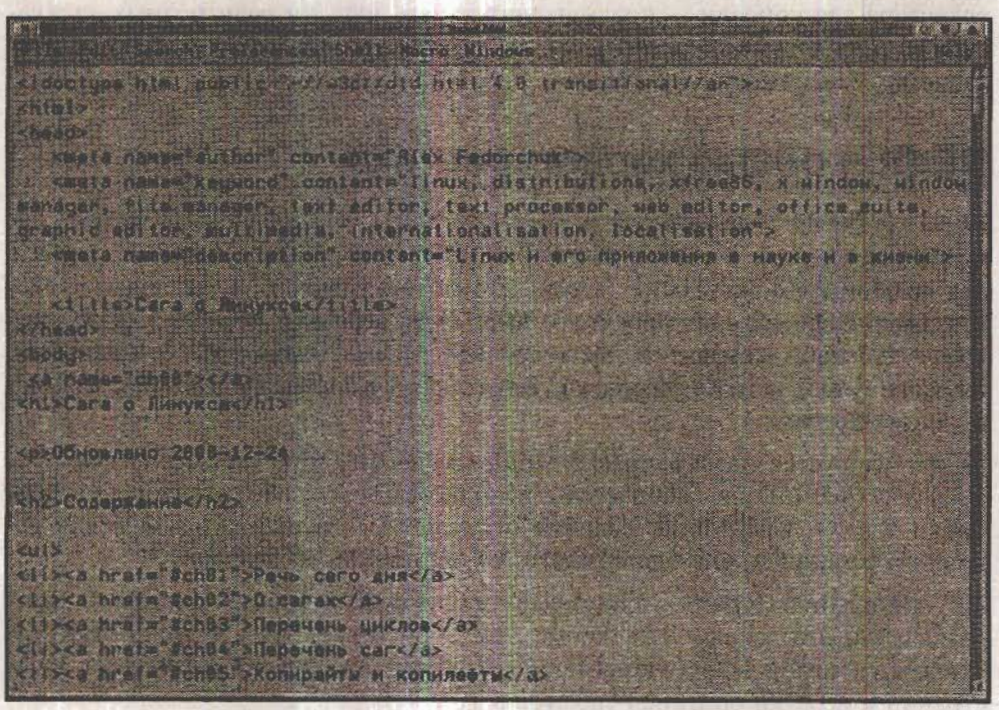


Рис. 21.1. Текстовый редактор NEdit — внешний вид

В строке заголовка, помимо управляющих элементов окна (определяющихся используемым оконным менеджером), указывается имя текущего файла, а также фиксируется, подвергался ли он изменениям со времени последней записи; в этом случае после имени появляется метка (modified).

Основные возможности редактора NEdit становятся ясными, если рассмотреть его главное меню. Оно насчитывает, за исключением непременного **Help**, семь пунктов, для быстрого вызова которых предусмотрена комбинация клавиш <Ctrl>+<подчеркнутая_литера>:

- **File** (Файл) (<Ctrl>+<F>);
- **Edit** (Правка) (<Ctrl>+<E>);
- **Search** (Поиск) (<Ctrl>+<S>);
- **Preferences** (Настройки) (<Ctrl>+<P>);
- **Shell** (Оболочка) (<Ctrl>+<L>);

- **Macro** (Макросы) (<Ctrl>+<C>);
- **Windows** (Окна) (<Ctrl>+<W>).

Большинство действий, соответствующих подпунктам главного меню, также могут быть выполнены с помощью клавишных комбинаций (обычно <Ctrl>, реже <Alt> в сочетании с литерой и, иногда, клавишей <Shift>); что приятно, комбинации эти работают не только при латинской, но и при кириллической раскладке клавиатуры (при правильно установленной локали, конечно, — без этого даже ввод русских символов в NEdit просто невозможен).

Рассмотрим содержание главного меню подробнее.

File

Он включает в себя в основном обычные файловые операции, как то (рис. 21.2):

- создание файла (**New**, <Ctrl>+<N>), сопровождаемое открытием нового окна, относящегося, тем не менее, к текущему сеансу NEdit;
- открытие файла (**Open**, <Ctrl>+<O>), в том числе из списка последних (**Open Previous**), количество которых настраиваемо (см. далее); чрезвычайно полезная опция — открытие файла путем выделения его имени (при необходимости — и полного пути) в редактируемом тексте (**Open Selected**, <Ctrl>+<Y>); как и при создании файла, каждый документ открывается в своем собственном окне;
- закрытие файла (**Close**, <Ctrl>+<W>); в случае его модификации по умолчанию следует запрос на запись изменений, а также установок по умолчанию, если они изменялись;
- сохранение файла (**Save**, <Ctrl>+<S>), в том числе и под другим именем (**Save As**); последняя опция позволяет сохранить текстовый файл не только в формате UNIX (с кодом LF в качестве символа возврата каретки), но и формате DOS (он же Windows), где в этом качестве применяется комбинация кодов CR+LF; возможно также дополнение строк, оборванных при включении переноса слов (о чем будет разговор в разделе о настройках), до полного абзаца, завершаемого нажатием клавиши <Enter> (**Add line breaks where wrapped**);
- уникальная опция отмены изменений в уже записанном файле (**Revert to Saved**);
- вставка некоего существующего файла в текущий документ (**Include File**, <Alt>+<I>), осуществляемая в позиции курсора;
- открытие файла макросов или тэгов (**Load Macro File** и **Load Tags File** соответственно), о чем подробнее — ниже;
- печать текущего файла (**Print**, <Ctrl>+<P>) или выделенного фрагмента (**Print Selection**);
- выход из программы (**Exit**, <Ctrl>+<Q>).

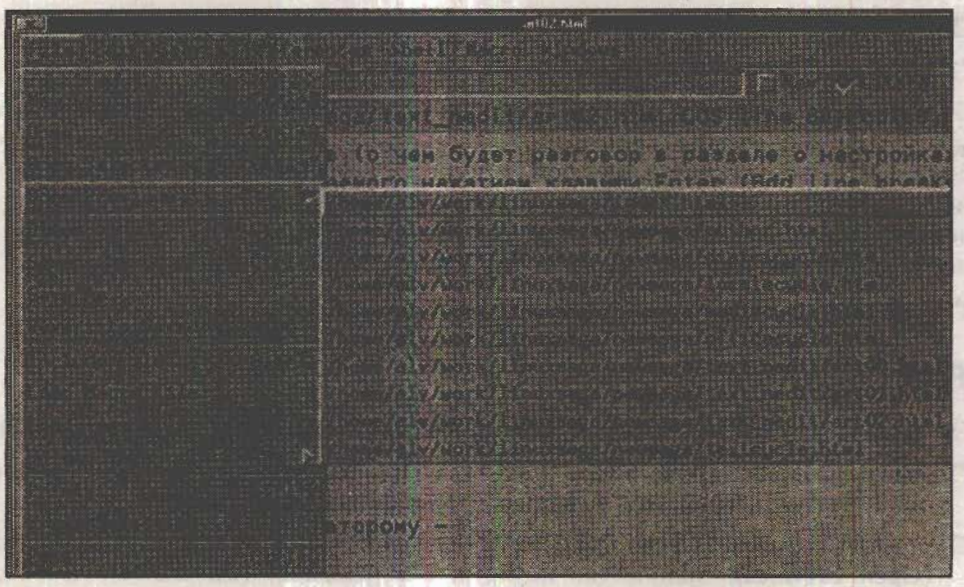


Рис. 21.2. Главное меню, пункт **File**

Edit

Содержание пункта **Edit** (Правка) также достаточно тривиально, это (рис. 21.3):

- отмена нескольких последних операций и возврат отмененных действий (**Undo**, <Ctrl>+<Z>, и **Redo**, <Shift>+<Ctrl>+<Z> соответственно);
- вырезание (**Cut**), копирование (**Copy**) и вставка (**Paste**) выделенного фрагмента; закрепленные за этими операциями комбинации клавиш — привычны пользователям Windows: <Ctrl>+<X>, <Ctrl>+<C> и <Ctrl>+ +<V> соответственно;
- вставка выделенного фрагмента в виде колонки (**Paste Column**, <Ctrl>+<Shift>+<V>); вставляемый фрагмент в этом случае как бы вклинивается в существующий текст;
- удаление выделенного фрагмента (**Delete**, клавиша <Del>);
- выделение всего документа (**Select All**, <Ctrl>+<A>);
- выделение текста слева (**Shift Left**, <Shift>+<Ctrl>+<9>) или справа (**Shift Right**, <Shift>+<Ctrl>+<0>) от позиции курсора;
- конвертация букв выделенного фрагмента из верхнего регистра в нижний, и наоборот (**Lower-case**, <Shift>+<Ctrl>+<6>, и **Upper-case**, <Ctrl>+ +<6> соответственно);

- подпункт **Fill Paragraph** (Заполнение абзаца) (<Ctrl>+<J>) конденсирует абзацы в соответствии с принятыми в **Preferences** (Настройки) правилами переноса слов (о чем подробнее расскажу в разделе о настройках редактора);
- пункты **Insert Form Feed** (Вставка форм) (<Alt>+<Ctrl>+<L>) и **Insert Control Code** (Вставка управляющих символов) (<Alt>+<Ctrl>+<I>) позволяют вставлять всякого рода управляющие символы; в частности, во втором случае вызывается панель с предложением ввести ASCII-код желаемого символа в десятичном счислении.

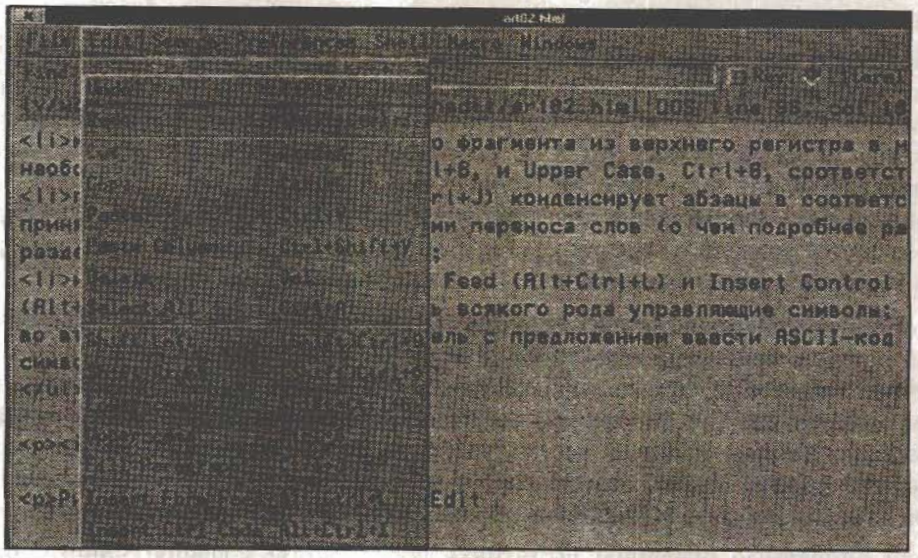


Рис. 21.3. Главное меню, пункт **Edit**

Следует заметить, что вставка выделенного фрагмента может осуществляться не только через меню, но и стандартным для Linux способом — щелчком средней кнопки мыши. При этом важно не фиксировать курсор в позиции вставки — это автоматически приводит к снятию выделения и очищению буфера. В этом проявляется отличие NEdit от, скажем, редакторов для KDE, где выделенный фрагмент остается в буфере и после снятия выделения, вплоть до выделения нового фрагмента, и может быть вставлен повторно в новой позиции.

Однако и в NEdit с помощью мыши можно многократно вставлять выделенный фрагмент, т. к. после первой вставки щелчком средней кнопки выделение не снимается, и процедуру можно повторять (или — удалить выделенный фрагмент, нажав клавишу <Del>, не меняя положения курсора) до

фиксации курсора в новой позиции; таковая происходит после нажатия любой из клавиш управления курсором, но не при перемещении по телу документа с помощью линейки прокрутки.

Вообще говоря, в терминологии NEdit различаются два типа выделения: первичное (primary, highlighted text) и вторичное (secondary, underlined text), действия над которыми различны.

Первичное выделение осуществляется протаскиванием курсора мыши при нажатой ее левой кнопке или стрелками указателя курсора при нажатой клавише <Shift>, как и в Windows. Выделенный таким образом фрагмент может быть скопирован, удален и вставлен через меню или соответствующие клавишные комбинации, а также вставлен щелчком средней кнопки мыши.

Первичное выделение возможно не только для строк, но и для прямоугольных фрагментов. Оно осуществляется мышью обычным способом, но при нажатой клавише <Ctrl>. С выделенным прямоугольным фрагментом возможны те же действия, что и со строчным. Единственно, прямоугольный фрагмент может быть выделен только при использовании какой-либо моноширинной гарнитуры в качестве экранного шрифта (а NEdit, как будет показано в разделе о настройках, в отличие от большинства текстовых редакторов, допускает использование и пропорциональных гарнитур).

Вторичное выделение осуществляется только мышью. Оно служит для быстрой вставки фрагмента в текущую позицию курсора. Чтобы произвести вторичное выделение, нужно зафиксировать курсор в позиции для вставки, затем навести его (не фиксируя!) на начало выделяемого фрагмента, нажать среднюю кнопку мыши и вести курсор до требуемого места. После этого средняя кнопка мыши отпускается — и выделенный фрагмент (выделение при этом пропадает) волшебным образом копируется в новую позицию.

Если описанную операцию выполнить при нажатой клавише <Shift>, произойдет перемещение выделенного фрагмента. То есть — точно так же, как и при перетаскивании мышью в Windows (drag-and-drop). Прием, насколько мне известно, не имеющий аналогов в других приложениях для X Window, далеко не все из которых не поддерживают и обычный drag-and-drop как таковой.

Search

Опции поиска в NEdit вполне заслуживают отдельного положения в главном меню, поскольку возможности поиска и замены в NEdit чрезвычайно широки (рис. 21.4). Здесь и поиск введенной последовательности символов (<Shift>+<Ctrl>+<F>), и повторный поиск (<Shift>+<Ctrl>+<G>), и поиск фрагмента, идентичного выделенному (<Shift>+<Ctrl>+<H>). Возможен поиск как буквенных последовательностей (**Literal**), в том числе и с чувствительностью к регистру (**Case Sensitive Literal**), так и регулярных выражений (**Regular Expression**), поиск вперед и назад (**Search Forward** и **Search**

Backward соответственно). При этом поддерживается история поиска: клавиша управления курсором <↑> вызывает появление предыдущей искомой последовательности, клавиша <↓> — последующей.

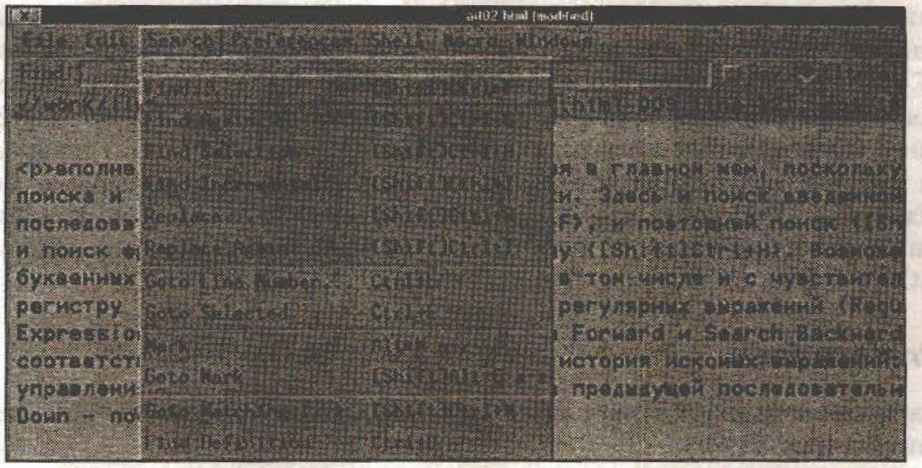


Рис. 21.4. Главное меню, пункт **Search**

Следует отметить, что строка для ввода искомой последовательности может быть выведена не только в диалоговом окне по вызову соответствующего пункта меню (рис. 21.5), но и включена в верхней части окна программы, между строкой меню и строкой состояния (с помощью команды **Find Incremental** (Наращиваемый поиск), <Shift>+<Ctrl>+<I>). В обоих случаях кириллица в строке поиска не отображается, заменяясь знакомой пользователям Windows псевдо-грекофранцузской абракадаброй; тем не менее, самому по себе поиску это отнюдь не мешает — кириллические символы в теле документа находятся исправно.

Рис. 21.5. Диалоговое окно для ввода выражений поиска/замены

Разумеется, возможна и замена найденных фрагментов (**Replace**, <Shift>+ +<Ctrl>+<R>), и повторная замена (**Replace Again**, <Shift>+<Ctrl>+<T>). Опции замены — те же, что и для поиска: с чувствительностью к регистру, с использованием регулярных выражений, вперед и назад и т. д.

В пункте же **Search** (Поиск) — переход к строке по ее номеру (**Goto Line Number**, <Ctrl>+<L>) и к выделенному фрагменту (**Goto Selected**, <Ctrl>+<E>), установка закладок (**Mark**, <Alt>+<M>), в качестве которых могут использоваться буквы латинского алфавита (от A до Z), и переход к ранее установленным закладкам (**Goto Mark**,<Shift>+<Alt>+<G>) и еще пара пунктов, содержания которых я не очень понял. Это, во-первых, **Goto Matching (...)** (Перейти к выражению) (<Shift>+<Ctrl>+<M>), не вызывающий никаких действий, и, во-вторых, **Find Definition** (Поиск определений) (<Ctrl>+<D>), не активизированный вообще.

Preferences

Содержание данного пункта меню вполне понятно — это настройки, среди которых (рис. 21.6):

- включение/выключение строки состояния (**Statistics Line**, <Alt>+<A>) и строки поиска (**Incremental Search Line**);
- выведение/скрытие нумерации строк (**Show Line Numbers**);
- выбор языкового режима (**Launguage Mode**, в смысле — языка программирования, а не человеческого);
- включение/выключение автоматических отступов в тексте (**Auto Indent**);
- настройка режима переноса строк (**Wrap**);
- установка величины табуляции (**Tabs**);
- выбор экранного шрифта (**Text Font**);
- включение/выключение подсветки синтаксиса (**Highlight Suntax**);
- разрешение (или запрещение) создания резервных копий редактируемого документа при записи (**Make Backup Copy, *.bck**);
- включение/выключение специальных возможностей — **Incremental Backup** (Наращиваемое резервирование) и **Show Matching (...)** (Показать выражение);
- включение/выключения забоя при наборе (**Overtype**, <Ctrl>+<B>);
- установка запрета изменения (**Read Only**) текущего документа.

Уже из этого краткого перечня можно представить себе возможности настройки редактора NEdit, о чем я подробнее буду говорить в следующем разделе. Пока же отмечу только, что все эти настройки имеют силу только в

текущем сеансе. Чтобы сделать их перманентными, требуется внести соответствующие изменения в пунктах **Default Settings** (о чем — ниже), и сохранить их (**Save Defaults**).

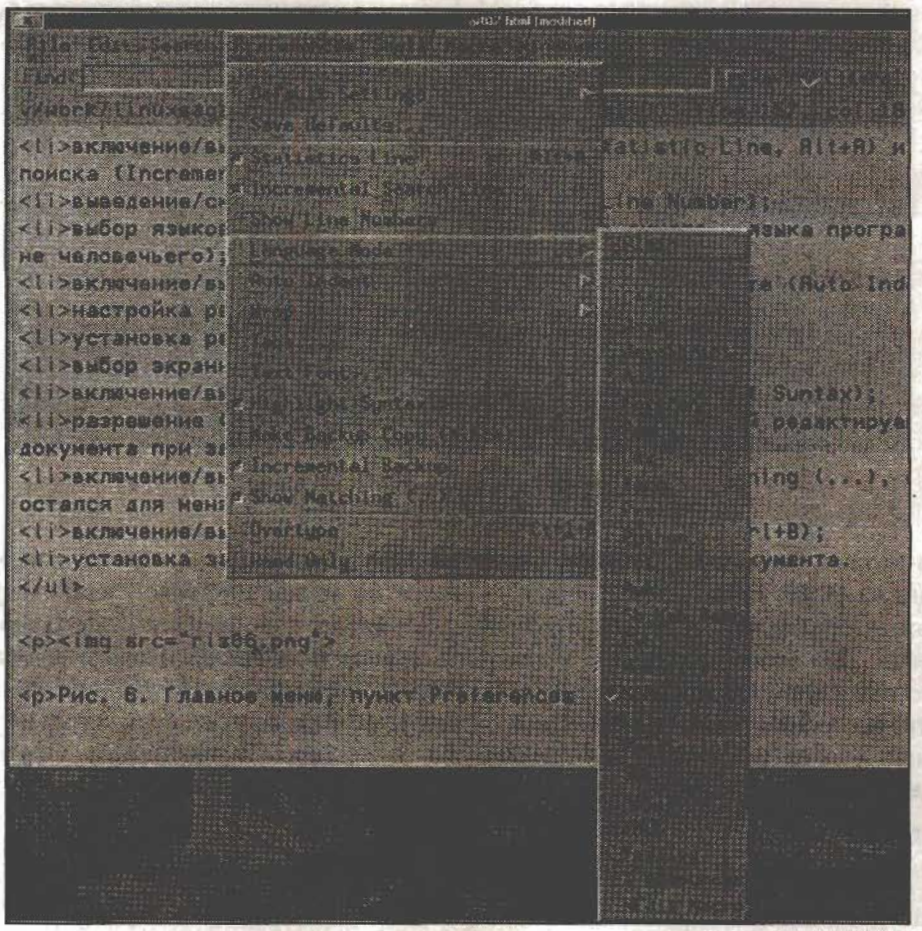


Рис. 21.6. Главное меню, пункт **Preferences**

Shell

Shell — достаточно необычный пункт меню текстового редактора. С его помощью можно, во-первых, запустить на исполнение любую команду или программу. Это делается с помощью подпункта **Execute Command** (Выполнение команд) (<Alt>+<X>), который вызывает нечто вроде мини-терминала. В нем может быть введена соответствующая команда оболочки

(или любая другая) с любыми требующимися параметрами. Поддерживается посредством клавиш управления курсором <↑> и <↓>, история команд, правда, только отданных в текущем сеансе (рис. 21.7).

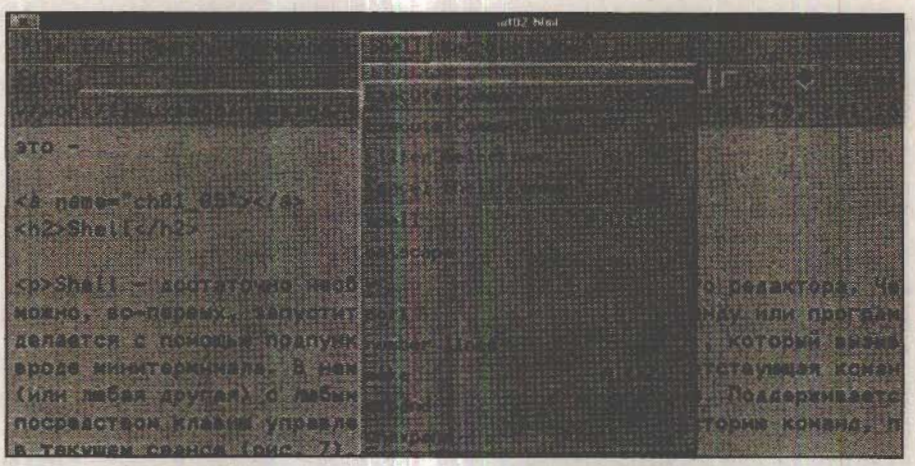


Рис. 21.7. Главное меню, пункт **Shell**

Во-вторых, командой **Execute Command Line** (Выполнение строки как команды) (<KP Enter>, т. е. "серый" <Enter> на малой цифровой клавиатуре) можно определить в качестве команды на запуск строку из текущего документа. Для этого достаточно установить курсор в любое ее место и выбрать соответствующий пункт меню (или набрать вышеприведенную неудобопонятную клавишную комбинацию, вероятно). Разумеется, если строка эта имеет смысл и записана синтаксически правильно. В противном случае в теле документа автоматически появится сообщение об ошибке. Например:

```
netscape
An error occurred running /usr/lib/netscape/netscape-communicator.
```

Или даже полная справка по использованию команды, как в случае

```
ispell
Usage: ispell [-dfile | -pfile | -wchars | -Wn | -t | -n | -x | -b | -S |
-B | -C | -P | -m | -Lcontext | -M | -N | -Ttype | -V] file.....
```

и т. д. Разумеется, эта опция наиболее полезна для программистов. Однако и простым смертным она будет весьма нелишней при отладке сценариев командной оболочки. Необходимость в чем может возникнуть при желании нарастить функциональные возможности NEdit.

Дело в том, что такие сценарии могут быть встроены в меню **Shell**. Некий их набор включен в него по умолчанию. Это проверка правописания (**spell**),

статистика (**wc**, от word count — подсчет строк, слов и знаков), сортировка строк и их нумерация, а также сугубо программистские сценарии — например, **make** для сборки программы из набранного исходного текста посредством запуска одноименной программы управления компиляцией. Однако ничто не мешает пополнять меню **Shell** своими сценариями любого назначения, освободив его от ненужных лично вам элементов. Как — расскажу в одном из следующих разделов.

Macro

Этот пункт разделяется на две секции (рис. 21.8). Первая, стабильная, предназначена для создания и запуска пользовательских макрокоманд. В ней можно видеть:

- включение режима протоколирования макросов (**Learn Keystrokes**, <Alt>+<K>);
- нормальное завершение (**Finish Learn**, повторное <Alt>+<K>) и прерывание (**Cansel Learn**, <Ctrl>+<.>) протоколирования;
- воспроизведение (**Replay Keystrokes**, <Ctrl>+<K>) и повторение (**Repeat**, <Ctrl>+<.>) запротоколированных макрокоманд.

Вторая секция — настраиваемая, в нее можно включить макрокоманды, как идущие в комплекте, так и созданные собственноручно. Например, с помощью протоколирования — иначе они не сохранятся по завершении текущего сеанса.

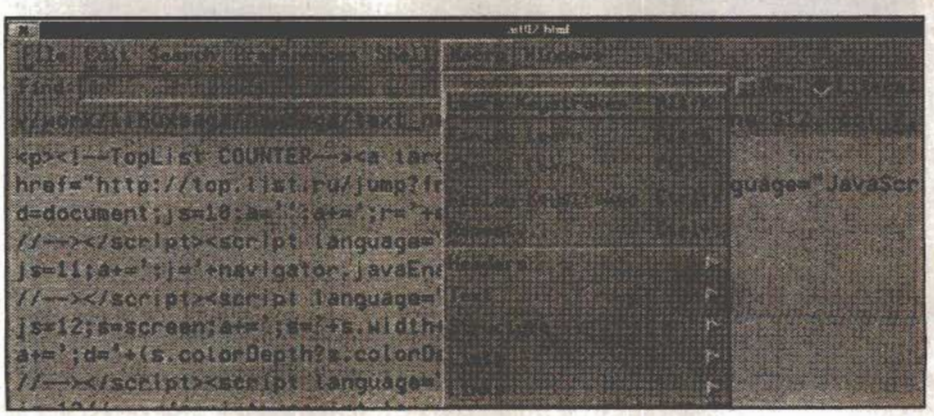


Рис. 21.8. Главное меню, пункт **Macro**

В комплекте с NEdit идет несколько макросов — дополнение частично напечатанных слов путем сопоставления их со словарем (**Complete Word**,

<Alt>+<D>) или ответы на письма — с цитированием (**Quote Mail Reply**) и без оного (**Unquote Mail Reply**).

Кроме того, имеется серия языково-зависимых макросов, предназначенных для программирования на C++. Они появляются в меню только при включении соответствующего языкового режима. Однако, как и в случае с **Shell**, не видно причин, по которым нельзя было бы исключить из меню заведомо ненужные макросы, заменив их собственными. Именно это и будет темой заключительных разделов главы.

Windows

Понимается под этим, как не трудно догадаться, управление окнами, правда, довольно ограниченное. Во-первых, можно разделить окно с текущим документом на две независимо скроллируемые части (**Split Window**, <Ctrl>+<2>). Только по горизонтали, но зато сколько угодно раз: исходное окно делится ровно пополам, затем каждая половина — еще пополам, и т. д., насколько хватит разрешения экрана (рис. 21.9).

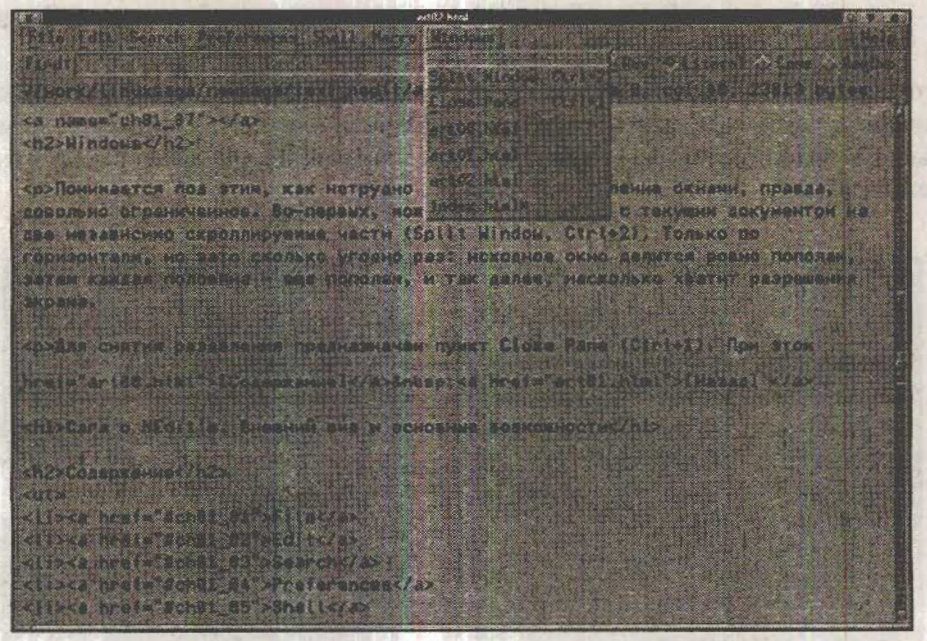


Рис. 21.9. Главное меню, пункт **Windows**

Для снятия разделения предназначен пункт **Close Pane** (Снять разделение) (<Ctrl>+<1>). При этом закрывается та часть окна, в которой находится курсор.

Наконец, в пункте **Windows** (Окна) содержится список открытых в текущем сеансе документов. Каждый из них открывается или создается (командами меню **File — Open** (Открыть) или **File — New** (Создать) соответственно) в своем окне с полным набором его атрибутов (управляющими элементами, меню, строкой состояния и т. д.). Однако все они принадлежат текущему сеансу и, скажем, изменение настроек сказывается на них всех. А переключение между окнами, кроме принятого в данном оконном менеджере, можно осуществить и через их список в пункте меню **Windows** (Окна).

Help

Встроенная помощь редактора NEdit весьма богата, что можно видеть из простого перечисления содержания этого пункта меню (рис. 21.10). Он включает:

- **Getting Started** (Быстрый старт) — краткое описание основных особенностей программы;
- **Basic Operations**, где описаны приемы выделения текста (**Selecting Text**), поиска и замены фрагментов (**Finding and Replacing Text**), их вырезания и вставки (**Cut** и **Paste**), использование мыши (**Using the Mouse**), комбинации горячих клавиш (**Keyboard Shortcuts**), а также формат файлов NEdit (**File Format**);
- **Features for Programming** (Особенности программирования), т. е. возможности для програмистов; не будучи таковым, задерживаться здесь не буду;
- **Regular Expressions** (Регулярные выражения), на что также распространяется вышесказанное; здесь весьма подробно описаны синтаксис регулярных выражений и прочие их особенности, а также приведены примеры; с этим надеюсь разобраться в будущем;
- **Macro / Shell Extensions** (Расширения макросами и сценариями), напротив, заслуживает всяческого внимания; именно здесь подробно описаны: использование скриптов оболочки командной строки, процесс протоколирования макрокоманд, синтаксис встроенного макроязыка NEdit и пр.;
- **Customizing** (Индивидуализация) — не менее интересная позиция; в ней приведены сведения о всех видах настройки NEdit; что составит предмет следующего раздела;
- **NEdit Command Line** (Опции командной строки) — описание параметров для запуска NEdit; большая часть достигаемых таким путем опций может быть реализована также и через настройки;
- **Server Mode and nc** (Серверный режим) — описание использования NEdit в режиме "клиент-сервер";

- **Crash Recovery** (Восстановление при сбоях) — описание действий в экстремальных ситуациях.

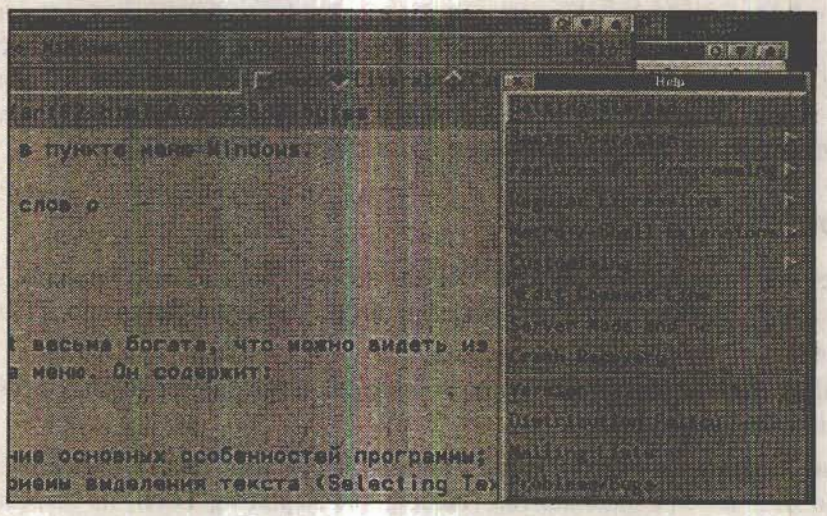


Рис. 21.10. Главное меню, пункт **Help**

Кроме перечисленного, в пункте **Help** (Помощь) содержатся подробные сведения о текущей версии редактора, политике его распространения, приведены списки рассылки и небольшой перечень проблем и ошибок.

Пора, однако, заканчивать с рассмотрением главного меню. Поскольку в NEdit существует еще и меню контекстное (в терминологии NEdit — **Window Background Menu**), вызываемое, как положено, щелчком правой кнопкой мыши на выделенном фрагменте текста. Правда, оно достаточно простое, содержит лишь пункты **Undo** (Отмена) и **Redo** (Возврат отмененного), **Cut** (Вырезать), **Copy** (Копировать) и **Paste** (Вставить), что в комментариях не нуждается. Однако, как будет показано ниже, и оно может быть наращено.

Таким образом, из рассмотрения меню функциональность NEdit в первом приближении становится ясна. Он обладает очень развитыми средствами ввода и редактирования текстов как общего содержания, так и исходных текстов на языках программирования и разметки. Меню его организовано логичным и привычным для пользователя Windows способом, комбинации "горячих" клавиш многочисленны и обычно имеют простой мнемонический смысл. Все это, вместе с мощной системой помощи, делает NEdit простым в освоении.

К недостаткам NEdit (или, вернее, особенностям, которые могут не нравится), можно отнести отсутствие инструментальной панели с кнопками и открытие каждого файла в собственном окне (а не в том же самом, с перемещени-

ем через систему вкладок, например). Впрочем, и тот, и другой момент — спорны: отсутствие кнопочной панели, как я уже говорил, компенсируется системой "горячих" клавиш, а перемещение между отдельными окнами, хотя и приводит к загромождению дисплея, не менее удобно, чем между вкладками. Единственно, не помешала бы возможность аранжировки открытых окон, но этого можно обычно добиться средствами большинства оконных менеджеров.

Еще в NEdit нет штатных средств управления проектом. Хотя бы в элементарном исполнении, т. е. в виде открытия и сохранения группы связанных файлов (глав книги, например, или серии Web-страниц). Тем не менее, все это не перечеркивает высоких достоинств редактора.

Однако мало было бы проку от богатства возможностей редактирования, если бы они не сопровождались адекватным богатством настроек. Что же, посмотрим, каковы они, возможности настройки.

Настройка NEdit

Я уже говорил, что бóльшая часть настроек редактора NEdit сконцентрирована в пункте **Preference** (Настройки) главного меню. И даже кратко перечислил возможности настройки текущей сессии. Однако нас больше интересует не сиюминутный результат настройки, но ее долгосрочная перспектива. Для чего следует рассмотреть пункт **Default Settings** (Умолчания) меню **Preference**. Содержание его близко другим пунктам меню **Preference** и включает установки (рис. 21.11):

- языкового режима (**Language Modes**);
- включения/выключения автоматических отступов (**Auto Indent**);
- режима переноса слов (**Wrap**);
- величины табуляции (**Tabs**);
- экранного шрифта (**Text Fonts**);
- настройки меню (**Customize Menus**);
- режима поиска (**Searching**);
- подсветки синтаксиса (**Syntax Highlighting**);
- переключателей вывода строки состояния (**Statistics Line**), строки поиска (**Incremental Search Line**) и нумерации строк (**Show Line Numbers**);
- резервного копирования (**Make Backup Copy (*.bck)**);
- показа регулярных выражений — **Show Matching (. .)**;
- порядка сортировки списка ранее открывавшихся файлов (**Sort Open Prev. Menu**);

- состава контекстного меню (**Popups Under Pointer**);
- выдачи предупреждающих сообщений (**Warnings**);
- начального размера окна при запуске NEdit (**Initial Window Size**).

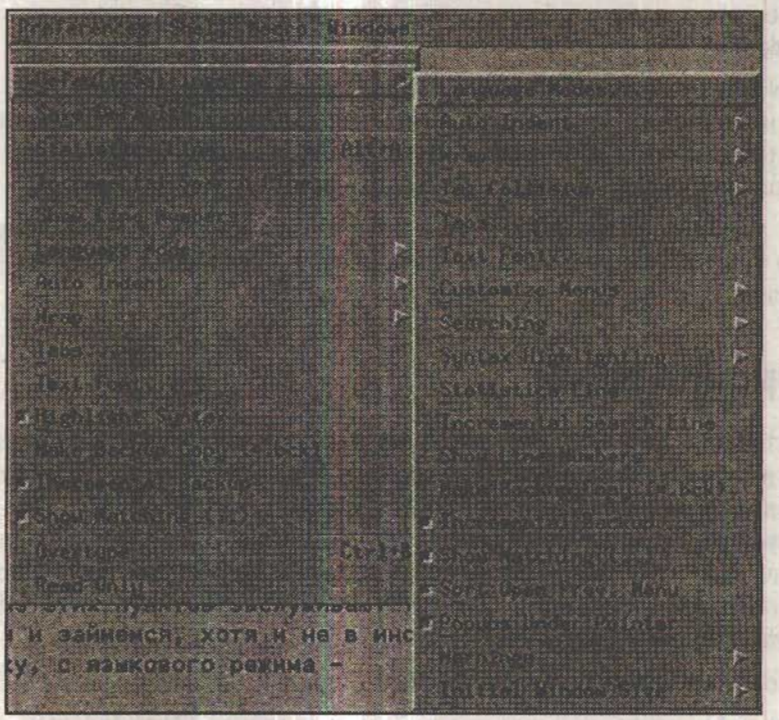


Рис. 21.11. Настройки NEdit, пункт меню **Preferences — Default Swttings**

Большинство из этих пунктов заслуживают того, чтобы задержать на них внимание. Чем мы и займемся, хотя в несколько иной последовательности. Но начнем по порядку, с языкового режима **Language Modes**.

Language Modes

Как уже говорилось, под этим подразумевается язык не человеческий, а программистский. И список поддерживаемых режимов включает, не считая чисто текстового (Plain), почти все известные мне (и многие неизвестные) языки программирования и разметки. Здесь и C и C++, Java и JavaScript, Ada, Fortran, Pascal, Python, Tcl, awk, языки командных оболочек линии sh и линии csh, языки разметки SGML/HTML и LaTeX, PostScript, SQL и про-

чая, и прочая, и прочая. Разумеется, есть и режим собственного языка NEdit Macro (рис. 21.12).

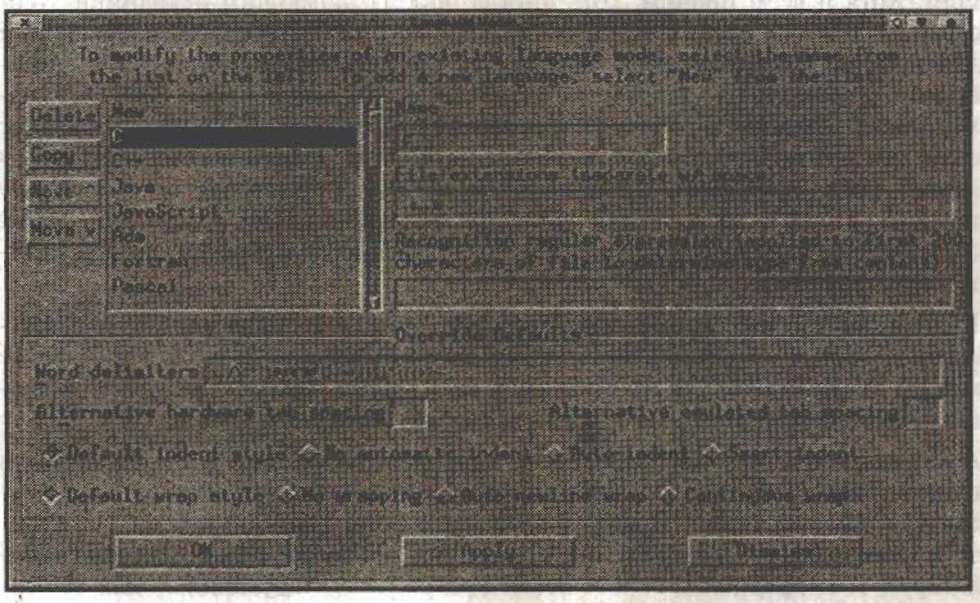


Рис. 21.12. Выбор языкового режима

Кроме языков, поддерживаемых в штатном комплекте, можно ввести поддержку любого иного языка программирования, в том числе и самоизобретенных. Для этого достаточно выбрать в списке языков пункт **New** (Новый), определить для нового языка его имя, закрепленное расширение имени файла и прочие параметры, которые можно видеть на рис. 21.12.

Первое видимое следствие переключения языкового режима — появление подсветки синтаксиса соответствующего языка (если, конечно, эта опция включена, о чем — ниже). Так, по умолчанию в NEdit задействован режим **Plain text** (Простой текст). В этом случае, скажем, тэги в html-документе показываются теми же черными (для примера) буквами), что и его содержание. Если же переключиться в режим **SGML/HTML**, тэги, их аргументы и значения последних заиграют различными цветами. Впрочем, при открытии файла определение его типа обычно происходит автоматически, по расширению (и обычно — правильно), и соответствующий языковый режим включается сам собой.

Однако различие между языковыми режимами не исчерпывается цветовой гаммой. Многие макросы (о которых — в следующем разделе) имеют языковую привязку. И появляются в меню **Macro** (Макросы) только тогда, когда включается соответствующий их языку режим.

Auto-indent, Wrap и Tabs

Разумеется, подсветка синтаксиса необходима в первую очередь программистам. Хотя и всем прочим, хоть иногда просматривающим исходники Web-страниц, тоже не вредит. А вот следующая группа настроек предназначена и для простых смертных, т. к. определяет правила набора текста. Так, **Auto-indent** (Автоотступ) — это просто включение или выключение (**Off/On**) возможности создания автоматических абзацных отступов. Поясню на примере: если вам нужно набрать серию абзацев с общим отступом семь пунктов от левого края, вы задаете (например, клавишей табулятора) отступ только первого абзаца — во всех остальных при включении (**On**) опции **Auto-indent** (Автоотступ) заданный отступ будет появляться автоматически, до его уничтожения вручную.

В этом же пункте, кроме переключения, есть еще одна опция — **Smart-indent** ("Умные" отступы), но она не активизирована. Подозреваю, что ее можно включить только при режимах каких-либо языков, предусматривающих некие фиксированные отступы в строках исходного текста.

А вот величина отступа, задаваемого клавишей табуляции, и определяется в пункте **Tabs** (Табуляция). Здесь нужно просто задать (в знаках) величину отступа при табуляции. Правда с оговоркой — **for hardware space** ("По железу"), поскольку тут же присутствует и опция — **emulate tabs** (Эмуляция табулятора), т. е. предполагается и некая программная табуляция — для терминалов, клавиши <Tab> не имеющих.

Опция **Wrap** (Перенос слов) важна, т. к. она отражается на ряде других параметров набора текста. Для нее возможны варианты (рис. 21.13):

- **None** (Нет), т. е. каждый абзац набирается в одну строку вплоть до нажатия клавиши <Enter>; это вполне приемлемо для исходных текстов программ, но вряд ли очень удобно для набора текстов просто, т. к. строка "убегает за горизонт";
- **Auto Newline** (Автоперенос) — выбор этой позиции приводит к тому, что каждая строка обрывается на границе окна, в результате чего сплошной текст (вроде этого, например) приобретает читабельность; однако при экспорте его в какой-либо текстовый процессор обнаруживается, что каждая строка заканчивается символом возврата каретки (т. е. образует самостоятельный абзац), искоренение которых, например, в MS Word — занятие не из самых веселых;
- **Continuous** (Неразрывная строка) — оптимальный выбор для набора длинных связных текстов; в этом случае при достижении границы окна происходит разрыв экранной (не истинной!) строки без появления лишнего символа возврата каретки и, соответственно, без образования нового абзаца; каковой появляется только по нажатии клавиши <Enter>; это аналогично условиям переноса строк по умолчанию в emacs; в последнем, в отличие от NEdit, разрыв экранной строки маркируется обратным

слэшем (\), что, конечно, наглядно показывает отличие строк экранных и истинных, но, с другой стороны, загромождает текст;

- **Wrap Margin** (Ограничение строки), последняя из опций переноса слов, позволяет явным образом задать длину строки в знаках, после чего происходит ее разрыв символом возврата каретки.

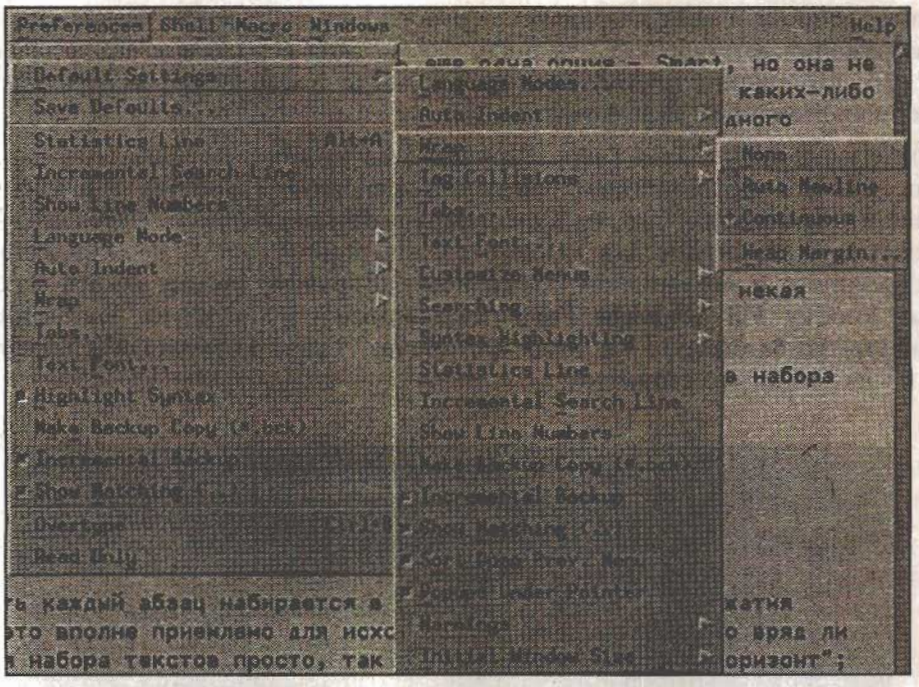


Рис. 21.13. Настройка режима переноса слов

Я столь подробно остановился на параметрах переноса слов, потому что они представляются мне одними из наиболее существенных при наборе текста. Особенно если в дальнейшем предполагается экспорт его в текстовый процессор или какую-либо программу верстки. К тому же именно установками пункта **Warp** (Перенос слов) определяется реакция на команду меню **Edit** — **Fill Paragraph** (Заполнение абзаца) (<Ctrl>+<J>): при установке в позиции **Continuous** (Неразрывная строка) текст в NEdit, независимо от того, в каком режиме переноса слов был набран, будет сконденсирован в единые абзацы, разделами между которыми будут только явные нажатия клавиши <Enter>.

Не уверен, что выразился вполне внятно, поэтому поясню на примере: некий текст был набран в режиме **Auto Newline** (Автоперенос), после чего выяснилось, что его следует экспортировать в MS Word; чтобы избавить себя в дальнейшем от ручного искоренения символов возврата каретки или голов-

ной боли по измышлению автоматизации этого процесса, достаточно просто пройтись по всем абзацам текста, нажимая <Ctrl>+<J> — и при включенной опции **Continuous** (Неразрывная строка) мы получим его аккуратно поделенным именно тем образом, что и требуется.

Text Fonts

Этому вопросу я всегда придавал первостепенное значение. Во-первых, из соображений эстетических (грешным делом, люблю красивые шрифты), во-вторых, по причине плохого зрения. Так вот, с точки зрения выбора экранных шрифтов NEdit заслуживает аплодисментов.

При вызове пункта **Text Fonts** (Шрифты текста) возникает диалоговое окно (рис. 21.14) с предложением определить основную гарнитуру (по умолчанию — нормального, т. е. не курсивного и не полужирного, начертания) и отдельно — гарнитуры для каждого из начертаний (опять же, курсивного, полужирного и полужирного курсивного). Впрочем, можно нажать экранную кнопку **Fill Highlight Fonts from Primary** (Определить выделенные шрифты по базовому) — и тогда все прочие, кроме нормального, начертания унаследуют его гарнитуру.

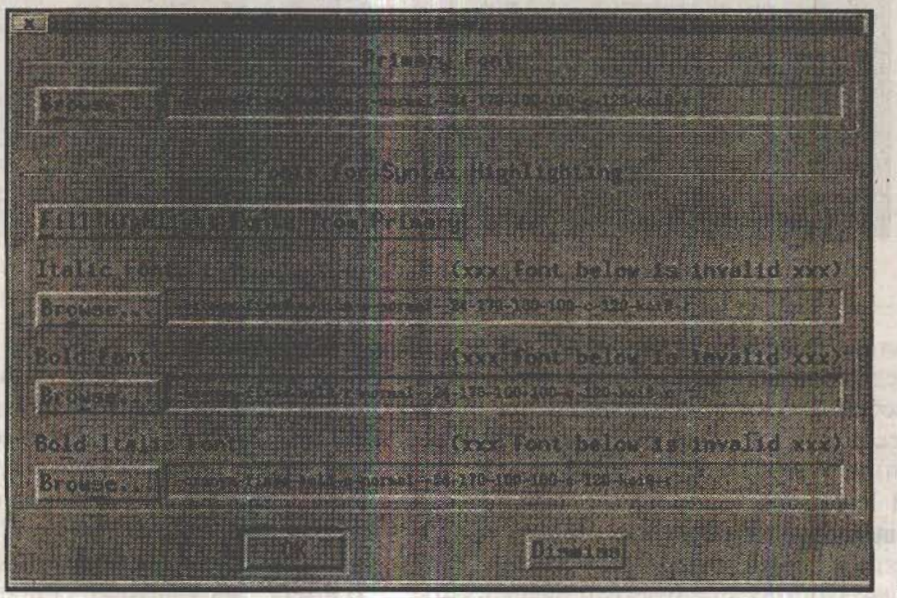


Рис. 21.14. Настройка экранных шрифтов

В строке **Primary Font** (Базовый шрифт) (как, впрочем, и в любой другой) определить гарнитуру (и прочие параметры) экранного шрифта можно вво-

дом вручную, с указанием всех требуемых атрибутов. Однако проще нажать экранную кнопку **Browse** и выбрать шрифт полужирного начертания из списка всех доступных системе шрифтов. Выбор осуществляется в диалоговом окне — в трех его колонках необходимо явным образом отметить все три доступные для выбора атрибута — **Font** (Шрифт) (т. е. гарнитуру), **Style** (Начертание) и **Size** (Размер) (сиречь кегль), иначе последует сообщение о некорректном определении шрифта.

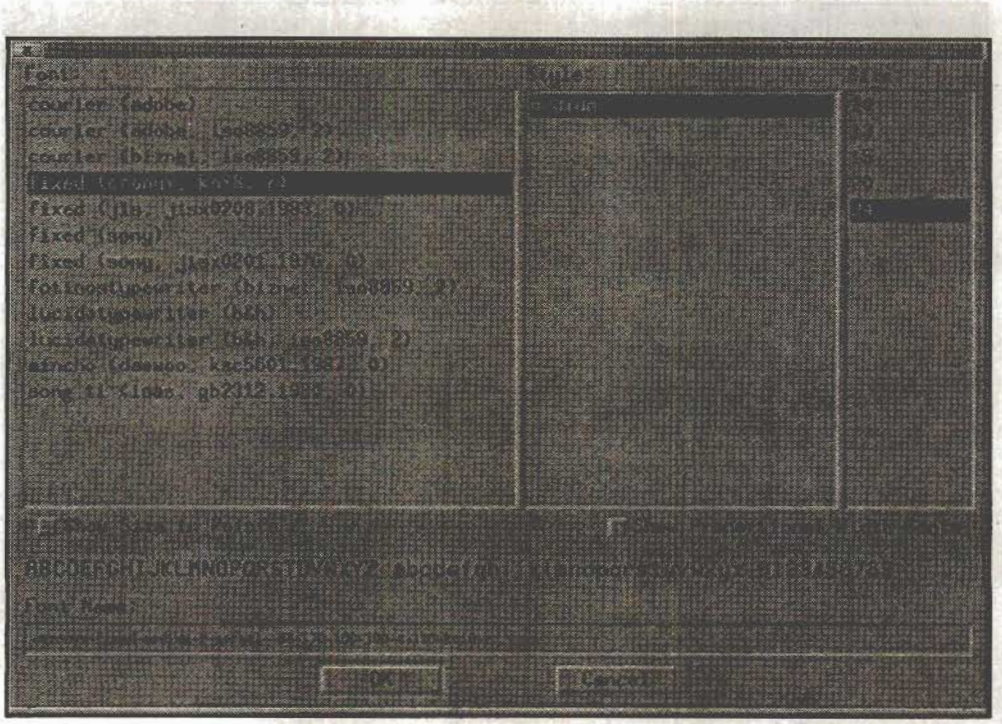


Рис. 21.15. Выбор шрифта и его атрибутов

Что характерно (вернее, не характерно для большинства текстовых редакторов, оперирующих обычно только моноширинными гарнитурами): хотя по умолчанию в списке присутствуют моноширинные шрифты, можно включить также показ шрифтов пропорциональных и использовать их в свое удовольствие.

Подчеркну, что таким образом можно определить шрифт только для экранного представления набираемого текста. На шрифты интерфейсных элементов, а также системы помощи это не окажет никакого воздействия. Разумеется, и они могут быть подобраны по вкусу, но об этом — несколько ниже.

Customize Menus

Эта команда дает возможность настроить три элемента — пункты **Shell** и **Macro** главного меню, а также контекстное, или, в терминологии NEdit, фоновое, меню (**Window Background Menu**), вызываемое правой кнопкой мыши (рис. 21.16).

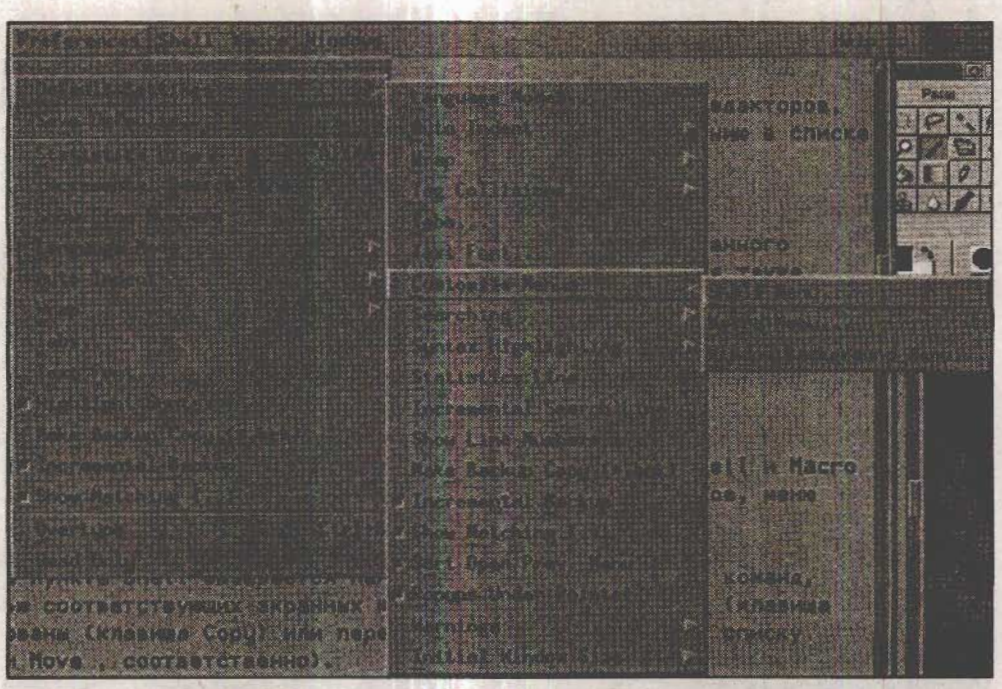


Рис. 21.16. Настройка меню **Shell**, **Macro** и контекстного

При настройке пункта **Shell** вызывается диалоговое окно со списком имеющихся команд (рис. 21.17), которые с помощью соответствующих кнопок могут быть удалены (кнопка **Delete**), скопированы (кнопка **Copy**) или перемещены вверх или вниз по списку (кнопки **Move^** и **Move˅** соответственно).

В расположенном ниже поле командной строки (**Shell Command to Execute**) содержится собственно код скрипта командной оболочки. Здесь можно непосредственно вручную отредактировать соответствующую команду.

Например, в комплекте NEdit предусмотрен скрипт для проверки орфографии (пункт **spell**). Каковой, вызывая стандартную программу *ispell* в окне терминала, имеет вид

```
cat>spellTmp; xterm -e ispell -x spellTmp; cat spellTmp; rm spellTmp
```

и, естественно, понимает только английский язык. Однако обучить его русскому языку (или любому другому, при наличии соответствующего словаря) — трудов не составит. Для этого достаточно указать в командной строке параметр `-d russian` (предписывающий обращаться при спеллинге (проверке орфографии) именно к русскому, а не какому другому словарю) — и все в порядке.

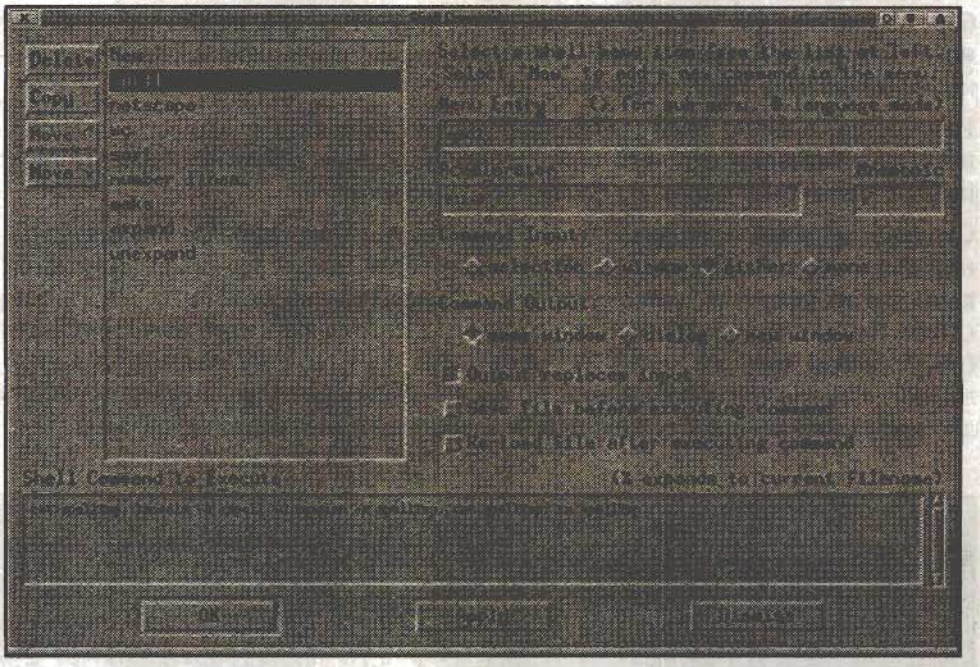


Рис. 21.17. Панель настройки меню **Shell**

Заодно, при желании, можно указать и параметры окна терминала (цвет текста и фона, геометрию и т. д.). Или — просто заменить вызов `xterm` на ту терминальную программу, которая вам больше нравится. В результате чего скрипт проверки орфографии примет примерно следующий вид:

```
cat>spellTmp; rxvt -e ispell -d \
russian -x spellTmp; cat spellTmp; rm spellTmp
```

Кроме того, в строке **Menu Entry** (Ввести в меню) можно изменить название скрипта (именно содержимое этой строки и выступает в качестве пункта меню) или организовать иерархическое меню (с помощью знака >), а также привязать скрипт к какому-либо языку (для чего имя его маркируется знаком @ с указанием на последний, например **@C++**); в последнем случае скрипт будет фигурировать в меню только в том случае, если выбрать соответствующий языковый режим.

Кроме того, в поле **Accelerator** (Ускоритель) за скриптом можно закрепить некую комбинацию "горячих" клавиш. А поле **Mnemonic** (Мнемоника) предназначено для ввода литеры, которая будет в меню выделена и послужит для быстрого доступа к команде (с помощью комбинации клавиш <Alt>+ +<литера>). Можно изменить и ряд условий исполнения команды, о чем будет разговор в соответствующем разделе.

И так можно поступить (при необходимости) с любым из имеющихся в наборе скриптов. Не следует только забывать, завершив редактирование одного, нажимать клавишу **Apply** (Принять) перед переходом к следующему — иначе изменения будут утеряны.

Разумеется, в меню **Shell** (Оболочка) можно включить и скрипты собственного изготовления, для чего в списке имеется пункт **New**. Однако это будет нашим занятием в одном из следующих разделов.

Сходным образом настраивается и меню **Macro** (Макросы). На предназначенной для этого панели (рис. 21.18) также имеется список макрокоманд с кнопками для их удаления, копирования и перемещения вверх и вниз, поле имени (**Menu Entry**) с возможностью создания многоуровневых иерархических меню (посредством того же знака >), привязкой к языковому режиму и закрепленной клавишной комбинации, а также обширное поле, предназначенное для кода макроса и доступное для непосредственного редактирования.

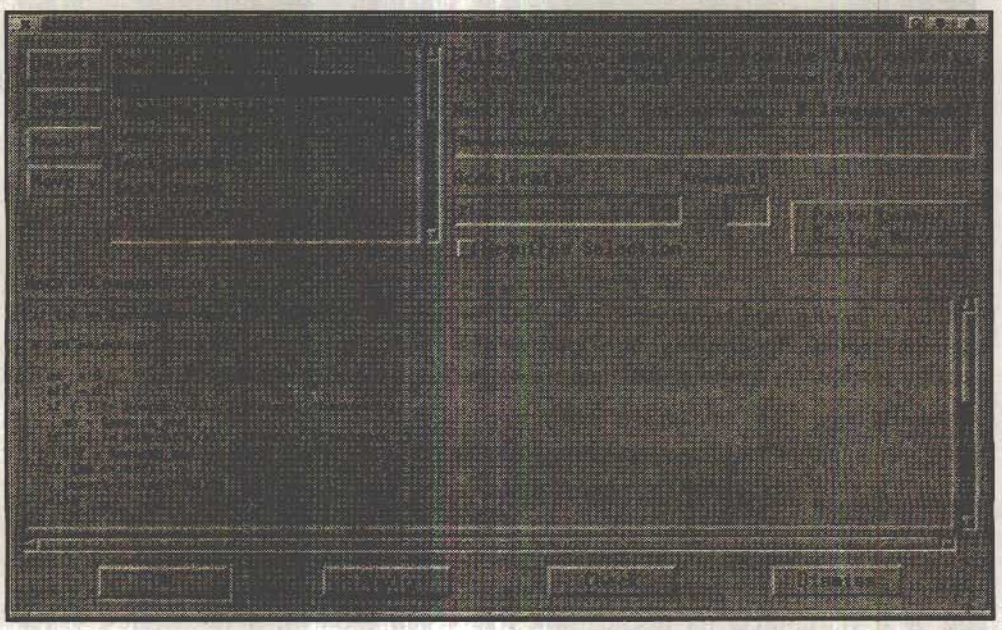


Рис. 21.18. Панель настройки меню **Macro**

Выбрав из списка макросов пункт **New** (Создать), можно нарастить меню собственными командами на языке NEdit Macro. Во-первых, написав в поле **Macro Command Execute** (Исполняемая макрокоманда) соответствующий код вручную, во-вторых, посредством кнопки **Paste Learn/Replay Macro** (Поместить записанное) поместив туда запротоколированную (с помощью команды меню **Macro — Learn Keystrokes** (Протоколирование) или клавишной комбинации <Alt>+<K>) последовательность действий. Впрочем, создание макросов будет подробно рассмотрено в следующем разделе.

Абсолютно аналогичным образом можно настроить и контекстное (фоновое) меню, вызываемое щелчком правой кнопки мыши на выделенном фрагменте (рис. 21.19). Как уже говорилось, по умолчанию оно включает привычные для Windows-мигранта пункты **Undo** (Отмена), **Redo** (Возврат), **Cut** (Вырезать), **Copy** (Копировать) и **Paste** (Вставить), которые с помощью кнопок **Delete** (Удалить), **Copy** (Копировать), **Move^** (Переместить вверх) и **Move˅** (Переместить вниз) можно удалить, скопировать и переместить.

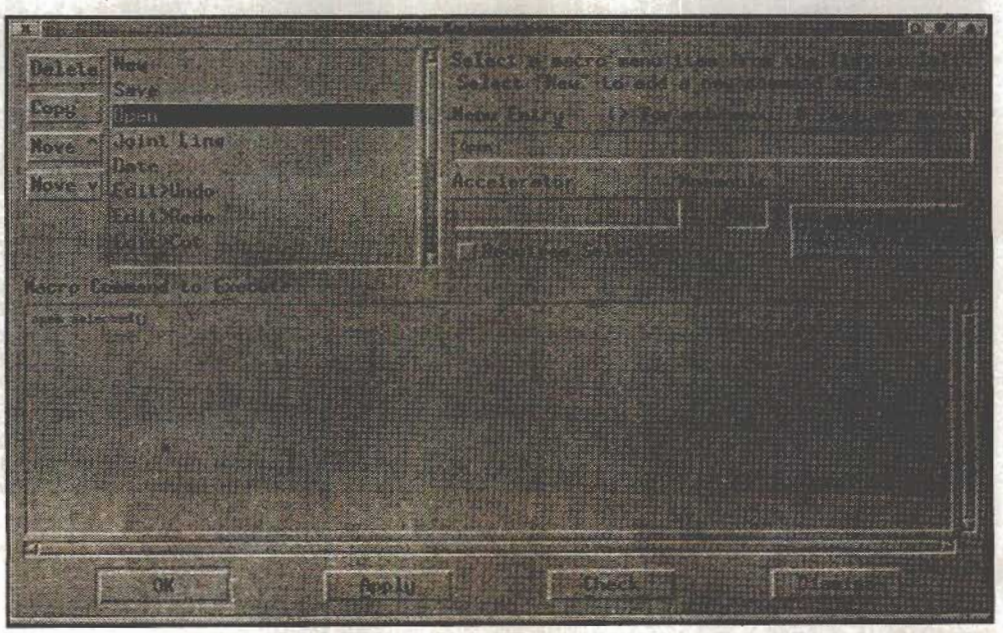


Рис. 21.19. Панель настройки **Window Background Menu**

А избрав в списке пункт **New**, можно пополнить контекстное меню собственными командами, написанными на языке NEdit Macro вручную или посредством протоколирования действий.

Searching

Покончив с настройками меню, можно заняться определениями параметров поиска. Я уже говорил, что возможности для этого в редакторе NEdit весьма обширны. А настраиваемость придает им еще и гибкость (рис. 21.20). В пункте **Searching** (Поиск) присутствуют две опции:

- **Verbose**, т. е. включение поиска многословных фрагментов;
- **Keep Dialogs Up** (Принять к дальнейшему), включение которой позволяет осуществлять непрерывный поиск последующих вхождений (по умолчанию для поиска следующего вхождения требуется выбрать в главном меню команду **Search — Find Again**).

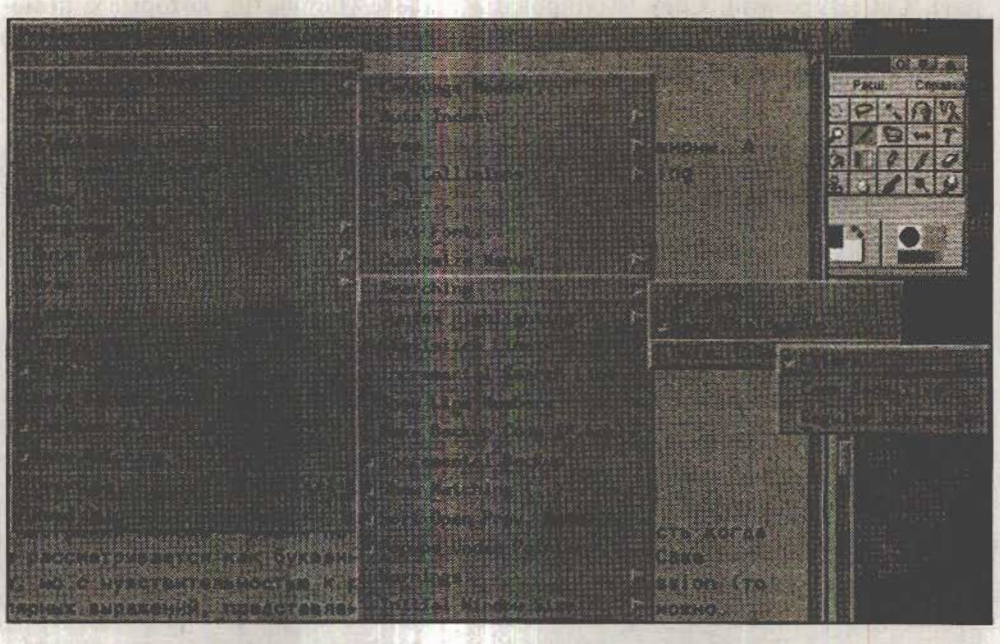


Рис. 21.20. Настройка условий поиска и замены

Третий пункт настройки поиска — режим по умолчанию: **Literal** (Символьный) (т. е. когда искомое значение рассматривается как буквенная последовательность), **Case Sensitive** (Различение регистра) (то же, но с чувствительностью к регистру) и **Regular Expression** (Регулярные выражения) (т. е. поиск регулярных выражений, представление о синтаксисе которых можно получить из соответствующего пункта меню **Help**).

Syntax Highlighting, Statistics Line, Incremental Search Line и Show Line Numbers

Эта группа настроек — просто флажки, с помощью которых можно показать или скрыть:

- подсветку синтаксиса (**Syntax Highlighting**) языка, соответствующего данному режиму;
- строку состояния (**Statistics Line**);
- строку поиска (**Incremental Search Line**);
- нумерацию строк (**Show Line Numbers**).

Правда, для пункта **Syntax Highlighting** (Подсветка синтаксиса) настройке поддается также **Recognition Patterns** (Распознание шаблонов) и **Text Drawing Styles** (Определение стиля подсветки). Первое — это определение языковых конструкций, которые должны выделяться подсветкой, а второе — приписывание каждому элементу языковых конструкций цвета и шрифтоначертания. Для языка HTML, например, можно определить различные цвета тэгов, их атрибутов и значений последних, комментариев, метатэгов и т. д.

Make Backup Copy и Sort Open Prev. Menu

Это также простые флажки. Первый разрешает или запрещает создание резервных копий редактируемого файла. Включение его приводит к тому, что при перезаписи измененного документа предыдущая его версия сохраняется в виде файла с расширением bck.

С помощью **Sort Open Prev. Menu** (Сортировка недавнего) можно включить алфавитную сортировку ранее открывавшихся файлов, входящих в список **File — Open Previous** (Открыть недавнее). При выключении этой опции файлы в нем следуют в хронологическом порядке.

Warning

Этот пункт представляет собой конфигурирование сообщений при закрытии файла или выходе из редактора. Здесь имеются всего два флажка. Первый отвечает за выдачу предупреждения о несохраненности изменений в закры*ваемом файле, второй же — разрешает или запрещает сообщение об изме*нении установок по умолчанию при закрытии программы. На мой взгляд, оба — далеко не лишние, и не худо бы включить их по умолчанию.

Initial Window Size

Как не трудно понять, здесь задается размер окна по умолчанию, возникающего при запуске NEdit. Предусмотрено пять вариантов такового:

- 24 на 80 знаков;
- 40 на 80 знаков;
- 60 на 80 знаков;
- 80 на 80 знаков;
- настройка геометрии (**Custom...**).

Первые четыре в комментариях не нуждаются, пятый же вариант позволяет задать произвольное количество строк и столбцов. Подчеркну только, что речь идет о размере окна именно в знаках, а не в пикселах, поэтому реальный размер окна зависит от используемых по умолчанию гарнитуры и кегля шрифта.

Конфигурирование .Xdefaults

Надеюсь, из приведенного описания становится ясно, что с помощью команды меню **Preferences — Defaults Settings** (Умолчания) в NEdit можно настроить почти все. Единственно, следует не забыть сохранить произведенные изменения, дабы сделать их перманентными командой **Preferences — Save Defaults** (Установить по умолчанию).

Однако способов изменения некоторых параметров с помощью команд меню обнаружить не удается. Так, в первозданном (и, возможно, не отвечающем вашему эстетическому идеалу) виде остались цвета фона и курсора, шрифт надписей меню и цвет его линейки, положение полосы прокрутки и еще кое-что.

Отчаиваться не следует — на сей предмет существует метод левой резьбы. То есть — ручной правки конфигурационного файла /$HOME/.Xdefaults, описывающего особенности X-приложений.

Секции для NEdit в нем нет. Однако создать ее (хотя бы собственными средствами этого редактора) труда не составит. Для этого достаточно вставить такую конструкцию:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! nedit

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

где строки с восклицательными знаками, естественно, не обязательны и служат всего-навсего ограничителями секции, а строка `! nedit` — комментарий, дабы не забыть, о чем, собственно, идет речь.

Теперь дело за малым — нужно создать наполнение секции. Это можно сделать по наитию — методом проб и ошибок. А можно обратиться к системе помощи — тому самому ее пункту, где говорится о конфигурировании NEdit и, в частности, о ресурсах X Window: **Help — Customizing — X Resources** (Ресурсы системы X).

Здесь можно найти список почти всех параметров, которые не подлежат изменению через меню **Preferences** (Настройки), но которые в принципе можно (хотя и не все — нужно) изменить.

```
nedit.Shell: /bin/csh
```

Указание на оболочку командной строки, скрипты которой выполняются через меню **Shell**.

```
nedit.wordDelimiters:.,/\\`'!@#%^&*()-=+{}[]":;<>?
```

Знаки, маркирующие (в дополнение к пробелам и табуляции) границы между словами при их выделении мышью (двойным щелчком на слове) или перемещении курсора комбинацией клавиши <Ctrl> и стрелок управления курсором

```
nedit.remapDeleteKey: False
```

Указание значения `True` переопределяет клавишу <Delete> как <Backspace>.

```
nedit.stdOpenDialog: False
```

Возвращение значения `True` устанавливает стандартный для приложений, основанных на Motif, стиль диалоговых панелей.

```
nedit.bgMenuButton: Shift[Control][Meta][Alt][Btn3Down]
```

Способ доступа к контекстному (или фоновому) меню. По умолчанию для этой цели служит, как и в Windows, правая (стандартно для UNIX — третья, отсюда **Button 3**) кнопка мыши. Однако можно использовать и любой из четырех других вариантов.

```
nedit.maxPrevOpenFiles: 30
```

Максимальное количество файлов в списке, вызываемом командой меню **File — Open Previous** (Открыть недавнее) (т. е. ранее открывавшихся или создававшихся). Насколько мне удалось понять, значение это сверху ограничено только размерами дисплея и его разрешением — при большом количестве список уходит за пределы видимости. Список открывавшихся ранее документов хранится в конфигурационном файле /$HOME/.neditdb.

```
nedit.printCommand: (system specific)
```

Этот параметр и серия следующих за ним (начинающихся с nedit.print) определяют условия печати документов из редактора NEdit. По умолчанию все они используют системные установки. Насколько я понимаю, лучше их таковыми и оставить, если нет веских причин для иного решения.

```
nedit.multiClickTime: (system specific)
```

Время ожидания второго щелчка кнопкой мыши, в течение которого он считается за двойной. Здесь логично сохранить системную установку — не зря же вы ее делали для всех приложений.

```
nedit*scrollBarPlacement: BOTTOM_LEFT
```

Эта строка описывает положение и вид полосы прокрутки. При значении в примере (как ни странно, это не значение по умолчанию — уже при первом запуске NEdit полоса прокрутки располагается справа) она должна располагаться вдоль левой границы окна и прокручиваться вперед смещением ползунка сверху вниз. Другие возможные значения: `BOTTOM_RIGHT`, `TOP_LEFT` и `TOP_RIGHT`; в двух последних случаях прямой скроллинг осуществляется движением ползунка снизу вверх.

```
nedit*text.autoWrapPastedText: False
```

По умолчанию запрещает автоматический перенос слов при наборе текста. Лучше так и оставить — различные опции переноса слов для каждого конкретного случая проще настроить командой меню **Preferences — Wrap** (Перенос слов) (или **Preferences — Default Settings — Wrap** (Перенос слов)), как было описано выше.

```
nedit*text.foreground: black
```

Переходим к самому интересному — настройке цветовой гаммы. Это — цвет шрифта набираемого текста. Смысл значения, думаю, понятен, а от рекомендаций воздержусь — на шрифт и цвет товарищей нет.

```
nedit*text.background: white
```

В этой строке определяется цвет фона редактируемого поля.

```
nedit*text.selectForeground: black
```

А здесь определяется цвет текста выделенного первичным способом (левой кнопкой мыши или стрелками указателя курсора при нажатой клавише <Shift>) фрагмента.

```
nedit*text.selectBackground: gray80
```

Определение цвета фона выделенного первичным способом фрагмента. Следует заметить только, что число после наименования цвета обозначает яркость (saturation) его в процентах. То есть `gray80` — это светло-серый, а, скажем, `gray20` — темно-серый. Во всех относящихся к цветовой гамме пунктах возможны также значения имени цвета с прилагательными — `light` или `dark`.

```
nedit*text.highlightBackground: red
```

Насколько установлено эмпирически, определяет цвет подсветки парных элементов (например, открывающей скобки при фиксации курсора на закрывающей).

```
nedit*text.cursorForeground: black
```

Ну, понятно, просто цвет курсора.

```
nedit*text.lineNumForeground: gray47
```

Цвет, которым передается нумерация строк (если эта опция включена).

```
nedit*text.blinkRate: 600
```

А это частота мерцания курсора в миллисекундах. Если установить значение 0, курсор, к счастью, мерцать не будет вообще.

```
nedit*foreground: black
```

Задает цвет текста меню и диалоговых панелей.

```
nedit*background: gray70
```

А здесь определяем цвет фона меню, диалоговых панелей и полосы прокрутки.

```
nedit*fontList: helvetica-bold-14
```

В этой строке можно, наконец, определить гарнитуру, начертание и кегль шрифта интерфейсных элементов. За исключением шрифта в полях, скажем, имен файлов при их открытии или сохранении, или в полях ввода кода и параметров в диалоговых окнах настройки меню, например (ни в том, ни в другом случае изменить шрифт мне не удалось — видимо, это следует делать через общие настройки Mottif-приложений). Не оказывает влияния эта строка (также как и настройка через меню) и на шрифт вывода сообщений системы помощи, о чем скажу ниже.

Далее следует предложение использовать всякого рода расширения имен элементов, такие, как .background и т. п., для изменения цветов, шрифтов и прочих характеристик следующих далее элементов.

```
nedit.statsForm
```

Предлагает определить вышеуказанные характеристики для статусной строки и строки поиска. При желании одна характеристика может распространяться на оба элемента, для этого вместо "." следует использовать "*". Например, строка

```
nedit*statsForm*background: gray
```

задает для них единый серый цвет фона.

```
nedit*menuBar
```

В этой и нижеследующих строках можно по той же модели определить характеристики линии главного меню,

```
nedit*textHorScrollBar
```

... полосы горизонтальной...

```
nedit*textVertScrollBar
```

... и вертикальной прокрутки.

```
nedit*helpText
```

Эта строка, якобы, позволяет определить характеристики окна системы помощи. Однако у меня она на шрифт Help-сообщений никакого действия не оказала.

Тем не менее, эту проблему удалось победить методом ползучего эмпиризма, а именно, поместив собственноручно сконструированную строку

```
nedit*font: -cronyx-fixed-medium-r-*-*-20-*-*-*-*-*-koi8-r
```

Не возьмусь судить, насколько она синтаксически грамотна, но, по крайней мере, после этого шрифт сообщений в окне помощи действительно изменился.

Таким образом, в секцию `!nedit` файла /$HOME/.Xdefaults можно внести изрядное количество строк. Но можно — не значит нужно. Достаточно ограничиться только теми строками, которые определяют характеристики, не устраивающие вас по умолчанию.

Файлы .nedit и .neditdb

Таким образом, редактированием файла /$HOME/.Xdefaults удается настроить то немногое, что не получается настроить через главное меню. Однако в домашнем каталоге пользователя можно обнаружить еще два конфигурационных файла, имеющих отношение к Nedit, — ~/.nedit и ~/.neditdb.

Не факт, что возникнет необходимость их ручной правки, но полноты картины для вкратце опишу их структуру.

Содержание файла ~/.nedit составляет описание тех самых параметров, которые настраиваются с помощью команды меню **Preferences — Default Settings** (Умолчания). К коему и следует, как гласит комментарий, прибегать для его изменения. Тем не менее он доступен и для ручного редактирования, что, как и всякую свободу выбора, следует расценить только положительно.

Начальная (после комментариев) строка файла — указание на номер версии:

```
nedit.fileVersion: 5.1
```

Затем следуют секции описания настраиваемых меню **Shell** (Оболочка), **Macro** (Макросы) и контекстного:

```
nedit.ShellCommands: \
    spell:Alt+B:s:EX:\n\
  ...
nedit.macroCommands: \
    Headers>header1:F1::: {\n\
  ...
nedit.bgMenuCommands: \
    Undo:::: {\n\
  ...
```

соответственно. Описывать их содержимое я не буду. Скажу только, что первая строка после заглавия секции содержит название скрипта или макроса (в том самом виде, как оно отображается в соответствующем меню (так, для приведенного макроса — двухуровневом, `Headers>header1`). Затем, разделенные двоеточиями, следуют всякого рода дополнительные сведения, такие как закрепленная клавишная комбинация или мнемоническое обозначение.

Следующие секции

```
nedit.highlightPatterns: C:Default\n\
    C++:Default\n\
   ...
nedit.languageModes:     C:.c.h:::::::"./\\`'!|@#%^&*()-
=+{}[]"":;<>?~"\n\
    C++:.cc.hh.C.H.i.cxx
   ...
nedit.styles:      Plain:black:Plain\n\
    Comment:gray20:Italic\n\
   ...
```

определяют разнообразные параметры (подсветка, отступы и т. п.) для различных языковых режимов.

За этим следуют опции переноса слов, автоматических отступов, подсветки синтаксиса и прочие, подробно рассмотренные выше, в разделе о настройке с помощью команды меню, смысл их достаточно прозрачен. Завершающие разделы — описание геометрии открываемого окна, шрифтов редактируемого текста и т. п., — также не должны вызвать вопросов. И посему, скажем, при необходимости можно изменить, например, кегль шрифта, не обращаясь к меню. Другое дело, что вряд ли такая необходимость возникнет...

Ну, а файл ~/.neditdb — это просто список (задумчиво называемый базой данных) ранее открывавшихся файлов с указанием полных абсолютных путей. Что характерно, в списке пункта меню **File — Open Previous** (Открыть недавнее) они даются без дублирования — все же буквы db в названии присутствуют не зря.

Макросы и Shell

Надеюсь, мне удалось продемонстрировать если и не все богатство штатных возможностей редактора NEdit, то — изрядную его долю. Однако величие программы в том, что штатными средствами его функциональность не исчерпывается. Поскольку может быть почти неограниченно наращена с помощью внутреннего языка макрокоманд — раз, и с помощью подключения скриптов командной оболочки — два.

Вкратце о возможности создания собственных макросов (путем написания их кода руками) я уже упоминал. Однако есть и другой способ, который для начала может оказаться более простым — протоколирование действий. Для этого требуется всего-навсего:

1. Включить режим протоколирования (командой меню **Macro — Learn Keystrokes** или комбинацией клавиш <Alt>+<K>).
2. Произвести посредством клавиатуры (но не с помощью мыши!) все необходимые действия в той последовательности, в какой мы хотим их сохранить на будущее.
3. Завершить режим протоколирования (командой меню **Macro — Finish Learn** или повторным нажатием комбинации клавиш <Alt>+<K>).
4. В случае ошибки при наборе команд — прервать протоколирование (командой меню **Macro — Cansel Learn** (Конец записи)) и, включив протоколирование по новой, повторить требуемые действия уже без ошибок.

Запротоколированная последовательность действий будет функционировать в течение данного сеанса работы с NEdit. И может быть вызвана первый раз командой меню **Macro — Reply Keystrokes** (Выполнить записанное) и, в дальнейшем, командой **Macro — Repeat** (Повторить выполнение). Однако при следующем запуске NEdit будет утрачена.

Чтобы этого не случилось, следует сохранить ее в меню для потомства. Для этого, успешно завершив протоколирование данного макроса, отправляемся в меню **Preferences** (Настройки), выбираем там пункт **Default Settings** (Умолчания), а в нем — опцию **Customize Menu** (Настройка меню). Каковая, в свою очередь, обнаруживает, как уже говорилось, пункт **Macro Menu** (Меню макросов), вызывающий диалоговое окно **Macro Commands** (Макрокоманды).

В списке доступных команд (в левой части окна) фиксируемся на пункте **New** (Новый). В поле **Menu Entry** (Ввести в меню) задаем название команды и, при необходимости, название более высокого уровня для группы сходных команд, например, таким образом:

```
Group commands>Command1
```

(без пробелов перед и после знака >). Здесь же можно приписать команду к какому-либо из доступных языковых режимов, например,

```
Group commands>Command1@SGML/HTML
```

будет указывать, что данная команда активизируется только при выборе языкового режима **SGML/HTML**.

Затем заполняем поле **Accelerator** (Ускоритель). Здесь за нашей командой можно закрепить какую-либо клавишу (из числа, например, функциональных или Windows-клавиш) или их комбинацию: например, сочетание клавиш <Alt> или <Ctrl>, возможно, совместно с <Shift> плюс какая-либо буква, желательно, имеющая мнемонический смысл.

Для этого достаточно просто зафиксировать курсор в поле **Accelerator** (Ускоритель) и нажать требуемую клавишу (например, <F12>) или их комбинацию (скажем, <Ctrl>+<литера>). Следует только внимательно следить, чтобы эта клавишная комбинация не использовалась для вызова штатных функций NEdit: в этом случае приоритет в любом случае будет за последними, а, возможно, клавишная комбинация не будет вызывать ни старого, ни нового закрепленного действия.

Вслед за этим можно заполнить поле **Mnemonic** (Мнемоника), введя в него какую-либо букву из имени нашей команды, как оно указано в поле **Menu Entry** (Ввести в меню). При вызове меню буква эта будет подчеркнута и, теоретически, может использоваться для быстрого вызова команды (сочетанием ее с клавишей <Alt>). Однако на деле это работает далеко не всегда. Вернее, почти всегда не работает, поскольку большинство букв латинского алфавита уже задействованы для штатных команд NEdit.

Наконец, последнее из предварительных действий — это установка, по потребности, флажка **Requires Selection** (Требует выделения). Он предназначен для команд, которые осуществимы только с предварительно выделенными фрагментами (типа копирования, вырезания и т. п.).

А вот теперь нажимаем кнопку **Paste Learn/Replay Macro** (Вставить записанное) (в правой части панели). И текст макроса волшебным образом появляется в поле **Macro Command to Execute** (Исполняемая макрокоманда). Где может быть любым образом отредактирован, дополнен, сокращен и т. п. После чего жмем кнопку **OK**, выходя их режима редактирования **Macro Menu** (Меню макросов) — и с удовольствием и пользой применяем новую функцию на практике. Если все работает нормально — сохраняем текущую ситуацию командой **Preferences — Save Defaults** (Установить по умолчанию) для увековечивания внесенных нами изменений.

Каковые, как уже говорилось, фиксируются в секции

```
nedit.macroCommands:
```

файла ~/.nedit из пользовательского каталога. Где также могут быть отредактированы непосредственно.

Таким образом можно насытить меню редактора NEdit командами для выполнения часто требуемых, но отсутствующих в штатном комплекте действий. Например, как будет показано ниже, для ввода тэгов HTML. Однако и этого может показаться недостаточно для полного счастья. Например, возникнет потребность в систематическом исполнении каких-либо внешних программ. Например, команд оболочки.

Конечно, на сей предмет можно прибегнуть к стандартному окну терминала — во-первых, к строке мини-терминала (вызываемой командой меню **Shell** (Оболочка) — **Execute Command** (Исполнение команд) или клавишной комбинацией (<Alt>+<X>)) — во-вторых, и к непосредственному вводу

команд в поле редактирования NEdit с последующим их запуском (командой меню **Shell — Execute Command Line** (Исполнение строки как команды)) — в третьих. Однако все это — не предел комфорта.

К тому же подчас необходимо, чтобы команда оболочки выполнялась в отношении редактируемого в данный момент в NEdit документа. Примером чего служит приведенная в предыдущем разделе команда `ispell` для проверки орфографии. И тут нам на помощь приходит возможность редактирования меню **Shell** (командой **Preferences — Default Settings — Customize Menu — Shell Menu** (Меню оболочки)).

Выбрав соответствующие пункты приведенной последовательности, вызываем диалоговое окно **Shell Commands** (Команды оболочки) и выбираем из списка доступных команд пункт **New** (Новый). Подобно тому, как это проделывалось для макросов, заполняем поля **Menu Entry** (Ввести в меню), т. е. вводим имя команды меню (также, возможно, иерархически построенного и привязанного к языковому режиму), **Accelerator** (Ускоритель) (для закрепления комбинаций клавиш) и **Mnemonic** (Мнемоника) (для выделения литеры быстрого вызова).

Затем определяемся с входными элементами команды в строке переключателей **Command Input** (Стандартный ввод). Здесь можно предписать команде исполняться для выделенного фрагмента (**selection**), текущего окна (**window**), каждого элемента документа (**either**) или ни для чего (**none**). Объяснить смысл каждого переключателя в общем виде я несколько затрудняюсь, но на практике смысл их в отношении каждой конкретной команды обычно достаточно прозрачен.

Аналогично — и с выходными элементами команды. Каковые могут быть показаны в текущем или новом окне, а также в виде диалогового окна.

Следующая ниже группа переключателей при установке в положение **On** предписывает:

- замещать элемент ввода элементом вывода (**Output replace input**);
- сохранять текущий файл перед выполнением команды (**Save file before executing command**);
- перечитывать его же после выполнения таковой (**Re-load file after execute command**).

Определившись и с этим, можно непосредственно задавать команду вместе со всеми требуемыми параметрами и их значениями. Это делается в поле ввода команды (**Shell Command to Execute**), точно так же, как в командной строке терминального окна или консоли.

Однако команду можно набрать и непосредственно в поле редактирования NEdit, после чего просто скопировать (с помощью мыши или клавишных комбинаций) ее в поле ввода команды. Преимущество последнего способа в том, что предварительно (с помощью меню **Shell Execute Command Line**)

можно проверить работоспособность команды и, руководствуясь сообщениями об ошибках, ее отредактировать. Можно, разумеется, и просто скопировать успешно запускаемую команду из окна терминала...

Завершив создание новой команды, принимаем введенные изменения (нажав экранную кнопку **Apply**) или выходим из режима редактирования **Shell** нажав **OK**. А выполнив проверку правильности созданной команды, сохраняем ситуацию (кнопкой **Save Defaults**) для использования в дальнейшем.

Пример применения: html-редактор своими руками

Не могу поручиться, что в предыдущем разделе внятно изложил возможности NEdit по созданию и редактированию макрокоманд и команд оболочки. И потому попробую проиллюстрировать эти возможности на конкретном примере — создании серии связных директив для составления не очень сложных (и преимущественно текстовых) html-документов.

При этом я прекрасно понимаю, что сказанное ниже способно вызвать у профессионального программиста лишь ироническую улыбку. Однако написал я это не показания крутизны для, а токмо к вящей славе редактора NEdit. Каковой позволяет настроить себя под конкретную задачу даже людям с моим уровнем подготовки.

Итак, перед нами (т. е. вашим покорным слугой) стоит задача составления "из головы" достаточно длинных и относительно структурированных документов, предназначенных в первую голову для размещения в Сети. При этом не исключается возможность, что они в дальнейшем будут воплощены и на бумаге в виде журнальных или книжных публикаций (для чего может потребоваться импорт документа в текстовый процессор или программу верстки). Что нужно для комфортного выполнения этой работы?

Путей — несколько. Можно:

- набирать текст в текстовом процессоре (типа StarWriter из комплекта StarOffice) с дальнейшим экспортом в формат HTML; тем, кто видел html-код, генерируемый текстовыми процессорами, ясно, почему этот вариант я обсуждать далее не желаю;
- набирать текст в визуальном Web-редакторе типа Netscape Composer; однако замечание из пункта первого имеет место быть и здесь (не говоря уже о личной антипатии к Netscape как средству редактирования Web-страниц);
- набирать текст в любом симпатичном и подходящем по возможностям html-редакторе с одновременным расставлением требуемых тэгов штатными его средствами; однако далеко не всегда такие редакторы хорошо приспособлены для столь "примитивной" работы; к тому же, подобно то-

му, как изобилие форматирующих возможностей текстового процессора нарушает течение оригинальной творческой мысли, всякого рода панели с кнопками тэгов не способствуют концентрации мысли на содержательной стороне документа; к тому же оба известных мне полнофункциональных html-редактора под Linux обладают одним общим недостатком: они позволяют набирать текст либо в одну строку (что неудобно визуально), либо с переносом слов через фиксированное количество знаков с образованием символа возврата каретки (что, отнюдь, не полезно при импорте в текстовый процессор);

- набирать текст в обычном текстовом редакторе с последующим помещением в редактор кода HTML, где и осуществляется его (текста) разметка; что, помимо сложностей с двухэтапным созданием документа, приводит к появлению двух содержательно идентичных (по идее) файлов, а какой из которых является более актуальным — удается вспомнить не всегда.

И потому оптимальным представляется составление документа в текстовом редакторе с одновременной разметкой html-тэгами. Но проставлять их вручную, даже если помнишь все наизусть, — занятие достаточно нудное. И посему возникает естественное желание его автоматизировать. Благо, NEdit с его развитым макроязыком и простыми средствами его применения такую возможность обеспечивает.

Для начала определимся, какие тэги требуются для каждодневного использования. Это:

- во-первых, заголовки нескольких (скорее всего первого-третьего) уровней;
- во-вторых, тэги образования абзаца `<p>` и разрыва строки `<br>`;
- в-третьих, структурные тэги `<strong>`, `<emphasis>` и `<preformatted text>`;
- в-четвертых, всякого рода списки (маркированные, нумерованные, списки с определениями) и их элементы;
- в-пятых, естественно, гиперссылки и именованные якоря (`<a href>` и `<a name>` соответственно).

Вот в такой последовательности и будем создавать соответствующие макросы. Включаем режим протоколирования (командой меню **Macro — Learn Keystrokes** или комбинацией клавиш <Alt>+<K>) и просто-напросто набираем с клавиатуры последовательность символов

```
<h1></h1>
```

Для обеспечения непрерывности набора логично вернуть курсор в позицию между открывающим и закрывающим тэгами, для чего соответствующее количество раз нажимаем клавишу <Left>. После чего завершаем протоколирование (командой меню **Macro — Finish Learn** (Конец записи) или повторным нажатием <Alt>+<K>), командой меню **Macro Replay Keystrokes** (Воспроизведение макроса) (или — комбинацией <Ctrl>+<K>) проверяем пра-

вильность исполнения и переходим в меню **Preferences** (Настройки) — **Default Settings** (Умолчания) — **Customize Menu** (Настройка меню) — **Macro Menu** (Меню макросов).

В появившемся диалоговом окне **Macro Commands** (Макрокоманды) указываем в поле **Menu Entry** (Ввод в меню)

```
Headers>header1
```

в поле **Accelerator** (Ускоритель) приписываем команде клавишу вызова (например, <F1>, благо она не закреплена за вызовом помощи), при желании определяем мнемоническую литеру (хотя большого смысла в этом я не вижу) и смело жмем на экранную кнопку **Paste Learn/Reply Macro** (Вставить записанное). В результате в поле **Macro Command to Execute** (Исполняемая макрокоманда) появляется код вроде следующего:

```
insert_string("<")
insert_string("h")
insert_string("1")
insert_string(">")
insert_string("<")
insert_string("/")
insert_string("h")
insert_string("1")
insert_string(">")
backward_character()
backward_character()
backward_character()
backward_character()
backward_character()
```

Разумеется, никто не заставляет нас оставлять в неприкосновенности столь неуклюжую конструкцию. Заглянув в **Help** (Помощь), т. е. в ее раздел, где рассказывается о синтаксисе макроязыка NEdit, не трудно сообразить, что `insert_string` означает просто вставку некоей символьной последовательности. И написать проще и короче:

```
insert_string("<h1></h1>").
```

Что и будет означать вставку тэга заголовка первого уровня.

А ознакомившись с примерами макросов на сайте проекта, можно по их образу и подобию создавать свои собственные, достаточно сложные конструкции.

Поскольку проверку правильности макрокоманды мы уже произвели, нажимаем **Apply** (Принять) для ее включения в список и переходим к следующе-

му пункту нашей программы. Конечно, можно повторить протоколирование набора для тэгов заголовков всех уровней последовательно. Но проще — с помощью кнопки **Copy** скопировать только что созданный макрос, изменить в поле **Menu Entry** (Ввод в меню) его название с `header1` на `header2`, закрепить за ним клавишу <F2> и в поле **Paste Learn/Reply Macro** (Поместить записанное) отредактировать код вручную, заменив строки

```
insert_string("<H1></H1>")
```

на

```
insert_string("<H2></H2>")
```

И подобную процедуру повторить для заголовков всех потребных уровней (вряд ли число их превысит четыре).

Затем переходим к созданию макросов для наиболее часто требующихся при наборе тэгов параграфа и разрыва строки. Порядок действий — идентичный: протоколирование ввода тэга, вызов окна **Macro Commands** (Макрокоманды), описание имени команды (с учетом ее положения в иерархическом меню), закрепление свободной комбинации "горячих" клавиш, помещение "заученного" кода макроса в поле команд для исполнения, включение новой макрокоманды в список доступных. И так — для всех тэгов, которые представляются нам требуемыми при повседневном использовании...

Завершив создание макросов, следует не забыть увековечить их в меню с помощью команды **Preferences** (Настройки) — **Save Defaults** (Установить по умолчанию). После чего они будут доступны в последующих сеансах NEdit, поскольку размещаются в секции `nedit.macroCommands:` файла ~/.nedit.

Описанный процесс создания макросов, которых хватает в 90 случаях из 100, занимает считанные минуты. Что, опять-таки, подтверждает величие и силу редактора NEdit: можно представить себе, как способен настроить его для своих задач человек с достаточной программистской квалификацией...

Не трудно настроить по собственному желанию и контекстное меню. Чего в нем недостает? Да тех команд, которые все равно требуют обязательного выделения мышью. Например, вставки гиперссылки. Добавляем их по тому же сценарию (командой меню **Preferences** (Настройки) — **Default Settongs** (Умолчания) — **Customize Menu** (Настройка меню) — **Window Background Menu** (Контекстное меню).

Не забыв при этом про крайне полезную опцию — возможность открывать файл, выделив его имя в редактируемом тексте. Включив ее в наше контекстное меню, получаем возможность перемещения по серии связанных гиперссылками Web-страниц — согласитесь, такая возможность есть не в каждом специализированном html-редакторе...

Чего еще хотелось бы от несложного html-редактора? Ну конечно, визуализации производимых изменений в каком-либо внешнем браузере. Каковым, с большой долей вероятности, будет Mozilla текущей версии.

Для этого мы прибегнем к настройке меню **Shell** (**Preferences — Default Settings — Customize Menu — Shell Menu**). В соответствующей панели определяем, как рассказывалось ранее, поле имени команды; если есть желание просматривать плоды своих трудов в разных браузерах, можно создать иерархическое меню вроде

```
Browser>netscape
Browser>links
Browser>opera
```

оставляя, скорее всего, пустыми поля **Accelerator** (Ускоритель) и **Mnemonic** (Мнемоника) (не столь это частая процедура, вызов внешнего браузера, чтобы тратить на нее драгоценную клавишную комбинацию, коих и так не в избытке).

А затем вводим в поле **Shell Command to Execute** (Исполняемая команда оболочки) строку

```
xterm -e mozilla %
```

или иную любимую терминальную программу в качестве первого элемента. И ни в коем случае не забыв параметр `%`, предписывающей Netscape отображать именно редактируемый в NEdit файл.

Теперь разбираемся с переключателями. В команде **Command Input** (Стандартный ввод) логично отметить значение **None** (Нет) (иначе есть риск получить по закрытии Netscape пустой файл), в команде **Command Output** (Стандартный вывод) — значение **Same window** (В том же окне). Кроме того, не худо включить также опцию записи текущего файла перед выполнением команды (**Save file before executing command**).

Все, принимаем изменения (кнопкой **Apply**) и проверяем действие команды. Если все нормально, выбор из меню **Shell** пункта **netscape** должен вызвать появление сначала терминального окна, а затем — окна браузера Netscape Navigator, отображающего редактируемый файл в текущем его состоянии. Если да — сохраняем ситуацию (командой меню **Preferences — Save Defaults**) и с гордым и независимым видом пользуемся собственноручно созданным html-редактором...

Подводя итоги, можно сказать, что по своим возможностям и настраиваемости NEdit вплотную приближается к Vim или emacs (а учитывая возможность сколь угодно широкого наращивания функциональности пунктов меню **Shell** (Оболочка) и **Macro** (Макросы) — возможно, и не уступает). В то же время он предоставляет пользователю удобный и привычный интерфейс, лишенный архаических особенностей как emacs, так и vi.

Недостатки NEdit я уже отмечал: это отсутствие инструментальной панели и истинно многооконного интерфейса, а также средств управления проектом. Однако первые два замечания — спорны, а последняя функция в элемен-

тарном исполнении, т. е. в виде открытия и сохранения группы связанных файлов (глав книги, например, или серии Web-страниц), может быть реализована с помощью встраиваемых в меню сценариев оболочки и макрокоманд. Тем более, что зачатки управления проектом, в виде возможности открытия файла при выделении его в редактируемом тексте, имеются.

Короче говоря, если NEdit и не отвечает в полной мере идеалу текстового редактора, то вплотную к нему приближается. Дополнительное его достоинство — отсутствие привязки к конкретной графической среде (типа KDE или GNOME).

Дополнительные источники

За дополнительной информацией о редакторе NEdit следует обратиться к сайту проекта — **http://www.nedit.org**, где имеется очень подробное руководство пользователя (в том числе — с детальным описанием встроенного макроязыка), а также примеры макросов, которые могут быть приняты за основу собственного творчества.

Глава 22

Система X Window: обзор приложений

Разумеется, в X-сеансе можно не только редактировать тексты — количество приложений графического режима давно превысило число консольных программ и возрастает с каждым днем. Судя по всему, новые программы создаются ныне почти исключительно под систему X Window — и самого разного назначения, от системных утилит до мультимедийных приложений. Так что для подробного их описания потребовалась бы отдельная книга. И потому в данной главе будет дан лишь краткий обзор наиболее важных для конечного пользователя приложений графического режима, да и то далеко не всех.

Текстовые процессоры

С легкой руки компании Microsoft и ОС Windows, основными пользовательскими приложениями графического режима считаются офисные пакеты. По моему мнению, FreeBSD (и UNIX-системы вообще) — не самая подходящая среда для систематического решения офисных задач. Однако такие задачи неизбежно возникают перед пользователем любой профессии — составление отчетов, проектов, смет и прочих бюрократических документов суть объективная реальность, данная нам в ощущениях начальства. И FreeBSD располагает средствами для их решения.

Центральным компонентом офисной работы является составление текстовых документов, оформленных в соответствии с определенными требованиями. Конечно, это можно (а по моему мнению — и нужно!) делать в обычном текстовом редакторе, а должное оформление придавать с помощью средств разметки — на базе ли HTML, языка TeX или просто утилитой groff. Однако для многих пользователей такой способ по тем или иным причинам неприемлем, и потому для целей бюрократии и делопроизводства разработан класс программ, именуемых в русскоязычной терминологии текстовыми процессорами.

До избежание недоразумений следует отметить, что в англоязычной (а подчас и переводной) литературе по UNIX под текстовыми процессорами (text processor в англоязычных источниках) понимаются не WYSIWIG-системы обработки текста (типичный пример — всем известный MS Word или отечественный Lexicon), а потоковые (неинтерактивные) программы форматирования текста, в сущности, интерпретаторы какого-либо языка разметки (именно к этому классу относятся программы groff и TeX). Собственно же визуальные системы для работы с текстами именуются процессорами слов (word processor). Однако перевод последнего термина как "текстового процессора" в русскоязычной литературе (в первую очередь по DOS/Windows) настолько устоялся, что изменить сложившееся положение не представляется возможным. Просто об этом следует помнить при знакомстве с дополнительными источниками информации.

Итак, текстовые процессоры. Программ этого класса можно в некотором количестве обнаружить в коллекции портов FreeBSD. Это и старый, но простой в обращении Ted, и текстовые модули из облегченных, но, тем не менее, интегрированных пакетов Siag и Offix, и, главное, компоненты полнофункциональных офисных пакетов из интегрированных сред KDE и GNOME.

В первой текстовые процессоры представлены программой KWriter, входящей в состав интегрированного пакета KOffice. Он, как, впрочем, и весь пакет, находится (несмотря на номер текущей версии — 1.1) в стадии разработки, и вполне работоспособным считаться не может. Хотя, возможно, положение изменится к тому времени, когда вы будете читать эти строки.

Текстовый же процессор AbiWord из среды GNOME ныне достиг стадии если и не зрелости, то — полной работоспособности. Он поддерживает все функции программ этого класса и, что немаловажно, приобрел способность работать с кириллицей. Задерживаться на нем я не буду — благодаря стандартному интерфейсу и функциональности освоение его трудностей не представит.

AbiWord запускается из командной строки терминала (или — из мини-терминала) одноименной командой (AbiWord, обратите внимание на регистр символов). Кроме того, он может быть вызван из стартового меню GNOME. Наличие в системе последнего для работы AbiWord обязательно, но сам по себе он способен функционировать под любым иным менеджером окон.

После запуска AbiWord открывается окно с пустым безымянным файлом в рабочей области. В верхней части окна — строка главного меню и три инструментальные панели — **Стандартная**, **Панель форматирования** и **Дополнительная** (при правильно установленной русской локали меню и сообщения программы будут русскоязычными). В нижней части — строка состояния. В рабочем поле выведены линейки: горизонтальная и вертикальная. Это —

вид AbiWord по умолчанию, он может быть изменен соответствующими настройками.

Главное меню AbiWord включает пункты: **Файл**, **Редактирование**, **Вид**, **Вставка**, **Форматирование**, **Сервис**, **Окно**, **Помощь**. В меню **Файл** доступны действия: создание файла, открытие существующего, сохранение, сохранение под другим именем, установка параметров страницы, печать и выход.

Наибольший интерес представляют пункты **Открыть** и **Сохранить как...**, поскольку именно ими определяется потенциал AbiWord по обмену документами с другими приложениями.

AbiWord способен считывать документы в следующих форматах: собственном abw, в том числе и автоматически сжатом ZABW, DocBook в формате DBK, Microsoft Word — DOC, XHTML — HTML. Rich Text Format — RTF, UTF8 — UTF8, WML — WML и текстовом — TXT.

Сохранение созданных в AbiWord документов, кроме собственных форматов ABW и ZABW, возможно в виде файлов DBK (DocBook), HTML, RTF, UTF8, WML, текстовом, а также в формате LaTeX. Кроме того, AbiWord имеет возможность взаимного обмена документами с такими КПК, как Palm и Psion.

AbiWord корректно работает с кириллическими документами в текстовом формате. Возможно также считывание документов MS Word, RTF и HTML с символами кириллицы, однако в этих случаях не исключены как потеря форматирования (или его искажение), так и отказы считывания. Однако в целом работа с кириллицей улучшается с каждой новой версией AbiWord.

Меню **Редактирование** объединяет действия по выделению, вырезанию, копированию и вставке текстовых фрагментов, поиску, замене и переходу, отмене и возврату выполненных операций.

Командами меню **Вид** осуществляется включение или отключение элементов интерфейса — инструментальных панелей, линеек на рабочем поле, строки статуса. Здесь же устанавливается показ непечатаемых спецсимволов, колонтитулов, устанавливается масштаб показа документа (75, 100, 200%, по ширине или по размеру страницы, а также произвольный с шагом в 1%) и происходит переключение в полноэкранный режим, в котором отключены все элементы интерфейса, кроме строки главного меню.

Через меню **Вставка** выполняются следующие операции:

- вставка разрыва страницы или раздела;
- устанавливается нумерация страниц;
- вставляется дата и (или) время в различных форматах, а также поля — даты, приложения, номера;
- вставка символов из кодовой таблицы с учетом шрифта (аналог Character Map в Windows);
- вставка рисунков — в форматах PNG, BMP, SVG (Scalable Vector Graphics).

В меню **Форматирование** определяются:

- гарнитура, начертание и кегль шрифта;
- параметры абзаца (отступы, интервалы, выравнивание);
- формат списков, нумерованных и маркированных, и вид маркеров для последних;
- параметры страницы — формат бумаги, ориентация, поля;
- многоколоночное (одна, две или три колонки) представление текста и величина табуляции.

Предполагается возможность стилевого оформление текстов, но эта функция пока не реализована.

В меню **Сервис** — три пункта:

- **Правописание**, где можно вызвать программу проверки орфографии или включить автоматическую проверку;
- **Статистика** — подсчет слов, абзацев, символов и т. п., с опцией автообновления через фиксированный промежуток времени;
- **Настройки**, где включается/выключается показ инструментальных панелей, линеек и т. п.

В меню **Окно** можно создать новое (пустое) окно или просмотреть список открытых документов — каждый из них выводится в собственном окне со всеми управляющими элементами. В меню **Помощь** вызывается (online) система подсказки AbiWord.

Многие доступные через меню действия выполняются также с помощью кнопок инструментальных панелей. В *стандартной панели* собраны кнопки, отвечающие за создание, открытие и сохранение (в том числе и под другим именем) файла, печать, проверку орфографии, вырезание, копирование и вставку текстовых фрагментов, отмену и возврат операций, включение многоколоночного представления и масштабирование документа.

В раскрывающихся списках *панели форматирования* устанавливаются структурные элементы документа (заголовки, блочный текст, нормальный текст), гарнитура и кегль шрифта, кнопками — его начертание, выравнивание абзаца, оформляются нумерованные и маркированные списки.

Кнопками *дополнительной панели* устанавливаются шрифтовые эффекты (подчеркивание и перечеркивание), верхние и нижние индексы, горизонтальные отступы перед абзацами и межстрочные интервалы.

По щелчку правой кнопкой мыши на рабочем поле доступно контекстное меню, через которое можно вырезать, скопировать или вставить текстовый фрагмент, а также установить шрифтовое оформление и параметры абзаца.

В целом AbiWord не может заменить полнофункциональный текстовый процессор, но вполне пригоден для составления деловых документов небольшого объема и несложного оформления.

Наконец, в FreeBSD доступен и текстовый процессор интегрированного пакета StarOffice в различных его модификациях. Его версия 5.2 для Linux может быть запущена в режиме эмуляции этой ОС. Однако в настоящее время она перестала быть актуальной ввиду известных сложностей при работе с кириллицей.

И потому больший интерес представляет открытая модификация пакета — OpenOffice.org. Она основана на том же коде, что и коммерческий продукт фирмы Sun — StarOffice 6.0, но, в отличие от последнего, распространяется свободно и бесплатно. В момент, когда пишутся эти строки, стала доступной первая бинарная русифицированная версия пакета для ОС Linux, собранная компанией Altlinux (**http://www.altlinux.ru**) в рамках проекта OpenOffice.ru. Она является дистрибутивонезависимой и имеет достаточно скромные системные требования: единственно непременное из них — наличие библиотеки Glibc версии не ниже 2.2.0, И потому принципиальных сложностей в работе пакета под FreeBSD (в режиме воспроизведения Linux-программ) не предвидится.

В настоящее время осуществляется и настоящее портирование OpenOffice под FreeBSD — думаю, результаты его не замедлят проявиться в ближайшее время.

Текстовый процессор Lyx

Все упомянутые выше программы в той или иной мере являют собой попытку воспроизвести функции текстовых процессоров для Windows (или, по крайней мере, развиваются под их влиянием). Однако в FreeBSD представлен и традиционный для UNIX-систем текстовый процессор, имя которому — Lyx.

Этот редактор резко отличается от своих собратьев по клану. И чтобы понять его отличие, следует сказать пару слов о методике оформления документов вообще.

В принципе существуют два метода оформления документов — визуальный и структурный (или стилевой). При использовании первого элементы оформления оказываются первичными и достигаются изменением кегля шрифта, его начертанием и прочими сугубо внешними средствами. Структурный же метод — это придание элементам документа определенных стилей, отражающих его внутреннюю логику; внешнее представление стиля при этом может изменяться как угодно и, в общем случае, независимо от их содержания.

Конечно, каждый уважающий себя современный текстовый процессор (тот же MS Word, например) располагает развитыми средствами работы со стилями. Однако он не ориентирует на них пользователя: структура их пользовательского интерфейса такова, что средства визуального оформления буквально мозолят глаза, тогда как стилевые средства нередко остаются глубоко в тени (если вообще не за кадром). И это — сознательная установка разработчиков визуальных текстовых процессоров: в одном из руководств по MS Word мне попалась фраза вроде того, что стилевые средства предназначены для профессиональных пользователей (что можно понимать так, что простому народу они не нужны).

Результат такой политики — то, что многие пользователи текстовых процессоров типа MS Word (в том числе и те, кто по всем критериям попадает именно в категорию профессиональных), достигнув изрядной изощренности в визуальном оформлении, просто не имеют понятия о стилях, структурировании документа и тому подобных материях.

Не таков Lyx. Конечно, и в нем можно оперировать чисто визуальными методами. Однако структура его такова, что это — не напрашивающееся решение. Напротив, более очевидным представляется использование методов оформления структурного. Что, кроме явной экономии времени, способствует и дисциплине мысли — сужу по собственному опыту.

Именно поэтому Lyx представляется оптимальным средством для оформления документов в FreeBSD (и любых других открытых UNIX-подобных системах). Однако этим значение его не исчерпывается. Потому как Lyx — это не более, чем графическая надстройка над издательской системой TeX. И может выступать в роли полноценного средства для верстки оригинал-макетов, особенно структурированных документов нарративного характера.

Однако вернемся непосредственно к предмету разговора. Процессор Lyx — программа, основанная на библиотеке Motif, но может быть собрана с ее свободными аналогами — OpenMotif или Lesstif. В FreeBSD он входит в коллекцию портов, а в большинстве версий представлен и как бинарный пакет. Во многих дистрибутивах Linux бинарный пакет Lyx компилируется статически, однако в FreeBSD он обязательно требует установки библиотеки OpenMotif. Естественно, необходимым условием работы Lyx является также и установка TeX. Впрочем, и то, и другое в FreeBSD осуществляется автоматически — и при инсталляции пакета, и при сборке через систему портов.

Ряд отечественных дистрибутивов Linux (от компании Altlinux, например) включает в себя Lyx в полностью (и корректно) русифицированном виде. В FreeBSD для этого требуются некоторые (хотя и существенно упростившиеся в последнее время) действия. О них я скажу чуть позже, пока же будем считать, что таковые уже успешно выполнены.

После установки Lyx и его запуска (в X-сеансе, из командной строки или средствами оконного менеджера) на экране возникает главное окно про-

граммы с заставкой (рис. 22.1), которая исчезает по прошествии некоторого времени или после создания нового файла (автоматически при запуске таковой не возникает), открытия или импортирования (из текстового формата или документа TeX) существующего.

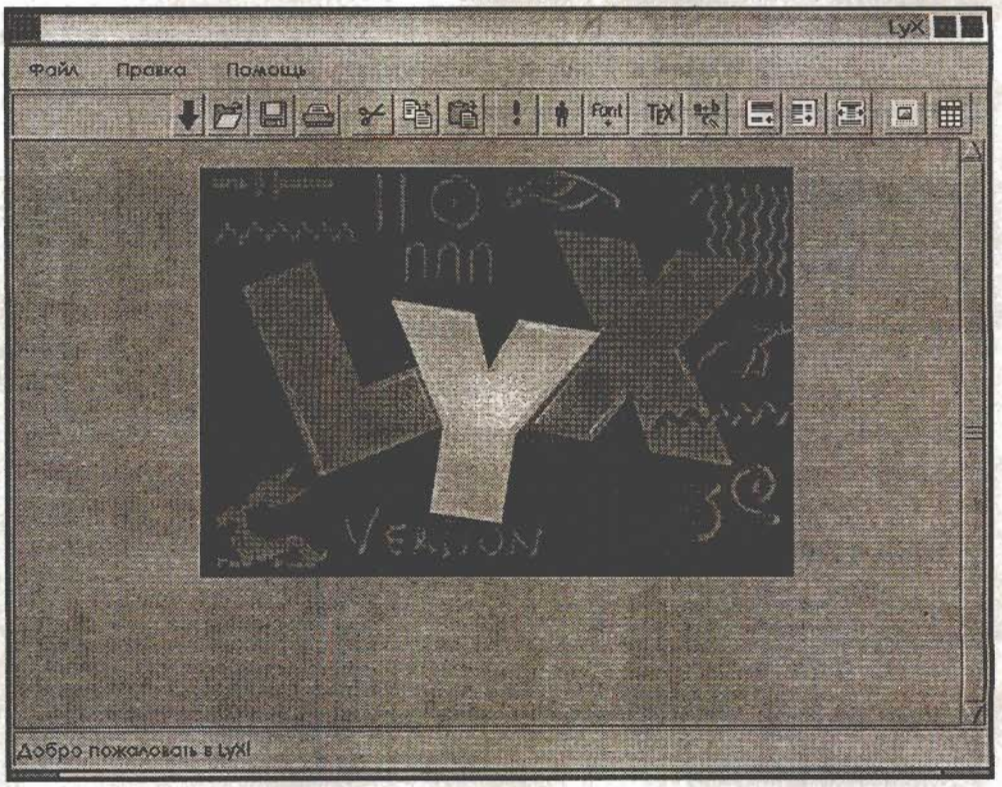


Рис. 22.1. Lyx после запуска

Одновременно с этим главное меню Lyx, в момент запуска содержащее только три пункта (**Файл**, **Правка**, **Помощь**), разворачивается во всю свою ширь (рис. 22.2) и включает следующие пункты:

- **Файл**, включающий обычные действия по созданию, открытию, сохранению и закрытию документов, их просмотру (через интерпретатор PostScript) и печати, а также экспорту и импорту;
- **Правка**, включающий отмену и возврат отмененного действия, копирование, вырезание и вставку фрагмента, поиск и замену и т. п.;
- **Вставить** — для рисунков, таблиц, сносок, примечаний и т. п.;

- **Формат** — изменение свойств символов, параграфов, страниц, а также документа целиком, и еще некоторые опции;
- **Просмотреть**, где собраны возможности визуализации выполненного форматирования;
- **Перейти**, обеспечивающий навигации по тексту;
- **Документы**, служащий для переключения между открытыми файлами (аналог пункта **Window** в Windows-программах);
- **Помощь**, весьма полная и обширная.

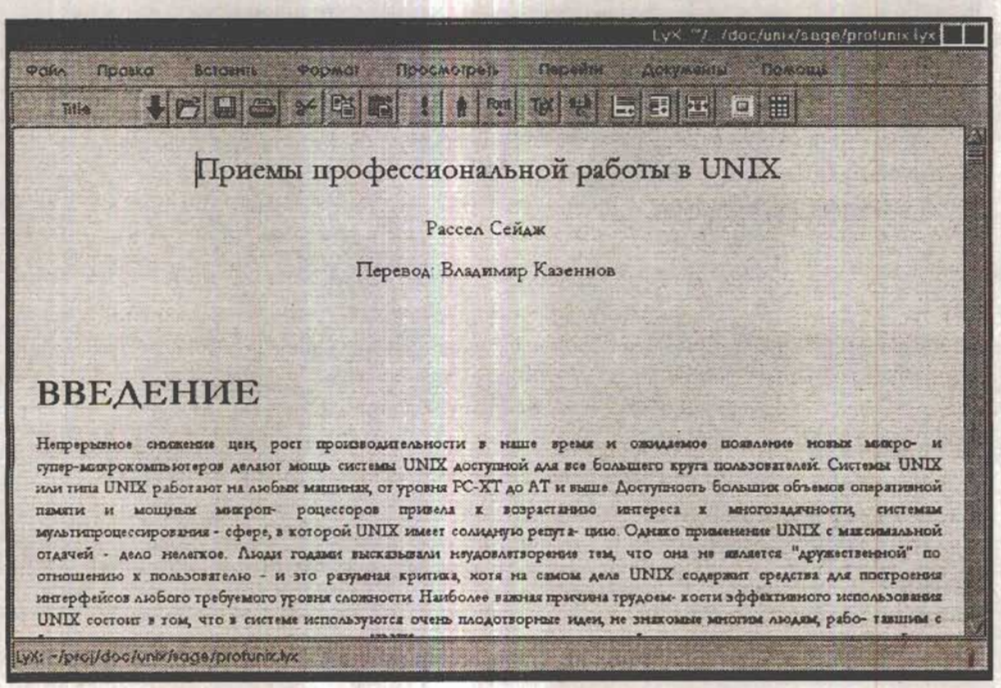


Рис. 22.2. Вид Lyx с загруженным документом

Правда, точная формулировка пунктов меню определяется русскоязычным файлом ресурсов и может варьировать от версии к версии. Кроме того, все это великолепие с русскими меню можно видеть, как я уже говорил, только при доведенной до ума русификации — иначе есть риск наблюдать обычную абракадабру. Хотя Lyx и принадлежит к правильным (почти!) приложениям, берущим сведения о языковом окружении из локали, *настройка шрифтов по умолчанию в нем на таковое отнюдь не рассчитана.* Так что прежде, чем использовать Lyx, придется затратить некоторое время на его конфигурирование.

Вторым важным элементом интерфейса Lyx является инструментальная панель — атрибут не очень обычный для Motif-приложений. Здесь можно видеть, во-первых, выпадающее меню для назначения стилей, во-вторых, кнопки для наиболее частых операций — открытия, сохранения и печати файла, вырезания, копирования и вставки фрагментов, вставки примечаний, отступов, математических выражений, таблиц, рисунков.

В общем, и главное меню, и инструментальная панель Lyx интуитивно понятны, хотя и отличаются от соответствующих стандартных интерфейсных элементов Windows (а также приложений таких сред, как KDE или GNOME). И потому можно немедленно начинать работу — уделив предварительно некоторое время настройкам программы — в первую очередь, на предмет работы с русским языком.

О настройках Lyx писалось немало — однако ныне значительная часть существующих в Интернете рекомендаций (в том числе и вашего покорного слуги) устарела. Взамен появилось руководство Алексея Крюкова (см. дополнительные источники), исчерпывающий характер которого позволяет мне остановиться на этой теме вкратце.

Итак, ныне все настройки Lyx выполняются в интерактивном режиме, командой меню **Правка — Параметры**. Она вызывает конфигурационную панель Lyx (рис. 22.3), в которой можно определить экранные шрифты документов и интерфейсных элементов, язык словаря для проверки орфографии, конверторы экспорта/импорта документов, начальные каталоги для документов и шаблонов, опции печати и т. п.

Результаты действий в конфигурационной панели фиксируются в главном файле настройки — ~/.Lyx/preferences. Это простой текстовый файл, доступный также для прямого редактирования. Делать это разработчиками не рекомендуется — он будет автоматически регенерирован при следующем запуске конфигуратора. Однако на первом этапе без такой правки обойтись не удастся.

Дело в том, что при установленной русской локали в меню впервые запущенного Lyx не удастся увидеть ни одной русской буквы — по умолчанию в меню используются шрифты Adobe, символов кириллицы не содержащие. И если нет должного навыка работы вслепую (т. е. с греко-французской абракадаброй), проще всего выйти из конфигуратора (и из Lyx), открыть в текстовом редакторе упомянутый файл (а он генерируется программой настройки автоматически), отыскать в нем секцию

```
# SCREEN & FONTS SECTION ############################
```

вручную вписать в ее строки значения, соответствующие заведомо кириллическим шрифтам из системы X Window, и определить требуемую кодировку русскоязычных документов. Например, при использовании KOI8 это может выглядеть таким образом:

```
\screen_font_encoding "koi8-r"
\screen_font_roman "-monotype-garamond"
```

```
\screen_font_sans "-monotype-centure gothic"
\screen_font_typewriter "-monotype-courier new"
\screen_font_popup "-monotype-century gothic-medium-r"
\screen_font_menu "-monotype-century gothic-bold-r"
```

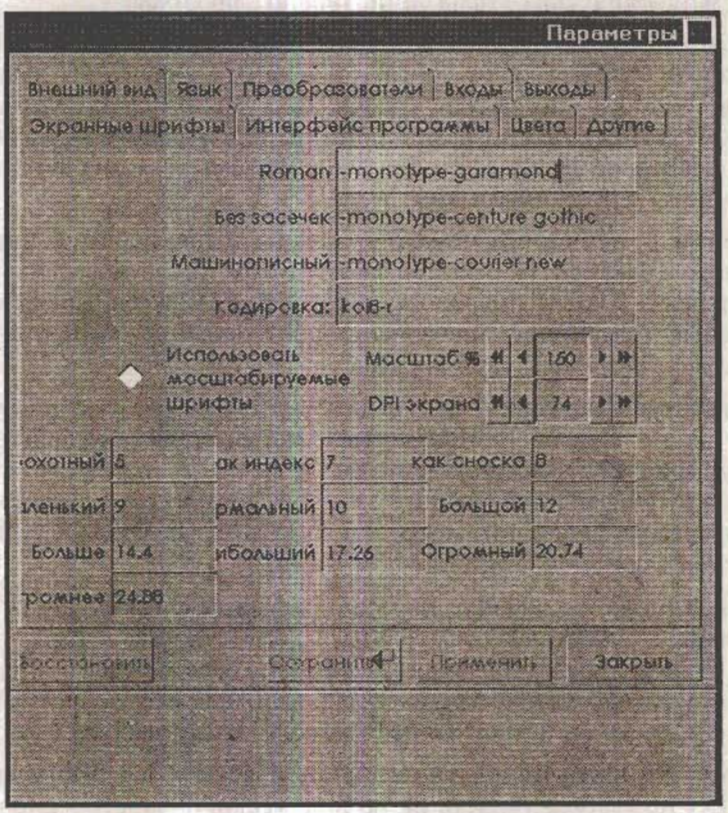


Рис. 22.3. Панель настройки Lyx

Легко догадаться, что первая строка определяет кодировку, три следующие — экранные шрифты для документов, две последние — шрифты элементов интерфейса. И для документа, и для интерфейса могут использоваться растровые шрифты (например, cronyx из стандартного комплекта XFree86), масштабируемые шрифты Type1 или TTF.

После этого следует перезапустить Lyx, убедиться в появлении кириллических букв в меню (а при желании — и в документах) и вернуться к панели настроек для дальнейшего конфигурирования.

Для эффективного использования Lyx следует четко уяснить, что он не является редактором типа WYSIWYG в том смысле, который применяется к

программам типа MS Word. Конечно, некоторое соответствие между экранным представлением документа и его видом на печати есть, но — не более. Поэтому не следует удивляться если вы, например, изменили ориентировку таблицы с горизонтальной на вертикальную (а Lyx, честь ему и хвала, позволяет развернуть как таблицу целиком, так и отдельные ее ячейки и их серии) и не получили никакого видимого эффекта. Полностью представить себе отпечатанный текст можно только при предварительном просмотре сгенерированного PostScript- или DVI-файла. Но зато уж это будет действительно адекватное представление (с учетом шрифтов, настроек принтера и т. п.), практически гарантирующее от неожиданностей в бумажной версии.

Вторая особенность Lyx — жесткая его ориентация на стилевое представление. Каковое ничего общего не имеет с форматированием как созданием визуальных эффектов, а отражает логическую структуру документа. И потому, в отличие от редакторов типа MS Word, стили находятся не в пункте меню **Формат**, а вынесены в всплывающее меню на основной инструментальной панели.

Смысл доступных по умолчанию стилей легко понять любому, хоть раз в жизни создававшему html-страницу непосредственно в кодах. Что и не удивительно, поскольку как HTML, так и TeX (база Lyx) суть не что иное, как языки разметки.

Так вот, среди доступных стилей первым следует стиль **Standart** (Стандартный), приписываемый любому введенному (или импортированному) текстовому блоку по умолчанию. Далее в списке можно видеть заголовок (**Title**), рубрики разного уровня, нумерованные и ненумерованные (**Part** и **Part***, **Section** и **Section*** и подобные соответственно), элементы ненумерованных и нумерованных списков (**Enumerate** и **Itemize** соответственно), **Paragraph** (Абзац) и **Subparagraph** (Подабзац), а также ряд более специфических: **Author** (Автор), **Date** (Дата), **Bibliography** (Библиография), **Abstract** (Резюме) и другие, зависящие от типа документа (**Article** (Статья), **Book** (Книга), **Letter** (Письмо) и т. п.). Тип же документа определяется предварительно командой главного меню **Формат — Документ**.

Эти особенности создают определенный дискомфорт в работе при отсутствии должных навыков. Возникает вопрос, а есть ли у Lyx достоинства, ради которых стоит такие навыки приобретать? Агитировать не буду, судите сами.

О возможностях стилевого оформления и форматирования говорить не стоит: они аналогичны любому современному текстовому процессору, хотя, как можно понять из вышесказанного, и осуществляются несколько иначе. А вот возможности вставки всяких специальных элементов — поистине безграничны. И осуществляются соответствующими командами пункта **Вставить** главного меню.

В Lyx легко и просто создаются оглавления и указатели (предметные, списки иллюстраций, таблиц, библиографические ссылки). Предусмотрены вставки

примечаний, сносок, заметок на полях, перекрестных ссылок, элементов индексных списков. В Lyx, помимо этого, существует понятие вставки кадров, содержанием которых может быть таблица или рисунок. Возможна просто вставка рисунков, правда, в одном-единственном формате — PostScript (ps, eps). В который, впрочем, силами практически любого графического редактора может быть претворено изображение любого иного формата.

Вставка рисунка происходит в два этапа. Сначала командой меню **Вставить — Рисунок** образуется пустая рамка. Щелчок мышью на ней вызывает панель, через которую определяется требуемый графический файл (ps или eps) и параметры рисунка: его ширина и высота (исходные, в сантиметрах, дюймах или процентах от размера страницы), представление (монохромное, в градациях серого, в цвете), разворот (на произвольный угол), заголовок и т. п. Все эти параметры в любой момент могут быть изменены (тем же щелчком мышью на рисунке). Что вполне компенсирует отсутствие возможности интерактивного масштабирования.

Чрезвычайно развиты функции работы с таблицами. По умолчанию командами меню **Вставить — Таблица** создается таблица в пять строк и колонок (это число может быть изменено в тот же момент с помощью ползунков). При этом первая строка автоматически приобретает статус шапки таблицы (**Caption**). Далее, в любой момент может быть добавлено любое количество как строк, так и колонок (правда, только по одной за раз).

Содержимое ячеек таблицы может быть выровнено по левому или по правому краю, а также центрировано (двусторонняя выключка не предусмотрена). В каждой ячейке таблицы, их группе, а также во всей таблице целиком могут быть показаны (по умолчанию) или скрыты ограничивающие рамки с любой из сторон (и со всех сразу). Две и более ячеек могут быть объединены, правда, только по горизонтали — объединение вертикально расположенных ячеек невозможно. Но зато можно изменить ориентацию любой ячейки или их группы. Как, впрочем, и всей таблицы целиком. Последнее делается через специальную панель, вызываемую щелчком правой кнопки мыши на любом элементе таблицы.

К недостаткам Lyx можно отнести ограниченные возможности обмена документами. Напрямую он способен открывать и сохранять только файлы собственного формата (Lyx). Командой **Файл — Импорт** в Lyx можно считать текстовый файл — с разбиением по строкам или по абзацам. Экспорт возможен в форматы LaTeX, DVI, PostScript, ASCII, HTML. Относительно бедности понимаемых графических форматов (только PS и EPS) я уже говорил.

Тем не менее, богатство возможностей по оформлению текстов и простота их использования (после освоения, каковое требует некоторых сил и времени), на мой взгляд, более чем окупают эти недостатки. Особенно если речь идет об объемных и сложно структурированных текстах, например, научного содержания, требующих развитого аппарата в виде оглавлений, примечаний,

ссылок, указателей и прочего, иллюстрированных большим количеством сложных таблиц и штриховых иллюстраций (для полноцветных иллюстраций Lyx — далеко не лучший выбор).

Следует также заметить, что нестандартное оформление текста в Lyx выполняется не очень просто и требует знания англо-американской полиграфической терминологии, не имеющей часто однозначных соответствий в терминологии русскоязычной. Однако для документов стандартного облика использование Lyx чрезвычайно эффективно: вполне приемлемые для полиграфического воспроизведения результаты достигаются практически без участия пользователя, требуя от него только понимания логической структуры документа. Именно ориентированность на логику содержания, но не на визуальное представление — черта, резко отличающая Lyx от типичных текстовых процессоров для Windows (и, рискну добавить, их последователей в среде X Window).

Помимо собственно Lyx, существует еще и аналогичная реализация его для среды KDE — kLyx. Однако она, во-первых, откровенно недоработана, во-вторых, развитие ее, судя по всему, прекратилось. Почему я и не стал на нем останавливаться.

Для полноты картины добавлю, что в коллекции портов FreeBSD можно обнаружить еще один специфичный для UNIX-систем визуальный текстовый процессор, не обнаруживающий воздействия Windows-программ — TeXmacs. Однако ввиду слабого с ним знакомства — от описания воздержусь, тем более, что информацию о нем можно обнаружить в дополнительных источниках.

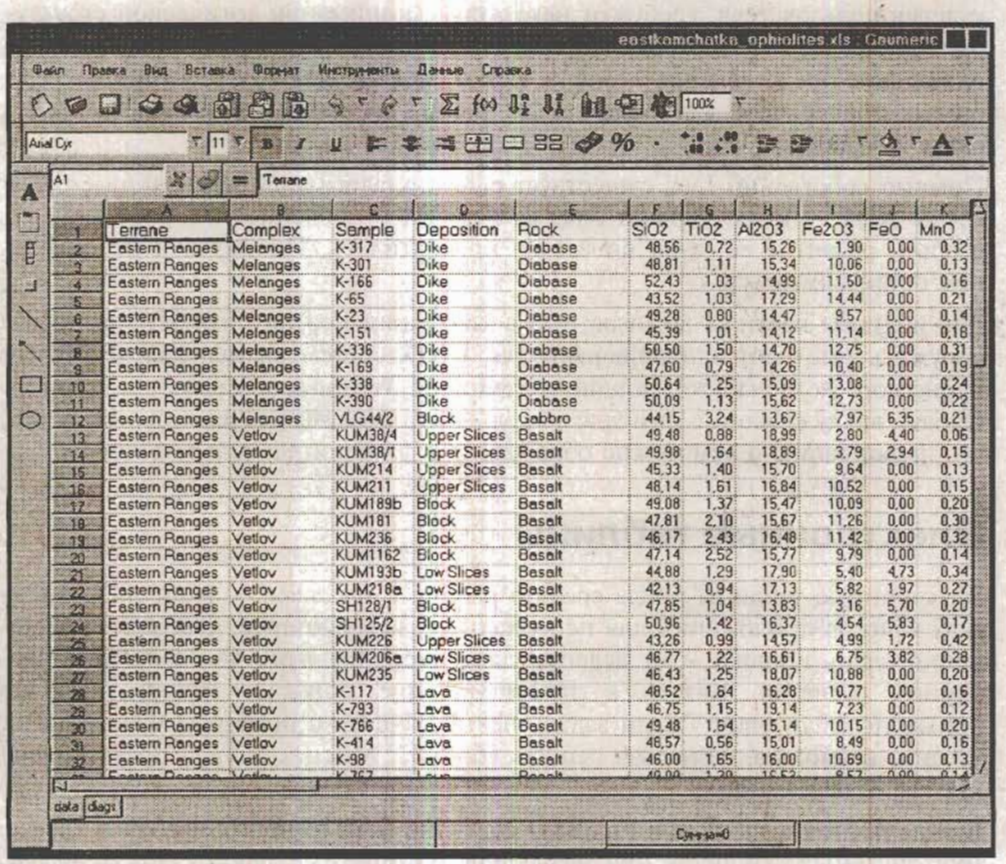
Электронные таблицы

Электронные таблицы после средств работы с текстами — один из важнейших классов приложений для персональных компьютеров. Сфера их применения очень широка — от ведения домашнего бюджета до сложных инженерных расчетов. Как-то в литературе мне попалась фраза, гласящая примерно следующее: "Если задачу нельзя решить с помощью электронных таблиц, возможно, ее нельзя решить вообще". Мнение если и преувеличенное, то — не намного.

В виде портов и пакетов в FreeBSD доступно немало программ этого класса. Для системы X Window предназначены автономные пакеты abs и abacus, табличные процессоры интегрированных пакетов OpenOffice и KOffice — StarCalc и KSpread соответственно, а также Gnumeric, ориентированный на среду GNOME. Все это программы, разные как по задуманной функциональности, так и по степени доработанности. Диапазон здесь — от простенького abs до мощного, но пока проблемного StarCalc и столь же, теоретически, функционального, но пока не вполне работоспособного KSpread. Так что фактически заслуживают рассмотрения только abacus и Gnumeric.

О первом можно обнаружить сведения в сети (см. дополнительные источники), поэтому остановлюсь только на втором.

Внешне это — весьма стандартный представитель своего сословия (рис. 22.4), обладающий Excel-подобным интерфейсом. Он позволяет создавать многостраничные рабочие книги, строить графики и диаграммы, рисовать непосредственно в пределах рабочего листа и т. д. Из продвинутых возможностей следует отметить весьма обильные функции статистического анализа и встроенный Basic-подобный язык (Gnomebasic).



	Terrane	Complex	Sample	Deposition	Rock	SiO2	TiO2	Al2O3	Fe2O3	FeO	MnO
1	Eastern Ranges	Melanges	K-317	Dike	Diabase	48,56	0,72	15,26	1,90	0,00	0,32
2	Eastern Ranges	Melanges	K-301	Dike	Diabase	48,81	1,11	15,34	10,06	0,00	0,13
3	Eastern Ranges	Melanges	K-166	Dike	Diabase	52,43	1,03	14,99	11,50	0,00	0,16
4	Eastern Ranges	Melanges	K-65	Dike	Diabase	43,52	1,03	17,29	14,44	0,00	0,21
5	Eastern Ranges	Melanges	K-23	Dike	Diabase	49,28	0,80	14,47	9,57	0,00	0,14
6	Eastern Ranges	Melanges	K-151	Dike	Diabase	45,39	1,01	14,12	11,14	0,00	0,18
7	Eastern Ranges	Melanges	K-336	Dike	Diabase	50,50	1,50	14,70	12,75	0,00	0,31
8	Eastern Ranges	Melanges	K-169	Dike	Diabase	47,60	0,79	14,26	10,40	0,00	0,19
9	Eastern Ranges	Melanges	K-338	Dike	Diabase	50,64	1,25	15,09	13,08	0,00	0,24
10	Eastern Ranges	Melanges	K-390	Dike	Diabase	50,09	1,13	15,62	12,73	0,00	0,22
11	Eastern Ranges	Melanges	VLG44/2	Block	Gabbro	44,15	3,24	13,67	7,97	6,35	0,21
12	Eastern Ranges	Vetlov	KUM38/4	Upper Slices	Basalt	49,48	0,88	18,99	2,80	4,40	0,06
13	Eastern Ranges	Vetlov	KUM38/1	Upper Slices	Basalt	49,98	1,64	18,69	3,79	2,94	0,15
14	Eastern Ranges	Vetlov	KUM214	Upper Slices	Basalt	45,33	1,40	15,70	9,64	0,00	0,13
15	Eastern Ranges	Vetlov	KUM211	Upper Slices	Basalt	48,14	1,61	16,84	10,52	0,00	0,15
16	Eastern Ranges	Vetlov	KUM189b	Block	Basalt	49,08	1,37	15,47	10,09	0,00	0,20
17	Eastern Ranges	Vetlov	KUM181	Block	Basalt	47,81	2,10	15,67	11,26	0,00	0,30
18	Eastern Ranges	Vetlov	KUM236	Block	Basalt	46,17	2,43	16,48	11,42	0,00	0,32
19	Eastern Ranges	Vetlov	KUM1162	Block	Basalt	47,14	2,52	15,77	9,79	0,00	0,14
20	Eastern Ranges	Vetlov	KUM193b	Low Slices	Basalt	44,88	1,29	17,90	5,40	4,73	0,34
21	Eastern Ranges	Vetlov	KUM218a	Low Slices	Basalt	42,13	0,94	17,13	5,82	1,97	0,27
22	Eastern Ranges	Vetlov	SH128/1	Block	Basalt	47,85	1,04	13,83	3,16	5,70	0,20
23	Eastern Ranges	Vetlov	SH125/2	Block	Basalt	50,96	1,42	16,37	4,54	5,83	0,17
24	Eastern Ranges	Vetlov	KUM226	Upper Slices	Basalt	43,26	0,99	14,57	4,99	1,72	0,42
25	Eastern Ranges	Vetlov	KUM206a	Low Slices	Basalt	46,77	1,22	16,61	6,75	3,82	0,28
26	Eastern Ranges	Vetlov	KUM235	Low Slices	Basalt	46,43	1,25	18,07	10,88	0,00	0,20
27	Eastern Ranges	Vetlov	K-117	Lava	Basalt	48,52	1,64	16,28	10,77	0,00	0,16
28	Eastern Ranges	Vetlov	K-793	Lava	Basalt	46,75	1,15	19,14	7,23	0,00	0,12
29	Eastern Ranges	Vetlov	K-766	Lava	Basalt	49,48	1,54	15,14	10,15	0,00	0,20
30	Eastern Ranges	Vetlov	K-414	Lava	Basalt	46,57	0,56	15,01	8,49	0,00	0,16
31	Eastern Ranges	Vetlov	K-98	Lava	Basalt	46,00	1,65	16,00	10,69	0,00	0,13
32	Eastern Ranges	Vetlov	K-763	Lava	Basalt	46,00	1,39	15,52	8,57	0,00	0,14

Рис. 22.4. Электронная таблица Gnumeric

Пожалуй, главное, что отличает Gnumeric от менее развитых собратьев — *весьма приличная совместимость с MS Excel. Он напрямую считывает файлы* последнего, созданные в версии 97/2000 (и более ранних), в том числе и содержащие символы кириллицы. При этом сохраняются формулы, элемен-

ты форматирования и даже встроенные в рабочие листы графики. Правда, возможность редактировать их утрачивается.

Кроме файлов Excel, Gnumeric воспринимает (посредством команды меню **Файл — Импорт**) также текстовые базы данных с самыми различными разделителями полей, таблицы формата XBase, Lotus-123 старых версий (файлы WK1) и еще некоторые. Сохранение данных, кроме "родного" формата (Gnumeric XML), возможно также в формате Excel (правда, версии 6/95), TeX, HTML, текстовом и еще некоторых, менее известных.

Графика и мультимедиа

Коллекция портов FreeBSD включает в себя набор графических и мультимедийных приложений, предназначенных для работы в системе X Window. Первые — это, с одной стороны, полнофункциональные средства для работы с растровой и, в меньшей степени, векторной графикой, с другой — отдельные утилиты для выполнения частных задач ограниченного масштаба.

Из мультимедийных же приложений я остановлюсь на тех, что ориентированы в основном на любительские применения, связанные с воспроизведением аудио- и видеоинформации различных форматов. Это не значит, что в FreeBSD невозможна профессиональная работа с мультимедиа-данными — просто средства для этого выходят за рамки моей компетенции.

Среди программ для работы с графикой на первое место следует поставить GIMP — универсальный полнофункциональный редактор растровой графики, единогласно признаваемый одним из самых удачных проектов, разрабатываемых в рамках модели открытых исходных текстов.

При запуске GIMP загружается его главная управляющая панель (рис. 22.5) и несколько дополнительных инструментальных панелей.

Управляющая панель GIMP включает строку меню, серию кнопок для выполнения различных действий, поля текущих цветов переднего плана/фона и текущего инструмента (т. н. действующей кисти).

Главное меню содержит три пункта — **Файл**, **Расширения (Xtns)**, **Справка**. В первом пункте сосредоточены базовые операции:

- создание файла (**Новый**);
- открытие файла (**Открыть**);
- захват изображения с внешнего устройства, в том числе со сканера (**XSane**), и с экрана;
- **Настройки**;
- **Диалоги**;
- **Выход**.

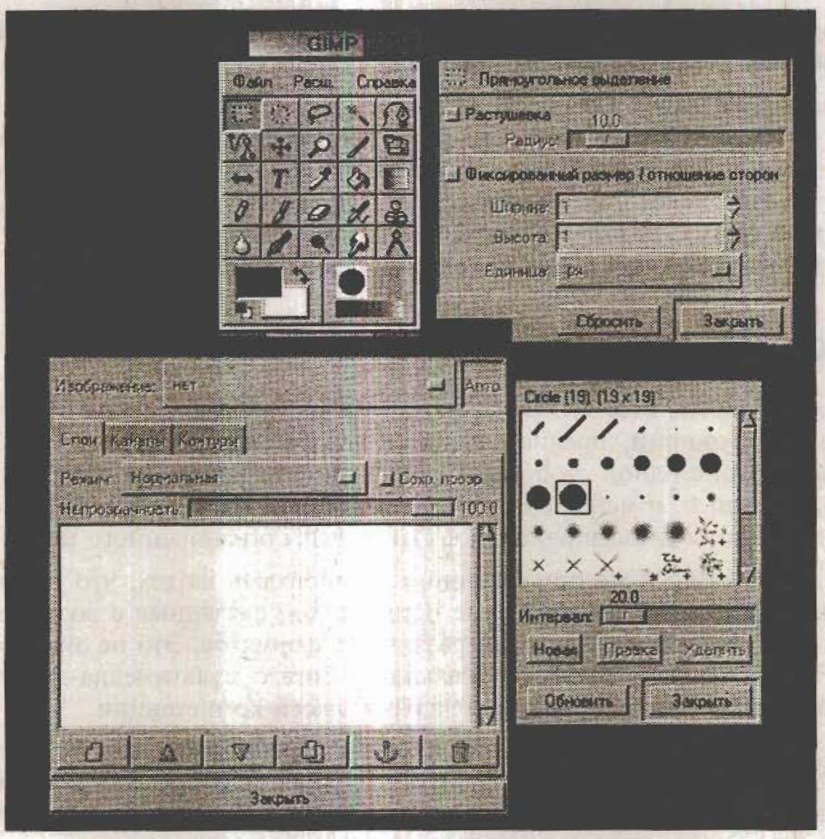


Рис. 22.5. Главная управляющая панель GIMP и дополнительные инструментальные

С помощью меню **Расширения (Xtns)** открывается доступ к следующим возможностям:

- просмотр модулей;
- детали дополнения;
- просмотр базы данных дополнительных модулей;
- редактор единиц измерения;
- разложение видео на кадры;
- доступ к пользовательским модулям (**Script-Fu**).

Здесь же сосредоточены ссылки на Web-ресурсы по GIMP и средства программирования модулей на Perl.

Однако меню управляющей панели играет ограниченную роль в программе. Все основные манипуляции с изображением совершаются (после открытия

или создания файла) щелчком правой кнопки мыши на нем. Этим вызывается всплывающее меню, содержащее многочисленные пункты:

- **Файл** (File);
- **Правка** (Edit);
- **Выделение** (Select);
- **Просмотр** (View);
- **Изображение** (Image);
- **Слои** (Layers);
- **Инструменты** (Tools);
- **Диалоги** (Dialogs);
- **Фильтры** (Filters);
- **Guides**;
- **Видео** (Video);
- **Script-Fu**.

Назначение первого пункта — манипуляции с файлами, а также общие программные настройки и печать. Следует отметить, что GIMP имеет собственный оригинальный формат файла — XCF. Именно в нем по умолчанию создается новый файл, для чего вызывается диалоговое окно, в котором определяются:

- размер изображения (высота и ширина в пикселах и дюймах); если в буфере присутствует скопированное изображение, создаваемый файл будет автоматически подстроен под его размер;
- горизонтальное и вертикальное разрешение (по умолчанию — 72 точки на дюйм);
- палитра создаваемого изображения — в цветах RGB или градациях серого;
- тип заливки.

Кроме этого, GIMP позволяет напрямую открывать и редактировать файлы практически в любом распространенном растровом формате — TIFF, GIF, JPEG, PNG, BMP, включая формат Adobe PhotoShop, AVI, EPS, WMF и многие другие. И по умолчанию (командой **Save**) сохраняет измененный файл в том же формате, в котором он был загружен.

Функция экспорта изображения осуществляется командой меню **Файл — Сохранить как**. Список экспортируемых форматов столь же широк, как и набор импортируемых типов файлов. При экспорте учитывается характер изображения: файл в индексированных цветах (например, GIF) нельзя напрямую сохранить в полноцветном виде (как TIF или JPEG). И — наоборот, для сохранения tif-файла в формате GIF необходимо предварительно выполнить индексацию.

Пункт **Правка** в основном аналогичен таковому стандартных Windows-программ: вырезание, копирование и вставка, **Undo** и **Redo** (число их уровней по умолчанию — 5). Возможно одновременное копирование нескольких изображений, для чего предназначены именованные буфера. Смысл остальных подпунктов обычно понятен из их названий (заливка, обводка и т. д.).

В пункте **Выделение** — всякого рода выделения: всего изображения, цветовых областей, плавающее, инвертирование выделения и запись выделенной области в виде канала.

В пункте **Просмотр** — изменение масштаба просмотра изображения: с шагом, кратным двум, точка за точкой или с помощью дополнительного навигационного окна, скрытие/показ линеек, направляющих и привязка к последним, а также открытие нового окна (с текущим файлом в нем).

Основные манипуляции с рисунком сосредоточены в пункте **Изображение**. Здесь его можно преобразовать (в RGB, индексированные цвета или градации серого), трансформировать, вращать, обрезать и т. д. Здесь же — работа с альфа-каналами и цветовыми палитрами. С помощью подпунктов **Размер холста** и **Масштабирование** — осуществляется обрезание рисунка до заданного размера и изменение его размера соответственно.

Смысл пункта **Слои** ясен из названия. В пункте **Инструменты** — множество подпунктов; выделение областей различной формы (многоугольников, полигонов, произвольной формы и т. д.), перемещение, обрезание, зеркалирование, вращение (что частично совпадает с некоторыми подпунктами из пункта **Изображение**), и т. п. Здесь же — средства работы с текстом и средства рисования.

Следует заметить, что практически все действия пункта **Инструменты** доступны и через инструментальные кнопки главной управляющей панели.

Пункт **Диалоги** аналогичен таковому меню главной панели. Это вызов всякого рода дополнительных панелей: слоев, каналов, кистей, палитр, шаблонов, градиентных заливок и т. д.

Следующий пункт — **Фильтры** — обеспечивает наложение разнообразных спецэффектов. Иногда ряд пунктов неактивен. Обычно это значит, что данный фильтр просто применим к типу открытого файла. Как, например, гауссово размытие по определению нельзя применить к файлу формата GIF.

Самое интересное в GIMP — пункт **Script-Fu**, не имеющий аналогов в других растровых редакторах. Это — набор скриптов для создания пользовательских спецэффектов. Что в целом близко к назначению фильтров, но отличается тем, что скрипты эти — набор интерпретируемых команд на языках (Tcl, Scheme и Perl), посредством которых можно создавать какие-нибудь уникальные эффекты. Кроме того, можно однократно запрограммировать какой-либо эффект и затем распространить его на различные исходные изображения. В отличие от применения фильтров, действие скриптов отменить (командой **Undo**) нельзя.

Кроме этого, в GIMP имеются функции для работы с видеопоследовательностями. Так, файл AVI (правда, только некомпрессированный) может быть разложен на составляющие его кадры, для которых можно изменить последовательность, применить сдвиг, удалить отдельные и т. п. Возможна и обратная процедура — сборка видеопоследовательности из отдельных изображений.

Возможности, предоставляемые GIMP, часто оказываются излишними для простых манипуляций с изображениями (масштабирования, коррекции яр-

кости/контрастности, преобразования из формата в формат). На этот случай в FreeBSD включен ряд более простых утилит для работы с растровой графикой.

К таким средствам относится пакет для просмотра серии изображений — программа XV. Она обеспечивает одновременное открытие многих файлов (например, всех изображений из каталога) и навигацию по ним, в том числе и организацию слайд-шоу. Кроме того, в нее входят средства несложного редактирования изображений и конвертации графических форматов.

Для каталогизации и просмотра графических файлов предназначена программа GQview. Она не содержит встроенных средств редактирования изображений, но допускает подключение внешних редакторов в большом количестве (по умолчанию это обычно GIMP, XV и некоторые другие).

Наконец, в составе графической среды GNOME в качестве штатного средства имеется программа gPhoto. Изначально она предназначена для работы с цифровыми фотокамерами (список поддерживаемых моделей весьма обширен). Однако в ней есть довольно удобные средства каталогизации графических файлов. Возможно также несложное редактирование изображений — изменение размера, яркости, контрастности, цветового баланса, вращение и отражение. Наконец, средства экспорта в формат HTML позволяют легко создавать галереи изображений для представления в Интернете.

Из специализированных программ для захвата изображений следует отметить Ksnapshot, входящую в среду KDE и предназначенную для снятия экранных копий (так называемых скриншотов). Она несколько напоминает Capture из комплекта CorelDRAW, хотя и победнее возможностями.

Программа Ksnapshot позволяет снять весь экран или только активное окно, скрыв или, напротив, выведя на первый план окно самого Ksnapshot; в отличие от Capture, нельзя скопировать объект произвольной формы. Результаты можно сохранить в виде GIF, JPEG, BMP и еще нескольких форматах, среди которых, правда, отсутствует TIFF.

К сожалению, векторная графика не является сильной стороной UNIX-платформ: для них не существует ничего функционально близкого современным векторным графическим редакторам для Windows и Mac OS. Исключения — рисовальный модуль OpenDraw из пакета OpenOffice, который по своим изобразительным возможностям примерно соответствует CorelDRAW 5-й или 6-й версий. И при этом обладает некоторыми уникальными особенностями, например, возможностью создания истинно трехмерных (OpenGL) изображений.

Существуют и средства проигрывания mpeg-файлов для графического режима. Одним из лучших считается программа xmms (X MultiMedia System). Как по интерфейсу, так и по возможностям она воспроизводит известный mpeg-проигрыватель для Windows — WinAmp. Эта программа позволяет от-

сосуществования. Правда, все сказанное далее о Windows относится только

крывать сразу все (или только выбранные) файлы каталога и, соответствен-

к семейству 9x/ME — опыта, касаемого NT/2000/XP, у меня нет, и за соответствующим сведениями могу только отослать к дополнительным источникам информации.

Распределение дискового пространства

Продуманная политика в отношении разбиения диска — необходимое условие успешного использования FreeBSD с любой иной ОС, особенно если таковых предполагается не одна. Я уже затрагивал этот вопрос в *главе 2 (разд. "Представление о файловых системах")* и *главе 4*, однако следует повторить основные положения.

Прежде всего, очевидно, что, поскольку количество первичных разделов на ATA-диске ограничено четырьмя, на одном физическом диске не может сосуществовать более четырех же ОС (о некоторых исключениях я скажу в следующем разделе).

Далее, некоторые ОС предъявляют свои требования к дисковым разделам для своих файловых систем. Так, Windows 9x/ME, как и DOS, сама по себе категорически не способна загрузиться с какого-либо раздела, кроме первичного на первом же физическом диске — т. е., в понимании FreeBSD, на слайсе ad0s1 (в случае ATA-диска). Правда, некоторые менеджеры начальной загрузки позволяют обойти это ограничение, но, насколько мне известно, все они — коммерческие.

Что же касается Linux, то эта ОС (вернее, BIOS машины и LILO — штатный загрузчик многих дистрибутивов) в некоторых своих реализациях требует, чтобы ядро ее целиком находилось не далее 1023 цилиндра диска. А корневая файловая система, на коей ядро локализовано, обязана подчас занимать первичный дисковый раздел (то бишь отдельный слайс), но никак не том в расширенном разделе.

Третий момент — инсталляторы многих т. н. user-ориентированных дистрибутивов Linux (не говоря уж о программе установки любой Windows в автоматическом режиме) часто маскируют от пользователя процесс разбиения диска на разделы и его логику. Конечно, "умолчальные" их схемы разбиения обычно (по крайней мере — в дистрибутивах, заслуживающих этого названия) разумны. Однако они ориентированы либо на Linux как единственную систему, либо на его сосуществование с Windows, но уж никак не с FreeBSD.

И потому при первичной подготовке диска лучше пользоваться программой sysinstall, предоставляющей пользователю почти полный контроль за ходом процесса, или программой fdisk из FreeBSD же. Впрочем, одноименная программа из Linux, или такая Linux-утилита, как cfdisk, предоставляют не меньше возможностей, и если они лучше знакомы (или просто привычней) — так и флаг им в руки.

Наконец, последнее. В некоторых случаях существенным оказывается порядок, в котором создаются дисковые разделы и устанавливаются ОС. Не секрет, что DOS/Windows 9*x*/ME при своей установке имеет обыкновение переписывать главную загрузочную запись (MBR) по своему разумению, после чего загрузка любой другой системы оказывается невозможной. Само по себе это не смертельно (и легко лечится), однако во избежание лишних телодвижений лучше устанавливать Windows раньше любой из иных систем.

Общих рекомендаций по разбиению диска при использовании FreeBSD и более чем одной иной ОС дать невозможно. Единственно, что однозначно необходимо — внимательное знакомство с требованиями к подготовке дисков для установки всех предполагаемых к использованию систем. Особенно это касается Linux — изобилие его дистрибутивов затрудняет выработку каких-либо рецептов. Разумеется, следует знать и требования FreeBSD — однако, надеюсь, с ними вы ознакомились в этой книге. И, кроме того, именно FreeBSD имеет минимальное количество ограничений. Единственным из которых, пожалуй, является естественное желание работать в своем собственном персональном разделе (слайсе).

Теперь рассмотрим конкретные примеры. Наиболее типичными ситуациями совместного использования FreeBSD и каких-либо других ОС мне видятся следующие:

- у пользователя Windows проявилось желание ознакомиться с FreeBSD, но к переходу на нее он морально не готов;
- аналогичное желание возникло в ходе совместного использования Windows и Linux;
- потребность в приобщении к FreeBSD наметилась у "записного линуксоида".

В любом случае порядок действий будет зависеть от того, требуется ли сохранить существующую систему (системы) или есть возможность начать дело с "чистого винта". Второй случай, разумеется, много проще, потому с него и начнем.

Самым простым, разумеется, будет случай совместной установки Linux и FreeBSD. В этом случае средствами любой из этих систем следует создать дисковые разделы и установить системы в любом порядке, обращая внимание только на местонахождение их начальных загрузчиков: тот из них, который предполагается в качестве мультисистемного boot-менеджера, должен быть записан в MBR, второй же — в загрузочный сектор корневого раздела "родной" ОС. Куда следует поместить и оба системных загрузчика, если предполагается использование внешнего менеджера загрузки (о чем подробнее — в следующем разделе).

В данном случае более-менее универсальной будет такая схема разбиения диска:

- небольшой (какой именно — следует посмотреть в рекомендациях к конкретному дистрибутиву) первичный раздел (слайс, если разбиение выполняется средствами FreeBSD) под корневую файловую систему Linux;

- слайс для разделов (partitions) FreeBSD размера, требуемого для этой ОС и ее данных;
- слайс (или расширенный раздел, если разбиение выполняется средствами Linux) размера, требуемого для создания таких файловых систем этой ОС, как /usr и /home, а также ее раздела подкачки.

Можно видеть, что при этом в запасе у нас остается возможность создания еще одного первичного раздела (слайса). Его можно зарезервировать для какой-либо иной ОС, например, OpenBSD. Или даже другого дистрибутива Linux — но только с одной-единственной файловой системой, ведь возможность создания расширенного раздела с логическими томами уже задействована.

А можно попытаться несколько сэкономить дисковое пространство, причем — двумя способами. Во-первых, можно (сразу за корневым разделом Linux) создать слайс для совместного использования раздела подкачки как этой ОС, так и FreeBSD — никаких препятствий к тому не возникает. Размер его должен быть равен удвоенному объему ОЗУ — это также общая рекомендация и для FreeBSD, и для современных дистрибутивов Linux с ядром 2.4.

Впрочем, экономия в 1 Гбайт (а больше 512 Мбайт ОЗУ бывает редко) при нынешних объемах дисков выглядит грошовой. Более радикальный способ — попытаться создать слайс для совместного использования этими ОС под файловую систему с данными (/home). Правда, тут возникает вопрос о том, какая именно файловая система (в физическом смысле) должна на этом слайсе располагаться. В принципе Linux и FreeBSD способны (при перекомпиляции ядра с соответствующими опциями) монтировать и читать файловые системы друг друга, хотя и с некоторыми ограничениями — это будет предметом подробного рассмотрения в одном из следующих разделов. Пока скажу только, что регулярно использую раздел ext2fs для обмена данными между FreeBSD и Linux, и ни с какими проблемами не столкнулся. Впрочем, рекомендовать это не рискну, оставляя его на страх и риск энтузиастов лозунга "экономика должна быть экономной".

Не намного сложнее и совместная установка FreeBSD с Windows. Для этого достаточно создать первичный раздел для последней (любым способом, хоть программой FDISK MS DOS), при необходимости — также и расширенный раздел с логическими томами, установить Windows, а на оставшееся пространство инсталлировать FreeBSD в собственный слайс обычным способом (не забыв только записать ее загрузчик в MBR). Возможность создания четвертого слайса можно или просто проигнорировать, или зарезервировать под какие-либо грядущие задачи.

Совместная *установка на один физический диск и Windows, и Linux, и* FreeBSD, во-первых, сложнее, во-вторых — наиболее многовариантна. Перво-наперво, каждая из них потребует собственного слайса (или раздела, если разбиение будет осуществляться средствами Linux). Во-вторых, при созда-

нии их потребуется учитывать ограничения (непременно) Windows и (возможно) Linux. В-третьих, необходимо будет сделать выбор — какой из двух систем (Window или Linux) отдать логические тома расширенного раздела (благо, FreeBSD в таковых не нуждается). И уж почти наверняка для управления образовавшимся "зоопарком" понадобится какой-либо из развитых менеджеров начальной загрузки.

Теперь — о вариантах с сохранением существующей ОС (и ее данных) при полностью занятом диске. О необходимости резервного копирования всей информации полагаемой критически важной, — напоминать не буду, это подразумевается по умолчанию. А далее — тут самым простым видится установка FreeBSD на диск с Windows. Для начала — дефрагментация диска (средствами любых соответствующих утилит), затем — высвобождение дискового пространства программой FIPS (желательно — из "чистого" DOS. И напоследок — установка FreeBSD на слайс, созданный из освободившегося объема, обычным, ранее описанным *(см. главу 4)* способом.

С FreeBSD поверх Linux — несколько сложнее. На изменение размера разделов с файловой системой ext2fs теоретически способны, насколько мне известно, программы установки дистрибутивов Mandrake, Altlinux и ASPLinux, а также коммерческий менеджер дисков (он же — менеджер загрузки) Acronis. Однако в любом случае степеней свободы много меньше. Так, если файловые системы Linux (такие как /, /usr, /home, а также swap-раздел) устанавливались в первичные разделы диска, то для установки FreeBSD потребуется пожертвовать одним из них.

Еще более тяжкий случай — наличие на полностью занятом диске и Linux, и Windows. Теоретически рассуждая, здесь можно предложить несколько схем действия. Пара-тройка из них опробована мной и практически, однако повторять свой опыт — и собаке классового врага не пожелаю. И в любом случае для той же пресловутой критически важной информации потребуется резервный носитель. А при наличии оного — поверьте, проще (и быстрее) переразбить диск и переустановить все системы заново.

Совместный запуск FreeBSD и других систем

Решив тем или иным способом проблему распределения диска и установки ОС, следует обеспечить возможность их загрузки по выбору пользователя. В элементарном виде вопрос этот решается до неприличия просто — с помощью штатного загрузчика FreeBSD — BTX loader, описанного в *главе 7. Он способен управлять загрузкой* почти любых ОС — иных BSD-систем (OpenBSD или, например, NetBSD), Linux, и, конечно же Windows (как ветки 9x/ME, так и NT/2000/XP). Возможность загрузки OS/2 также предусмотрена, хотя вряд ли это актуально в наши дни.

Собственно говоря, загрузчик FreeBSD процессом загрузки чуждых ОС почти не управляет — основная сфера его компетенции начинается после выбора для загрузки своей родной системы. А до этого он просто находит на дисках первичные разделы (они же — слайсы), в соответствующих секторах которых записаны какие-либо иные загрузчики, и предлагает передать им управление. Что делается (безальтернативно) нажатием функциональных клавиш <F1>—<F4> (в случае одного физического диска) или <F1>-<F8> (если таковых в системе два). Подозреваю, что он способен передать управление и на загрузочные сектора разделов большего количества дисков, но на практике проверить это мне пока не доводилось.

Номер функциональной клавиши соответствует номеру первичного раздела диска, а не порядку установки различных ОС. Так, если FreeBSD установлена на первом слайсе, а Linux — на третьем, то системы эти будут грузиться по нажатии клавиш <F1> и <F3> соответственно, даже если второй слайс не содержит никакой ОС или даже не создавался вообще.

К слову сказать, BTX loader будет предлагать (нажатием клавиши <F5>) загрузку со второго физического диска в любом случае, буде таковой существует в системе. Даже если он и размечен и сконфигурирован как слайс FreeBSD же и монтируется как ее файловая система (например, /home). Из этого отнюдь не следует, разумеется, что в данном случае со второго диска можно действительно загрузиться — ответом на такую попытку будет сообщение об "инвалидности" раздела (Invalid Partition), что потребует перезагрузки.

Однако из такого эксперимента можно понять, что на сам по себе процесс загрузки другой ОС BTX loader никакого влияния не оказывает. Ему глубоко безразлично, что находится в загрузочном секторе диска или первичного раздела (и находится ли там что-нибудь вообще) — достаточно факта наличия такового. И потому успех или неудача загрузки сторонних ОС определяется исключительно тем, насколько корректно установлены и сконфигурированы их "родные" загрузчики.

Впрочем, такая простота загрузчика FreeBSD имеет свои положительные стороны. Так, он в принципе не конфигурируем как внешний менеджер, однако в этом и не нуждается. При установке на диск новых ОС или добавлении второго физического диска с любой операционкой он автоматически (при перезагрузке машины) отыскивает их загрузочные записи и вносит соответствующие альтернативы в свое меню. В общем, в большинстве случаев его возможностей вполне достаточно для управления загрузкой "трех весельчаков" — Windows, Linux и FreeBSD.

Существенно бóльшими возможностями настройки обладает LILO (LInux LOader) — традиционный загрузчик Linux, принятый в качестве такового по умолчанию в большинстве дистрибутивов этой ОС. Однако это имеет оборотную сторону. Так, LILO способен управлять загрузкой Linux (еще бы!),

Windows, Free- и OpenBSD, но при этом лишь загрузка первой и второй из этих систем будет сконфигурирована автоматически — для работы с FreeBSD потребуются некоторые манипуляции.

Чтобы выступать в качестве мультисистемного менеджера загрузки, LILO при инсталляции Linux должен быть записан в MBR первого физического диска (а загрузчик FreeBSD, соответственно, — в загрузочный сектор BSD-слайса). При этом в файловой системе Linux будет создан файл /etc/lilo.config — простой текст, описывающий операционные системы и порядок их загрузки.

На деталях настройки LILO я останавливаться на буду — этот процесс хорошо документирован и многократно описывался практически в любой книге по Linux. Скажу только, как обеспечить загрузку FreeBSD его посредством.

Начальная секция файла /etc/lilo.config при установке этого загрузчика как мультисистемного выглядит примерно следующим образом:

```
boot=/dev/hda
install=/boot/boot.b
map=/boot/map
delay=##
```

Первая строка определяет главное загрузочное устройство — первый физический ATA-диск системы (в номенклатуре Linux), вторая — указывает на файл, содержащий загрузочный сектор, третья — файл т. н. карты отображения (названия этих файлов в разных дистрибутивах могут меняться. Последняя строка — время (в миллисекундах) задержки для выбора загружаемой ОС. Если таковой сделан не был, начинается загрузка системы по умолчанию, т. е. описанной в первой из последующих секций или отмеченной как default.

Кроме этого, в начальной секции обычно определяются более или менее обязательные параметры загрузки (видеорежим, имя файла сообщений и др.). Впрочем, для загрузки FreeBSD разбираться с этим нет необходимости — никаких изменений в начальную секцию вносить не потребуется.

В следующей секции, как правило, определяются условия загрузки Linux:

```
image=/boot/vmlinuz
root=/dev/hda#
label=Linux
```

что указывает, *соответственно, на файл* образа ядра Linux, имя корневого раздела для этой ОС (# — его номер), название пункта меню LILO. Очевидно, что и здесь менять ничего не нужно.

После этого могут быть определены условия загрузки Windows, буде таковая была ранее установлена:

```
other=/dev/hda1
table=/dev/hda
label=Windows
```

В этой секции определяются: раздел для Windows (как уже говорилось, почти обязательно — первый), местонахождение таблицы первичных разделов диска и метка для меню LILO. И эта секция в изменениях не нуждается.

А вот теперь пора прописать и нашу FreeBSD — почти так же как это сделано для Windows:

```
other=/dev/hda#
table=/dev/hda
label=FreeBSD
```

Здесь `/dev/hda#` — номер слайса FreeBSD, опять же в номенклатуре Linux. Теперь при перезагрузке машины в меню LILO появится соответствующий пункт, выбрав который, можно загрузить FreeBSD. Если потребуется сделать ее ОС по умолчанию, в начальную секцию следует добавить строку:

```
default=FreeBSD
```

значение которой должно совпадать с значением строки label. Разумеется, в общем случае оно может быть любым — freebsd, fbsd и т. п.

Однако и LILO не свободен от ряда ограничений. Которые определяются тем, что он, как и BTX Loader, реализует, по выражению Владимира Попова (см. дополнительные источники), "тупой", или "аппаратный" способ загрузки по цепи: сначала блоков диска в целом, затем — активного раздела. В противоположность этому, развитые менеджеры загрузки более интеллектуальны, располагая возможность доступа к файловым системам (и, соответственно, конфигурационным файлам) до загрузки ОС, к которым они (файловые системы, сиречь) относятся.

Статус универсального менеджера загрузки в мире свободного программного обеспечения постепенно приобретает GRUB, поскольку он более чем заслуживает своего имени (GRand Unified Bootloader). Он основан на спецификации Multiboot Specification, обеспечивающей универсальный способ загрузки ядра любой ОС. Естественно, из понятия "любой" в данном случае следует исключить ядра систем Самой Великой Компании. Но практически все открытые и свободные ОС (Linux, Free-, Open- и NetBSD, даже недоношенный Hurd) этой спецификации соответствуют и потому загружаются посредством GRUB без малейших проблем. Что же касается ОС, спецификации не соответствующих, то они также могут быть им загружены — по "цепочечному" механизму (так сказать, в соответствии со своей внутренней сущностью).

Менеджер загрузки GRUB включен в коллекцию портов FreeBSD, где его можно обнаружить в каталоге /usr/ports/sysutils/grub, и обычно доступен также в качестве бинарного пакета. Следует только следить, чтобы там имела место достаточно свежая (0.9x и выше) его версия — в ряде версий 4-й ветви FreeBSD включен пакет GRUB версий 0.5.xx, содержащий достаточно серьезные ошибки, приводившие подчас к печальным последствиям.

Свежие, надежные и устойчивые версии GRUB входят как основной или альтернативный загрузчик в состав большинства современных дистрибутивов Linux. Так что проблем с его получением и установкой не возникает. На худой конец его можно скачать с сайта проекта (**http://www.gnu.org/software/grub**) и установить самостоятельно — детальное описание процедуры имеется в сопровождающей документации (см. также дополнительные источники).

Совместный доступ к данным

Следующее, чем следует заняться после установки желаемых ОС и обеспечения их совместной загрузки — это организовать доступ к данным, созданным в одной ОС, из другой. Для связки FreeBSD и DOS/Windows это решается в одностороннем порядке, но не представляет ни малейших сложностей.

Файловые системы FAT16/VFAT/FAT32 поддерживаются ядром FreeBSD, устанавливаемым по умолчанию (ядро GENERIC), и на чтение, и на запись. И, соответственно, под ее управлением Windows-раздел диска может быть смонтирован в файловую систему FreeBSD командой

```
mount_msdos /dev/ad0s1 /mount_point
```

Не возбраняется и монтирование логических томов на расширенном разделе. Чтобы узнать требующиеся для этого точные имена файлов соответствующих устройств, следует дать команду

```
disklabel /dev/ad0s#
```

После монтирования любые данные с Windows-разделов могут быть прочитаны или записаны. При последней процедуре, разумеется, будут действовать "природные" ограничения FAT-систем, которые приведут к утрате информации об атрибутах доступах и, частично, временных атрибутов файлов и каталогов. Не могут быть скопированы на FAT-раздел любого рода также символические ссылки и прочие специальные файлы. И, конечно же, имена файлов, копируемых на раздел с файловой системой FAT16, будут усечены до DOS-стандарта 8.3.

При обратном копировании с FAT-раздела на раздел FFS главное неудобство — присвоение атрибута исполнения всем копируемым файлам. Само по себе это не очень страшно, но не позволяет просматривать определенные типы данных в файловых менеджерах типа deco простым нажатием клавиши

<Enter>. К счастью, существуют не очень сложные способы борьбы с таким явлением (один из них был описан в *главе 14* (раздел об утилите find).

Вообще-то для взаимодействия с носителями, отформатированными в DOS/Windows, предназначена программа mtools, реализованная для всех UNIX-подобных систем и портированная под FreeBSD. Основу ее составляет серия команд гибридного вида и синтаксиса. Так, команды `cp` и `copy` (для FreeBSD и DOS соответственно) подменяются командой `mcopy`, используемой, для примера, таким образом

```
mcopy $HOME/file a:file
```

где `file` — имя копируемого файла, причем обязательно в стандарте DOS (т. е. 8.3). Работает эта команда с одиночными файлами. Кроме того, существуют команды `mread` (для копирования файлов DOS в FreeBSD) и `mwrite` (для копирования файлов FreeBSD в DOS). Обе они позволяют оперировать группами файлов, но не подкаталогами.

Доступ из FreeBSD к разделам с файловой системой NTFS (как в NT-, так и в 2000-й вариации) возможен только в режиме чтения. Однако по умолчанию эта файловая система ядром не поддерживается — требуется перекомпиляция ядра с включением опции

```
options   NTFS          #NT File System
```

которую можно обнаружить в секции `# FILESYSTEM OPTIONS` файла /usr/src/sys/i386/conf/LINT. Однако запись на NTFS-разделы в текущих версиях FreeBSD невозможна. И, вследствие закрытости спецификации этой файловой системы, ожидать ее полной поддержки не приходится.

Невозможен также и доступ к FFS-разделам из Windows. Для последней системы нет даже таких ограниченных средств, как ltools или explore2fs, худо-бедно допускающих обмен данными с Linux.

А вот обмениваться данными между FreeBSD и Linux можно в любом направлении. Однако и тут есть некоторые ограничения. В Linux нормальное восприятие файловой системы FreeBSD, кроме того, что требует перекомпиляции ядра, гарантируется только в режиме чтения — поддержка записи на разделы FFS имеет статус экспериментальной (соответствующая опция в ядрах Linux версий 2.4.xx сопровождается весьма устрашающим комментарием). Собственного опыта в этом отношении у меня также нет, и потому гарантировать отсутствие проблем в данном случае не берусь. Но буду признателен за любые комментарии по этому вопросу.

Для FreeBSD поддержка файловых систем Linux гарантирована только для ext2fs. В ядре FreeBSD данная опция по умолчанию в ядре GENERIC выключена. Включение ее требует строки вида

```
options   EXT2FS
```

которая также помечена (причем — на протяжении длительного времени) как экспериментальная. Однако я регулярно пользуюсь разделом ext2fs (правда, на отдельном физическом диске) для обмена данными между этими системами — как в режиме чтения из FreeBSD, так и для записи. И ни с какими проблемами до сего дня не сталкивался.

Разумеется, и Linux-раздел перед использованием следует смонтировать, воспользовавшись обозначением файла соответствующего устройства в номенклатуре (разумеется!) FreeBSD:

```
mount_ext2 /dev/ad0s#? /mount_point
```

где `#` — номер слайса, на котором располагается требуемая файловая система, а `?` — соответствующая литера. И здесь вся необходимая информация может быть получена посредством команды `disklabel`.

Исходя из своего опыта, не вижу никаких причин, препятствующих созданию общего раздела для данных (/home) при совместном использовании FreeBSD и Linux, о чем упоминалось в этой главе ранее. И файловую систему для такого совместного раздела следует определить как ext2fs, после чего предписать его автоматическое монтирование в файле /etc/fstab:

```
/dev/ad0s#? /home ext2fs rw auto 0 0
```

Обращаю внимание на последнее поле в данной записи: значение 0 для параметра `Pass#` обязательно указывается — во избежание поползновений со стороны FreeBSD к проверке чужой файловой системы, что могло бы повлечь за собой последствия непредсказуемые.

Запуск Linux-приложений

Совместимость FreeBDS с приложениями Linux — также вопрос не одновариантный. Он включает в себя а) преобразование Linux-пакетов в формат, который может быть использован в FreeBSD, и б) собственно запуск бинарных Linux-приложений в FreeBSD.

Первая задача может возникнуть, если требуется скомпилировать приложение, доступное в исходных текстах в виде rpm-пакета, распространяемого, например, с таким дистрибутивом Linux, как Red Hat. На этот предмет существует утилита rpm2cpio. Она считывает данные с rpm-файла (или, если таковой не указан, со стандартного ввода), преобразуя их в формат архива cpio (именно поэтому мы и вспомнили об этом довольно старом и не вполне удобном формате в *главе 14*).

Делается это так:

```
$ rpm2cpio < file.rpm > file.cpio
```

Далее образовавшийся cpio-архив распаковывается стандартным образом:

```
$ cpio -i -d < file.cpio
```

Обращаю внимание на опцию -d, предписывающую создавать промежуточные подкаталоги при распаковке архива — она почти наверняка понадобится.

Разумеется, rpm-пакет может быть распакован и напрямую — одноименной программой rpm, включенной в коллекцию портов FreeBSD. Однако тут следует учесть, что это потребует многих дополнительных опций. Так, просто команда типа

```
$ rpm -ihv file.rpm
```

наверняка результата не даст — программа rpm будет пытаться найти базу данных rpm-пакетов по соответствующему "умолчальному" адресу, установленному в дистрибутивах Linux (обычно — /var/lib/rpm/). И, разумеется, безуспешно. Так что потребуется указание префикса или нового корневого каталога, вероятно — отказ от проверки зависимостей и еще некоторые опции, к чему я вернусь несколькими абзацами ниже. Так что для распаковки пакета с исходными текстами более простым способом может оказаться предварительное преобразование rpm-пакета в архив cpio.

Преобразование такое, кроме как для работы с исходниками, может потребоваться и в некоторых иных случаях. Например: на протяжении всего своего повествования я неоднократно упоминал, что при всех своих достоинствах дистрибутив FreeBSD отнюдь не блещет богатством кириллических шрифтов, как для консоли, так и для системы X Window. Однако богатство это может быть отыскано в отечественных дистрибутивах Linux, например, производства фирмы Altlinux (**http://www.altlinux.ru**). Последние же, принадлежа к линии Red Hat, распространяются, естественно, в виде rpm-пакетов. Так что никто не запрещает, приобретя, скажем, недавно вышедший дистрибутив Altlinux Master 2 (или скачав нужные пакеты из репозитория Sisyphus), заимствовать из него требуемые кириллические шрифты. Для чего и потребуются средства работы с пакетами rpm.

Вторая задача, возникающая при использовании Linux-приложений в FreeBSD, — обеспечение их запуска под управлением этой системы. Решается она подчас очень просто — посредством системы linux-base, устанавливаемой из бинарного пакета или из коллекции портов, как любое другое приложение. Единственное, что требуется после этого — это добавить в файл /etc/rc.conf строку

```
linux_enable="YES"
```

Впрочем, если воспользоваться программой sysinstall, все необходимые действия будут выполнены автоматически. Для этого в главном меню программы выбирается пункт **Configure** (Настройка), в его подменю — пункт **Startup** (Стартовые сервисы), в котором включается опция **linux (This host wants to be able to run linux binaries.)**. Вслед за этим автоматически устанавливается система linux-base (разумеется, соответствующий пакет или порт должны быть доступны на CD, жестком диске, FTP-сервере или ином источнике,

определенном командой меню **Configure — Media** (Источник установки)) и вносятся требуемые изменения в конфигурационный файл.

Теперь существует теоретическая возможность запуска Linux-приложений. Чтобы претворить ее в действительность, дело остается за малым — эти самые приложения установить. Однако прежде надо понять, что же делает система linux-base.

А делает она вот что: разворачивает в каталоге /usr/compat/linux нечто вроде теневого дерева каталогов, соответствующего файловой системе Linux. В этом дереве каталогов можно обнаружить подкаталоги bin, sbin, etc, lib, var и др., обеспечивающие функционирование Linux-программ. В частности, в /usr/compat/linux/lib имеет место быть важнейшая системная библиотека libs-2.x.x (от номера ее версии часто и зависит, будут ли конкретные Linux-приложения работоспособными в данной обстановке).

Имеется и каталог /usr/compat/linux/usr, предназначенный для пользовательских приложений и их разделяемых компонентов. Кончено, в соответствии с декларируемым стандартом LSB (Linux Standard Base — см. **http://www.linuxbase.org**), новые приложения по умолчанию должны бы устанавливаться в каталог /usr/compat/linux/opt, но далеко не все из них об этом знают...

Кроме каталога /usr/compat/linux/, в файловой системе FreeBSD можно обнаружить и каталог /compat/linux/. Однако это — всего-навсего символическая ссылка на вышеуказанный, что легко проверить командой

```
$ ls -l /compat
lrwxr-xr-x  1 root  wheel  10  1 май 13:30 compat -> usr/compat
```

Требуемые для установки Linux-приложения действия проще показать на примере, нежели объяснить словами. Так, ни один истинный любитель авторской песни не может пройти мимо сайта **http://www.badrs.ru**, содержащего богатейшую ее коллекцию. Одна беда — коллекция эта представлена исключительно в формате RealAudio. А проигрыватель его (RealPlayer) доступен (на **http://www.real.com**), помимо Windows-версии, только в реализации для Linux. И, не принадлежа к классу открытого софта (хотя и будучи бесплатным в базовой поставке), распространяется без исходных текстов, только в бинарном виде.

Тем не менее, и это не помешает бардолюбу-берклианцу. Отправляемся на **http://www.real.com** и (после довольно длинной и скучной процедуры регистрации) скачиваем бинарный Linux-пакет, например, для стандартного Red Hat. Это будет файл вида rpX_linux20_libc6_i386_rpm или подобный, в зависимости от версии. Копируем его в любой (например, пользовательский) каталог и устанавливаем как обычный rpm-пакет одноименной командой `rpm -i[hv]`.

О чем здесь следует помнить — так это о смене корневого каталога (естественно, таковым должен выступать не /, а /usr/compat/linux/) — во-первых, об указании на путь к базе данных rpm-пакетов — во-вторых, и игнорировании операционной системы — в-третьих. Что достигается указанием опций `--root`, `--dbpath` и `--ignoreos` соответственно. В итоге требуемая команда приобретает вид

```
$ rpm -i --ignoreos --root /usr/compat/linux \
 --dbpath /var/lib/rpm rpX_linuxXX_libc6_XX.rpm
```

После этого RealPlayer можно будет запустить командой с указанием точного пути к исполняемому файлу (из эмулятора терминала графического режима системы X Window, конечно):

```
/usr/compat/linux/usr/local/RealPlayerX/realplay
```

Или внести в переменную `PATH` своего профильного файла дополнительное значение

```
/usr/compat/linux/usr/X11R6/bin
```

В этом каталоге будет символическая ссылка

```
realplay -> /usr/lib/RealPlayerX/realplay
```

Аналогичны и действия для установки и запуска других Linux-приложений — а, согласно руководству пользователя FreeBSD, около 90% их способно успешно функционировать в этой ОС, включая Star Office, Linux-версия Netscape, Adobe Acrobat, RealPlayer 5 и 7, VMWare, Oracle, WordPerfect, Doom, Quake и многие другие.

При установке Linux-пакетов с CD следует помнить, что компакт-диск должен монтироваться не обычным для FreeBSD образом:

```
$ mount /cdrom
```

а в каталог, предусмотренный для этого "теневым" Linux-деревом, т. е.:

```
$ mount_cd9660 /dev/acd0 /usr/compat/linux/mnt/cdrom
```

Кроме того, не исключено, что потребуется предварительная смена корневого каталога:

```
$ chroot /usr/compat/linux/ /bin/bash
```

Проблема может возникнуть с наиболее новыми версиями программ для Linux, требующих для установки и работы высоких версий библиотеки libc-2. В систему linux-base текущих ветвей FreeBSD включена поддержка этой библиотеки в версии libc-2.1.*x*, и потому новейшие (и — очень хорошие) сборки, например, программ проекта OpenOffice.ru (OpenOffice.ru1.0 и Mozilla1.0, доступные на **http://www.altlinux.ru**) устанавливаться просто откажутся. Однако эту проблему можно решить заимствованием более новых

версий библиотеки из какого-либо современного дистрибутива Linux. Идеально — из того же дистрибутива Altlinux Master 2, именно в котором указанные пакеты собирались и тестировались.

О приложениях DOS и Windows в FreeBSD

Существует по крайней мере одна ситуация, в которой пользователю любой UNIX-подобной системы жизненно необходим запуск DOS-программ. Это — обновление версии BIOS машины. И тут обойтись без эмуляторов DOS практически невозможно.

Наиболее простое из таких средств эмуляции — команда `doscmd`, входящая в базовый комплект FreeBSD. Она предназначена в первую очередь для исполнения единичных DOS-команд без загрузки эмулируемой операционной системы как таковой (хотя последнее тоже не возбраняется).

Далее — pcemu, "полноценный" (если это понятие в данном случае применимо) эмулятор реального режима процессоров i86. То есть нечто, позволяющее запустить на P-4/2 ГГц стандартную PC-XT первого призыва, работающую (согласно описанию программы в /usr/ports/emulators/pcemu/pkg-descr) со скоростью PC-AT286. Чего, впрочем, для перепрошивки BIOS более чем достаточно.

Для эмуляции Windows-программ предназначена программа wine. Подобно doscmd, она не требует наличия эмулируемой ОС, позволяя, однако, запускать как 16-, так и 32-разрядные Windows-приложения, в том числе и игры, использующие DirectX. Правда, настройка ее — дело не самое простое...

Наконец, под FreeBSD возможен запуск Linux-версии программы VMWare — виртуальной машины, использующей защищенный режим процессора i386, "внутри" которой можно установить (и — использовать) почти любую из широко распространенных операционных систем, среди которых — и Windows линий 9x/ME и NT/2000/XP. Программа эта не принадлежит к категории открытых и в общем случае даже не является бесплатной (хотя при оговоренных условиях ее можно использовать безвозмездно, т. е. даром). Кроме того, наличие VMWare отнюдь не освобождает от приобретения копии ОС Windows (лицензионной, конечно — а вы что подумали?).

Вообще говоря, проблема запуска Windows-приложений — широка, многогранна и многоаспектна. И в предыдущих абзацах я даже вкратце на собирался ее охватить (тем более, что лично меня она как-то не очень трогает). А потому могу лишь констатировать, что в принципе такие возможности имеются. Как их реализовать — остается задачей для заинтересованных читателей в качестве домашнего упражнения...

Дополнительные источники

Основные сведения по использованию FreeBSD совместно с другими ОС следует искать в официальной документации проекта (и ее русских переводах, ссылки на которые многократно приводились в предыдущих главах).

Описание мультисистемного загрузчика GRUB можно найти в статье Владимира Попова: "GRUB действительно — Grand" (**http://www.softerra.ru/review/util/15595**). А вопросы эмуляции DOS/Windows в среде UNIX-подобных систем рассмотрены в статье: Калошин В. Свой среди чужих. — Чип, 2002. № 5 — с. 130—131.

Глава 24

Что день грядущий нам готовит

Кто судьбу свою предскажет?
Что сулит грядущий день?
Перед сном совсем не в тяжесть
Рюмка тминной или коктейль.

Арчибалд Сейс, английский востоковед

Вот и позади значительная часть книги. Надеюсь, ее читатели не ограничились просмотром текста, а по ходу дела поставили FreeBSD и, что называется, пощупали ее руками. Если да — они согласятся со мной в следующем: чем дольше работаешь в FreeBSD, тем система кажется все более родной и близкой, я бы сказал — домашней и уютной. В ней уже сделаны свои настройки, накоплены собственные сценарии и конфигурационные файлы, созданы удобные клавиатурные раскладки И расставаться со всем этим — все равно, что покидать отчий дом.

Однако любая, самая совершенная, система время от времени нуждается в обновлении или, говоря по простому, в update. Смена версий — неизбежна, как смена вех. И как в этом плане выглядят перспективы FreeBSD?

Все сказанное в этой книге ранее относилось к текущей стабильной ветке системы. В частности, личный мой опыт охватывает ее версии от 4.2 до 4.6. И должен заметить, что в ней от версии к версии появлялись небольшие, но полезные (и приятные) изменения. Тем не менее, ни одна из них не требует сноса всего предыдущего и немедленного перехода на новую версию. В принципе, функциональные отличия в рамках 4-й ветки довольно малы, и все усовершенствования могут быть внесены руками по ходу дела, без тотальной переустановки. Благо программа sysinstall и коллекция портов позволяют актуализировать систему в любой момент легко и безболезненно, не опасаясь за сохранность данных (но не освобождая от необходимости их резервного копирования).

Иное дело — переход к новой, 5-й, ветке, вышедшей в конце января 2003 года. Автору в процессе подготовки этой книги довелось ознакомиться с ее предварительными версиями, предназначенными для разработчиков. Впро-

чем, они оказались вполне работоспособными и дают достаточно полное представление об особенностях системы.

Самый простой способ обновить систему до 5-й версии — скачать (или приобрести) iso-образ ее инсталляционного компакт-диска. В целом установка мало отличается от таковой 4-х версий. И перезагрузка проходит без проблем.

Впрочем, можно воспользоваться предоставляемой программой sysinstall возможностью обновления ранее установленной ОС FreeBSD 4-х версий. Для этого нужно выбрать в пункт **Upgrade** (Обновление) главного ее меню и проделать все предписанные действия (ничем не отличающиеся от тех, что требуются при установке). Преимущество этого пути — полное сохранение ранее сделанных настроек. За исключением конфигурации ядра — оно потребует пересборки в любом случае. И при этом следует руководствоваться комментариями к новому ядру — некоторые опции настройки в нем изменились.

Теоретически рассуждая, если обойтись без полной переустановки, после обновления системы необходимо выполнить процедуру, именуемую "полный rebuild". То есть — полную перекомпиляцию базовых компонентов системы.

Само по себе это не сложно — нужно просто войти в каталог /usr/src и дать последовательно команды:

```
$ make buildworld
```

вызывающую перекомпиляцию, и

```
$ make installworld
```

устанавливающую пересобранные компоненты на должные места. Оба эти этапа можно объединить воедино командой

```
$ make world
```

Как говорят резонные люди, в одних случаях удобнее первый путь, в иных же — второй. На мой взгляд, при разделении компилирования и установки проще локализовать ошибки, буде таковые проявятся.

А сообщение об ошибке неизбежно возникнет, если при установке (или обновлении) не было предписано установить все исходные тексты вообще. Что понятно — процесс дойдет до компиляции, скажем, Games, и обнаружится, что исходных текстов игр нет. Чтобы избежать этого, процедуру пересборки придется разбить на ряд стадий. Каждая из них начинается с перехода в корневой каталог исходных текстов соответствующего базового компонента (например, в /usr/src/bin, и т. д.), в котором и следует отдавать вышеуказанные команды.

Должен предупредить, что процесс полной "перестройки" системы весьма долгий — на машине любой мощности он займет не один час. Так что ошибок, по возможности, лучше избегать. И не следует надеяться, что он приведет к видимому повышению производительности при оптимизации под конкретный процессор, например: по крайней мере, для Pentium 4 никакого прироста быстродействия заметить не удается. Однако, по словам тех же резонных людей, без него многие (если не все) приложения базового комплекта могут отказаться работать — ведь обновление системы приводит к установке не просто пересобранного, а нового ядра. Для Linux-пользователя это было бы эквивалентно переходу с ядра ветви 2.2 на ветвь 2.4 (хотя там проблем с работоспособностью приложений, как будто, не возникает).

И еще: я вовсе не призываю к немедленному переходу на FreeBSD 5. По свидетельствам очевидцев в первых версиях новой ветки часто могут встречаться ошибки, отлавливаемые обычно к версии *x*.2 (впрочем, собственного опыта у меня нет). Так что все сказанное можно рассматривать как мысленный эксперимент. Однако порядок обновления, вероятно, не изменится, так что к нему можно подготовиться заранее.

Установив тем или иным способом FreeBSD 5.0 (пусть даже виртуально), посмотрим, чем же новым она нас порадует. Первое, на что напирают разработчики, — улучшенная поддержка многопроцессорных систем. Однако не думаю, что это актуально для моих потенциальных читателей.

Далее — новая файловая система, UFS2. Обещается, что она будет и быстрее (а ведь это — до сих пор слабое место BSD-систем по сравнению с Linux), и устойчивее прежней. Правда, проверить это пока не удалось.

Кроме этого, в 5-й ветке реализована полноценная файловая система устройств — devfs. Что это такое — знают пользователи достаточно новых дистрибутивов Linux. То есть вместо единого каталога /dev, в который кучей свалены все устройства, как реально существующие в системе, так и те, которые только могут в ней существовать, предусмотрены отдельные каталоги для дисковых накопителей, приводов CD-ROM, терминалов и т. п. Причем для подключения устройств не потребуется создавать их с помощью сценария /dev/MAKEDEV или команды `mknode` — это делается само собой, с автоматическим присвоением старшего и младшего номеров устройств. Вплоть до того, что для PCMCIA или USB-устройств соответствующие файлы будут возникать и исчезать на глазах пользователя по мере их подключения или отключения.

Произошли и некоторые изменения в логической файловой системе FreeBSD (т. е. в дереве каталогов). В частности, программа sysinstall переместилась на предназначенное ей логикой место — в каталог /sbin.

Главные же новшества обнаруживаются глубоко в недрах ядра. Разумеется, в нем потребовались опции для поддержки новых файловых систем. Далее,

режим совместимости с Linux-приложениями также потребовал включения соответствующей опции

```
Options SYSVSEM
```

И еще как будто бы реализована поддержка полноценной графической консоли через линейный кадровый буфер, аналогичная имеющейся в Linux, с возможностью переключения разрешений (вроде 800×600 или 1024×768). Правда, с моей видеокартой (Matrox G450) такая попытка привела к мертвому зависанию машины.

Да, чуть не забыл: из ядра отныне исключена поддержка процессора i386 — за счет чего можно ожидать увеличения производительности на процессорах i486 и выше. Впрочем, как уже говорилось, на примере Pentium 4 я этого не заметил.

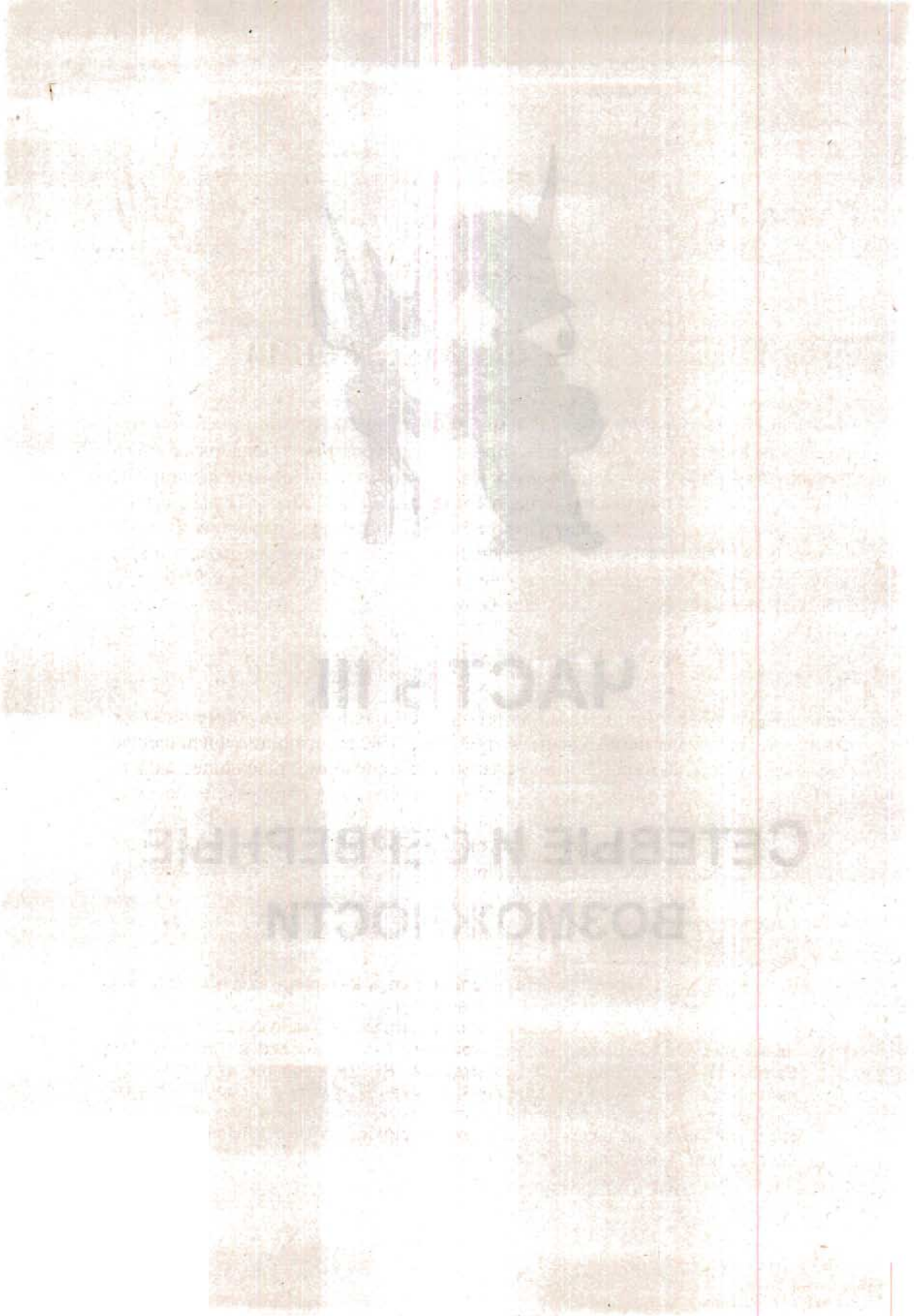
Подвожу итоги: будучи обычным пользователем, я не заметил большой разницы между 4-й и 5-й ветками. И это не удивительно — ведь традиционность и преемственность суть отличительные черты этой системы. От ветви к ветви она работает точно так же, как раньше. Только — лучше. И надеюсь, так будет продолжаться и впредь...

Дополнительные источники

- С подробнейшим, от отдельного порта и пакета, перечнем новшеств 5-й ветки FreeBSD можно ознакомиться в Release Notes на сайте проекта (**http://www.freebsd.org/freebsd5/relnotes-i386.html**).
- А почитать о новой файловой системе devfs (правда, применительно к Linux) можно в замечательных статьях Дэниэла Роббинса, русский перевод которых имеет место быть на **http://linux.yaroslavl.ru/Docum/Fs/1-fs_ru.html**.

ЧАСТЬ III

Сетевые и серверные возможности



Глава 25

FreeBSD в сетевом окружении

Времена, когда операционные системы разделялись на принципиально сетевые и принципиально несетевые, прошли безвозвратно. Поддержка стека протоколов TCP/IP встроена в ядро настольной однопользовательской ОС Windows XP точно так же, как в ядро FreeBSD. Но ОС FreeBSD изначально проектировалась как сетевая платформа. Компьютер, работающий под управлением ОС FreeBSD без подключения к сети, использует только половину своих возможностей. Сеть, не использующая компьютеры с ОС FreeBSD по назначению, теряет еще больше.

Основы компьютерных сетей

Сеть — это объединение компьютеров, позволяющее им обмениваться данными. Для этого необходимы устройства, обеспечивающие физическую связь между компьютерами и программное обеспечение, решающее задачи приема и передачи информации. Правила обмена информацией (сетевые протоколы) у сетей разных типов разные. ОС FreeBSD проектировалась вместе с комплектом (стеком) протоколов TCP/IP. Сеть, использующая для передачи данных стек протоколов TCP/IP, как известно, называется Интернет.

Примечание

Сеть ARPANET, выросшая впоследствии в сеть Интернет, задумывалась как исследовательская платформа, независимая от какого-либо производителя. Все протоколы TCP/IP основаны на стандартных документах, доступных как для использования, так и для критики. Они называются "Request for Comment" — "Предложение для обсуждения", RFC. Как правило, это полезные, написанные без лишних усложнений документы, которые помогают понять принципы TCP/IP и многих других протоколов. Найти сборник всех RFC в сети Интернет можно на различных сайтах, например, по URL (Uniform Resource Locator) **http://www.faqs.org/rfcs**.

Принципы TCP/IP. Модель OSI

Название TCP/IP составлено из названий двух протоколов разных уровней стека: Internet Protocol (IP, RFC 791) и Transmission Control Protocol (TCP, RFC 793). В набор TCP/IP входят еще несколько протоколов: Internet Control Message Protocol (ICMP, RFC 792), Address Resolution Protocol (ARP, RFC 826) и User Datagram Protocol (UDP, RFC 798). Универсальность и модульность TCP/IP способствовали его признанию и успешному распространению не только в глобальных сетях (WAN — World Area Network), для которых он разрабатывался, но и в локальных (LAN — Local Area Network), где он успешно конкурирует и сосуществует с протоколами корпоративных сетей. Сетевые функции ОС FreeBSD тесно связаны с особенностями TCP/IP.

Основным принципом передачи данных является разбиение общего массива данных на фрагменты, которые встраиваются в пакеты-сообщения (дейтаграммы). Пакеты снабжаются дополнительной информацией (заголовками, а иногда еще и окончаниями), необходимой для их продвижения в сети. Одной из основных идей создателей TCP/IP была идея разделения процесса передачи данных на несколько независимых друг от друга этапов (уровней). Выполнение каждого этапа обеспечивается отдельным аппаратно-программным модулем. Каждый модуль, выполняя строго определенную функцию, изолирует данные своего уровня от вышележащих уровней. Изолированность и стандартизация межуровневых интерфейсов обеспечивают независимость протоколов разных уровней, создавая основу для объединения разнородных компьютеров в единую сеть. Каждый нижележащий уровень добавляет к передаваемому пакету свой заголовок. Ни на одном уровне нет средств, чтобы "заглянуть" внутрь пакета, пришедшего "сверху".

Международная организация стандартов (ISO, International Organization for Standartization) в 1983 году, спустя девять лет после появления комплекса протоколов TCP/IP, приняла стандарт модели взаимодействия открытых систем (Open System Interconnection, OSI), имеющий модульную структуру. В модели OSI идеи модульности и независимости доведены до логического предела. Модель OSI предусматривает семь уровней сетевого соединения:

- Физический
- Канальный
- Сетевой
- Транспортный
- Сеансовый
- Представления данных
- Прикладной

То ли из-за некоторой надуманности и формализма, то ли из-за несвоевременности появления эта модель не получила популярности у производителей, тогда как стандартом де-факто стал TCP/IP. Стек протоколов TCP/IP несколько отличается от семиуровневого стандарта OSI, сохраняя модульность и определенность межмодульных интерфейсов. TCP/IP-модель сетевого соединения разделена лишь на четыре уровня, приблизительное соответствие которых уровням модели OSI показаны на рис. 25.1.

Рис. 25.1. Сравнение моделей TCP/IP и OSI

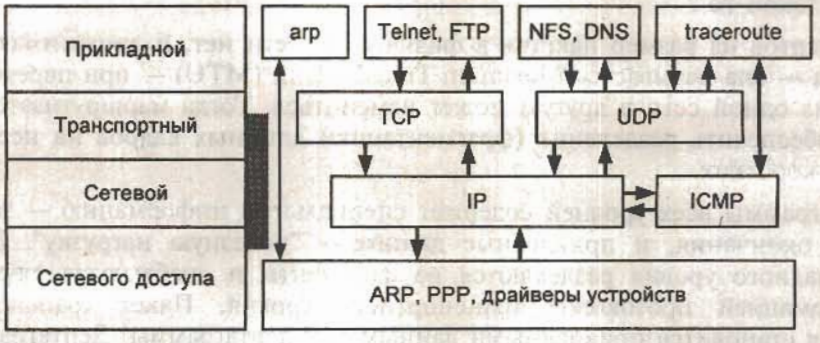


Рис. 25.2. Взаимосвязь и взаимодействие протоколов стека TCP/IP

Сетевой и транспортный уровни одинаково выделены в обеих моделях — именно эти уровни отвечают за передачу пакетов данных от отправителя к получателю. Видимо, здесь теоретики ISO не смогли преодолеть влияние практики TCP/IP. В действительности взаимодействие различных компонентов TCP/IP сложнее формальной уровневой схемы (рис. 25.2).

Протоколы стека TCP/IP

К протоколам прикладного уровня в модели сетевого соединения по TCP/IP относятся все протоколы, использующие на транспортном уровне TCP или UDP. Наиболее известные из них: File Transfer Protocol (FTP), Network Terminal Protocol (Telnet), Simple Mail Transfer Protocol (SMTP), Domain Name Service (DNS), Network File System (NFS), Routing Information Protocol (RIP) и многие другие.

На транспортном уровне используются протоколы TCP и UDP, на сетевом — IP и ICMP. Сообщения ICMP встраиваются в дейтаграммы IP и расширяют возможности IP в получении информации о состоянии сети, управлении скоростью передачи данных и в оптимизации маршрута передачи.

Ниже сетевого уровня множество различных протоколов обеспечивают взаимодействие IP со всеми типами сетей. Самый известный из них — ARP (Address Resolution Protocol), протокол разрешения адресов.

На канальном уровне доступны только аппаратные адреса участников передачи, сфера известности которых ограничивается сегментом физической сети, тогда как маршрутизация IP-пакета осуществляется по IP-адресу. Для определения аппаратного адреса компьютера по IP-адресу используется протокол ARP. Он применяется только в сетях, поддерживающих режим широковещательной передачи, и только в пределах одного физического сегмента сети. ARP-запрос "Кто знает аппаратный адрес компьютера с IP-адресом 149.80.217.5?" адресуется всем доступным компьютерам сегмента. Компьютер, который узнал свой IP-адрес, отвечает "Это я. Мой аппаратный адрес такой-то".

Стандартов на размер пакетов в физической сети нет, и допустимая длина пакета — она называется Maximum Transfer Unit (MTU) — при переходе пакета из одной сети в другую может измениться. Тогда маршрутизатор должен обеспечить разделение (фрагментацию) длинных кадров на несколько более коротких.

Дейтаграммы всех уровней содержат специальную информацию — заголовки и окончания, и прикладные данные — "полезную нагрузку". Данные прикладного уровня разделяются на фрагменты и снабжаются служебной *информацией протоколов транспортного уровня. Пакет транспортного* уровня становится прикладными данными IP-дейтаграммы. Дейтаграмма IP при переходе на нижележащий уровень "заворачивается" в кадр (так называются кванты данных на этом уровне) и формирует его "полезную нагруз-

ку". Кадр уровня сетевого доступа передается компьютеру-адресату в физическом сетевом сегменте, подчиняясь правилам, определяемым протоколами конкретной физической сети. Принимающая сторона разворачивает "посылку" в обратном порядке: Ethernet-кадр → IP-пакет → TCP-пакет.

На рис. 25.3 показано преобразование TCP-пакета при его переходе от транспортного уровня до уровня сетевого доступа.

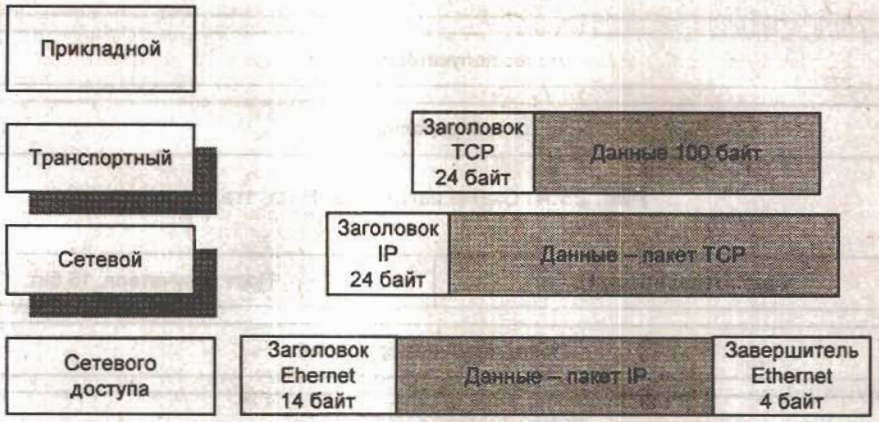


Рис. 25.3. "Обрастание" заголовками TCP-пакета при переходе на нижележащие уровни

Информация в заголовках протоколов

Заголовок Ethernet-кадра содержит аппаратные адреса отправителя (источника) и получателя, сведения о длине кадра и контрольную сумму кадра. Заголовок IP-пакета содержит в себе до шести 32-разрядных слов, включающих IP-адреса получателя и отправителя; длину заголовка (шестое слово заголовка может отсутствовать) и его контрольную сумму; время жизни пакета; информацию о фрагментированности и приоритетности пакета и некоторые другие сведения (рис. 25.4).

IP-пакет не предназначен для надежной передачи данных, функции контроля над качеством соединения вынесены на более высокие уровни. Только этот уровень знает уникальный сетевой адрес, однозначно идентифицирующий компьютер-получатель. IP-пакет может просто "затеряться": тогда любой маршрутизатор прекратит существование такого пакета по истечении его времени жизни.

Заголовок TCP-пакета содержит информацию, необходимую для установления двустороннего соединения с контролем передачи, структура его еще сложнее, чем структура заголовка IP (рис. 25.5).

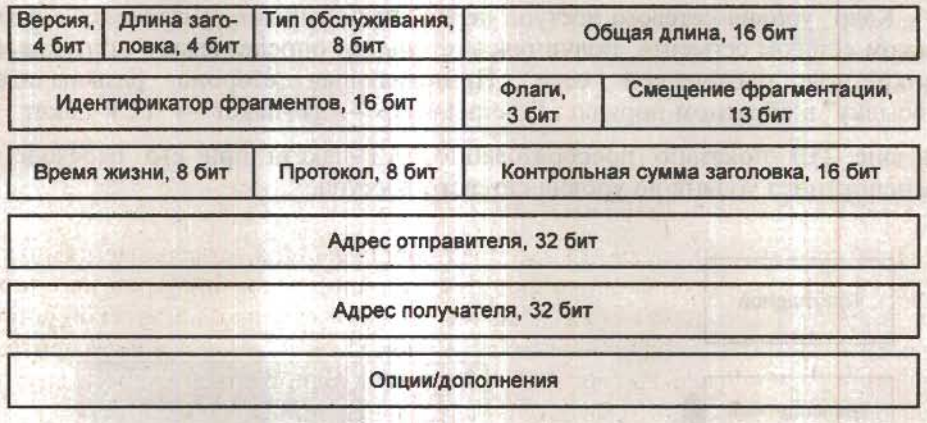


Рис. 25.4. Схема заголовка IP-пакета

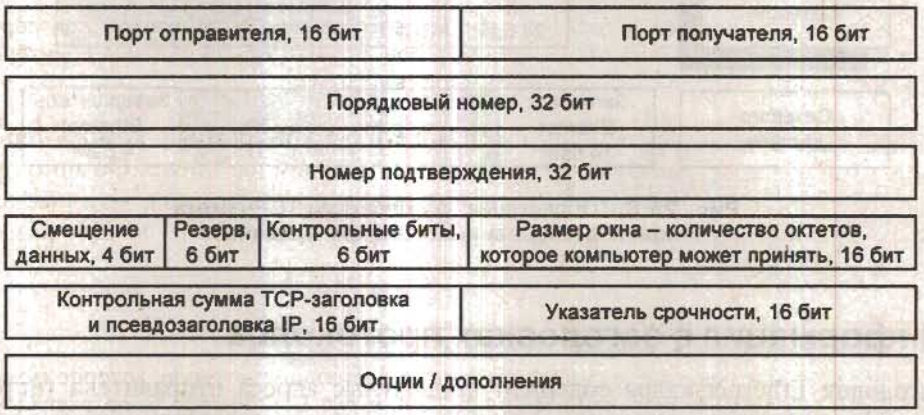


Рис. 25.5. Схема заголовка TCP-пакета

Программа — получатель пакета на компьютере-адресате определяется номером порта, указанным в заголовке.

Второй протокол транспортного уровня, используемый в Интернете для передачи данных без контроля получения, — это протокол UDP, который имеет заголовок, состоящий всего из двух слов (рис. 25.6).

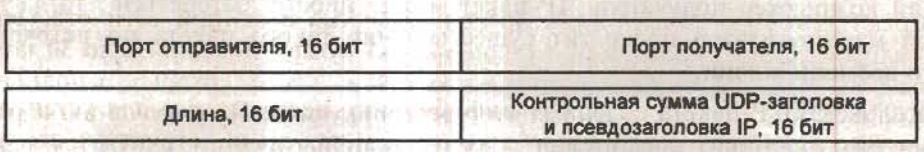


Рис. 25.6. Заголовок UDP-пакета

Несмотря на ненадежность протокола UDP, многие программы прикладного уровня применяют именно его для передачи данных, а не TCP, из-за высоких накладных расходов последнего.

Адресация в Интернете. Сетевая маска. Классы сетей

В любой сети можно выделить два рода составляющих: отдельные компьютеры (хосты) и группы компьютеров — сети. Адреса компьютеров из одной сети имеют общий начальный префикс и частное уникальное окончание. Выделение IP-адресов на уровне сетей и блоков сетей производится централизованно, т. к. адреса Интернета не должны дублироваться. Адреса внутри сетей распределяются по усмотрению владельцев сетей.

Примечание

Хост (host) — любое устройство, являющееся конечным адресатом при передаче данных по TCP/IP. Кроме компьютеров, это могут быть сетевые принтеры, маршрутизаторы (router), переключатели (switch) и др.

IP-адрес однозначно идентифицирует и сеть, и конкретный компьютер в этой сети. Он представляет собой 32-разрядное двоичное число. Обычно это число разделяют на четыре группы по восемь разрядов (октет), которые записывают в виде трехзначных десятичных чисел в диапазоне 0—255, разделенных точками, например, 215.17.103.146.

Примечание

IP-адрес накладывает ограничение на количество компьютеров, которое может быть в сети Интернет: всего 4 294 967 296 штук. В 1996 году была выпущена следующая версия протокола IP — IPv6, которая имеет уже 128-разрядный адрес. (Действующая версия IP — IPv4 имеет 32-разрядный IP-адрес) Адресов, зарезервированных в протоколе IPv6, должно хватить надолго. Современные версии ОС FreeBSD включают полную поддержку этого протокола.

Число компьютеров в сети (объем сети) характеризуется сетевой маской — 32-разрядным числом, в котором разряды, соответствующие адресу сети, заполняются единицами, а разряды, соответствующие адресам хостов, — нулями. Число записывается тетрадой трехзначных десятичных чисел или шестнадцатеричным числом. Например, маска 255.255.255.224 (0xffffffe0) показывает, что пять последних разрядов IP-адреса в данной сети отведены под адреса компьютеров. Сеть с такой маской может объединять до 30 компьютеров, т. к. адреса, все разряды "хостовой" части которых заполнены только нулями или только единицами, резервируются. По умолчанию адрес, в котором все разряды "хостовой" части заполнены нулями, считается адресом сети и записывается обычно так: XXX.XXX.XXX.XXX/NN (например,

215.17.103.128/27). Число за дробной чертой означает длину в двоичных разрядах сетевой части адреса. Адрес с единицами в "хостовых" разрядах является широковещательным адресом сети (broadcast). Сообщения, адресованные всем компьютерам в сети, направляются по этому адресу.

По объему выделяются три класса сетей: A, B и C с масками 255.0.0.0, 255.255.0.0 и 255.255.255.0 соответственно. Отличительным признаком класса сетей являются начальные биты их адресов. Первый бит адресов класса A всегда равен 0, т. е. первый октет адреса такой сети может принимать значения от 1 до 126. Первые два бита в адресах сетей класса B равны 1 и 0, и первый октет принимает значения от 128 до 191. Значения первых трех байтов адресов сетей класса C равны 1, 1 и 0, а диапазон значений первого октета составляет 192—223.

Кроме того, в первом октете никогда не используется значение 127, отведенное для закольцовывающегося интерфейса (всякий хост найдет по адресу 127.0.0.1 самого себя).

Адреса, начальный октет которых имеет значение от 224 до 254, иногда называют адресами класса D. Они отведены для специального использования другими протоколами в качестве групповых адресов. Как уже упоминалось, в адресах никогда не используются значения 0 и 255.

Три задачи поддержки сетевого взаимодействия

Операционная система компьютера, входящего в сеть (Интернет), должна обеспечить прием, передачу и все преобразования данных в соответствии с правилами сетевых протоколов. В процессе поддержки сетевого взаимодействия можно выделить три основные задачи:

- Адресации
- Маршрутизации
- Распределения данных

Задача адресации заключается в определении и поддержке адреса, назначенного компьютеру. Задача маршрутизации заключается в определении направления пересылки пакета. Задача распределения заключается в направлении данных к программе-получателю.

В качестве дополнительной, четвертой, задачи можно рассматривать организацию разрешения имен.

Что такое интерфейс?

Интерфейсом в рассматриваемом контексте называется соединение компьютера с сетью. Оно может осуществляться через сетевую карту или модем.

Модемы, как правило, используются для временного соединения типа "точка-точка". Для постоянного выхода в сеть в подавляющем большинстве случаев используются сетевые карты. Поддержка драйверов сетевых карт осуществляется ядром FreeBSD. Найти карту, которая бы не поддерживалась FreeBSD, трудно. Драйверы самых распространенных карт включены в стандартное (GENERIC) ядро, они опознаются системой при первой же загрузке (табл. 25.1). ОС FreeBSD поддерживает различные аббревиатуры для имен устройств, соответствующих сетевым картам различных типов, отражающие их названия. Ядро при загрузке системы находит сетевую карту, определяет ее тип и в дальнейшем разрешает обращение к карте по имени устройства.

***Таблица 25.1.** Сетевые карты, поддерживаемые ядром FreeBSD по умолчанию*

Название интерфейса устройства	Сетевой адаптер (группа адаптеров), соответствующий устройству
de	# DEC/Intel DC21x4x ("Tulip")
txp	# 3Com 3cR990 ("Typhoon")
vx	# 3Com 3c590, 3c595 ("Vortex")
dc	# DEC/Intel 21143 and various workalikes
fxp	# Intel EtherExpress PRO/100B (82557, 82558)
pcn	# AMD Am79C97x PCI 10/100 NICs
rl	# RealTek 8129/8139
sf	# Adaptec AIC-6915 ("Starfire")
sis	# Silicon Integrated Systems SiS 900/SiS 7016
ste	# Sundance ST201 (D-Link DFE-550TX)
tl	# Texas Instruments ThunderLAN
tx	# SMC EtherPower II (83c170 "EPIC")
vr	# VIA Rhine, Rhine II
wb	# Winbond W89C840F
wx	# Intel Gigabit Ethernet Card ("Wiseman")
xl	# 3Com 3c90x ("Boomerang", "Cyclone")
ed0	at isa? port 0x280 irq 10 iomem 0xd8000

Поддержка других карт требует перекомпиляции ядра. *При установке карт*, предназначенных для шины PCI, трудностей с обнаружением и настройкой устройства, как правило, не возникает. Номера прерывания IRQ (Interrupt Request) и номера каналов DMA (Direct Memory Access), адреса портов вво-

да/вывода (I/O Port Address) приходится настраивать вручную при установке старых ISA-карт.

Кроме реальных сетевых адаптеров, ОС FreeBSD поддерживает сетевые псевдоустройства (табл. 25.2), обеспечивающие решение разного рода сетевых задач повышенной сложности.

Таблица 25.2. Сетевые псевдоустройства GENERIC ядра FreeBSD

Название псевдоустройства	Комментарий
loop	# Network loopback
ether	# Ethernet support
sl 1	# Kernel SLIP
ppp 1	# Kernel PPP
tun	# Packet tunnel.
gif	# IPv6 and IPv4 tunneling
faith 1	# IPv6-to-IPv4 relaying (translation)
bpf	#Berkeley packet filter

Первая из трех задач поддержки сетевого взаимодействия — задача адресации — во всех UNIX-системах решается утилитой ifconfig — программой настройки параметров сетевого интерфейса. Адреса для всех интерфейсов компьютера назначаются этой утилитой при старте системы. С помощью ifconfig можно изменять все настройки интерфейса во время работы системы — перезагрузка для введения в действие новых значений параметров не нужна. FreeBSD-версия ifconfig отличается исключительно широким набором возможностей, которые присутствуют далеко не во всех UNIX-системах. Как все системные утилиты UNIX, ifconfig требует полномочий суперпользователя (superuser) для изменения параметров интерфейса и может вызываться любым пользователем в информационных целях.

Утилита ifconfig, вызванная только с именем интерфейса, выдает информацию о параметрах и состоянии указанного интерфейса. Опция `-a` и отсутствие имен интерфейсов в командной строке заставят утилиту ifconfig вывести информацию обо всех имеющихся интерфейсах.

Условно можно выделить несколько аспектов применения утилиты ifconfig:

- сбор информации об интерфейсе;
- назначение адреса для связи с сетью;
- назначение адреса для соединения типа "точка-точка";
- переключение состояния интерфейса.

Для назначения адреса с помощью утилиты ifconfig командная строка имеет следующий формат:

```
ifconfig [ -L ] [-m ] интерфейс [ тип_адреса ]
                      адрес[/длина_префикса] [ параметры ]
```

где можно перечислить следующие параметры:

- `интерфейс` — название устройства, соответствующего сетевой карте;
- `тип_адреса` — тип сети (сетевое семейство), по умолчанию `inet`. Все возможные значения: `inet`, `inet6`, `atalk`, `ether`, `ipx`;
- `адрес[/длина_префикса]` — назначаемый интерфейсу адрес;
- `параметры` — дополнительные параметры, назначаемые интерфейсу. Они весьма разнообразны — от переключателя состояний интерфейса до настроек опций передающей среды.

Например, следующая команда назначает адрес 202.19.100.11 интерфейсу ed0:

```
ifconfig ed0 202.19.100.11
```

Интерфейс может быть либо выходом в некоторую сеть, либо одним из концов соединения типа "точка-точка" (point-to point link). По умолчанию утилита ifconfig предполагает, что адрес, назначенный интерфейсу, является адресом в сети класса A:

```
ifconfig ed0
ed0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet 202.19.100.11 netmask 0xff000000 broadcast 202.255.255.255
        ether 00:d2:b8:0d:2c:2e
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
```

Поэтому для сетей другого объема нужно обязательно указывать сетевой префикс либо в форме *адрес/префикс*:

```
ifconfig ed0 202.19.100.11/24
```

либо с помощью параметра `netmask`:

```
ifconfig ed0 202.19.100.11 netmask 0xffffff00
```

Широковещательный адрес, если он по какой-то причине отличается от стандартного значения, можно назначить с помощью параметра `broadcast`.

Один и тот же интерфейс может иметь несколько адресов, такие адреса называются псевдонимами (alias) уже назначенного адреса. Так, например, тому же интерфейсу ed0 можно назначить адрес из другой сети с маской 255.255.255.224:

```
ifconfig ed0 192.168.0.1 netmask 0xffffffe0 alias
```

и организовать выход в Интернет для не анонсируемой сети 192.168.0.0/27. Если не указать параметр `alias`, ранее назначенный адрес будет просто заменен. Если по какой-то причине создается псевдоним с адресом из той же сети, что и основной адрес, тип соединения должен определяться как "точка-точка". Соединение такого рода определяется сетевой маской 255.255.255.255:

```
ifconfig ed0 202.19.100.95/32 alias
```

и интерфейс ed0 будет принимать пакеты, направляемые на адреса 202.19.100.11, 202.19.100.95 и 192.168.0.1. Соединения типа "точка-точка" применяются для конфигурирования IP-туннелей: прямых виртуальных каналов через Интернет, применяющих для передачи информации инкапсуляцию (вложение) IP-пакетов. Тогда в параметры утилиты ifconfig должен входить адрес другого конца туннеля и параметр `tunnel`, задающий адреса источника/приемника для внешних заголовков инкапсулирующих IP-пакетов. (Только для устройств-интерфейсов типа IP-tunnel.)

Остальные параметры интерфейса можно разделить на 4 группы, используемые при:

- переключении состояния интерфейса;
- создании/отмене псевдонимов;
- настройке параметров нижележащего уровня сетевого соединения;
- добавлении/удалении псевдоустройств интерфейса.

Интерфейс может быть включен или выключен. При выключенном интерфейсе система не пытается посылать сообщения через него. Перевести интерфейс из одного состояния в другое можно установкой параметров `up` и `down`:

```
ifconfig ed0 down
```

Эта команда "закроет" интерфейс ed0.

Назначенный интерфейсу псевдоним можно отменить параметром `-alias` или одним из его синонимов: `delete`, `remove`. (Многие параметры утилиты ifconfig имеют синонимы для совместимости с аналогичными параметрами утилиты ifconfig других UNIX.)

Параметры настройки нижележащего уровня позволяют, например, задавать MTU (Maximum Transmission Unit)

```
ifconfig ed0 mtu 750
```

перепрограммировать аппаратный адрес (Ethernet)

```
ifconfig ed0 ether 00:c0:d7:ba:3d:21
```

или конкретизировать среду передачи, если аппаратура это позволяет, а также устанавливать опции передающей среды. Так, для карты типа

Intel EtherExpress 10/100, поддерживающей скорости передачи 10 Мбит/с и 100 Мбит/с, можно установить нужные значения скорости и режима передачи:

```
ifconfig fxp0 media 100baseTX mediaopt full-duplex
```

Полный перечень возможностей утилиты ifconfig можно получить в прилагаемом системном руководстве.

Примечание

Утилита ifconfig выполняется сразу, перезагрузка системы для введения в действие новых параметров не требуется. Адрес изменяется сразу после выполнения команды, без перезагрузки системы, и ошибочное или намеренное изменение адреса интерфейса может привести к потере сетевого соединения. Поэтому не следует использовать утилиту ifconfig при удаленном доступе с помощью программы telnet или ssh.

Настройка шлюза

Задача маршрутизации заключается в определении аппаратного адреса для IP-пакетов, переходящих на уровень сетевого доступа. Эта задача решается даже тогда, когда интерфейс у компьютера один. Для определения направления передачи пакета ядром операционной системы в оперативной памяти компьютера поддерживается таблица маршрутизации, где указывается, как достичь IP-адрес назначения. Таблицу маршрутизации можно прочитать утилитой netstat:

```
netstat -rn

Routing tables

Internet:
Destination        Gateway            Flags     Refs      Use  Netif Ex-
pire
default            192.168.0.1        UGSc     17656  8579697    fxp0
127.0.0.1          127.0.0.1          UH        4336   623603     lo0
192.168.0/24       link#1             UC           5        0    fxp0
```

Так выглядит таблица статической маршрутизации для хоста с единственным внешним интерфейсом fxp0, выходящим в неанонсируемую сеть класса C. В таблице можно рассмотреть маршрут к локальной сети, закольцовывающийся интерфейс lo0 и default — маршрут по умолчанию, по которому направляются все пакеты с адресами, отличными от трех вышеперечисленных адресов.

Маршрутизация осуществляется ядром системы; делается это весьма часто, поэтому процедура разбора адресов должна выполняться очень быстро. Для сравнения сетевой части IP-адреса с адресом локальной сети производится логическое побитовое умножение целевого IP-адреса и местной сетевой маски. Результат — адрес сети назначения — сравнивается с адресом локальной сети, и пакет либо передается по направлению default, либо распространяется в локальной физической сети.

Аппаратные адреса компьютеров локального физического сегмента доступны. Они также хранятся в оперативной памяти компьютера, в таблице разрешения адресов. Результаты выполнения ARP-запросов какое-то время хранятся в оперативной памяти в виде таблицы, прочитать которую можно утилитой arp:

```
# arp -an
? (10.90.94.14) at 00:02:44:13:08:61
? (10.90.94.15) at 00:c0:df:f2:93:90
? (10.90.94.20) at 00:c0:26:a8:ee:22
? (10.90.94.130) at 00:50:04:f2:54:b6
? (215.91.221.59) at 00:80:c8:b9:16:49
? (215.91.221.130) at 00:04:dc:46:1f:9d
```

На компьютерах с подобной сетевой конфигурацией маршруты определяются статически, как правило, при старте системы утилитой route.

```
route add -default 192.168.0.1
```

Запись в таблице маршрутизации для сети 192.168.0/24 автоматически добавляется при выполнении утилиты ifconfig. Можно, например, добавить маршрут к сети 192.168.1/24, если хост 192.168.0.5 вдруг обнаружил у себя второй интерфейс, выходящий в эту подсеть:

```
route add -net 192.168.1/24 192.168.0.5
```

Второе слово в параметрах утилиты route определяет действие, которое будет производиться над таблицей маршрутизации. Действий может быть 6:

- `add` — добавить маршрут;
- `flush` — удалить все маршруты;
- `delete` — удалить указанный маршрут;
- `change` — поменять некоторые аспекты маршрута, например, шлюз;
- `get` — найти и показать маршрут к заданному пункту назначения;
- `monitor` — перейти в состояние монитора: постоянно сообщать об изменениях в таблице, не выходя в программную оболочку Shell.

С помощью утилиты route можно переменить маршрут по умолчанию default, добавить маршрут к той или иной сети или удалить его, но в практической деятельности она применяется не часто. Статические маршруты

клиентов сети назначаются при старте системы и не изменяются, а шлюзы для построения таблицы маршрутизации используют другие программы.

Распределение данных

Как пакеты находят программы, которым они адресованы и как программы находят свои пакеты? Эта задача в комплексе задач передачи данных по сети решается с помощью портов, номера которых встраиваются в заголовки пакетов транспортного уровня. Порт — 16-разрядное двоичное число. Часть портов закреплена за принимающими программами, это, как правило, число меньше 10 000. Порты с бóльшими номерами изначально свободны и распределяются между процессами, которым они нужны, по правилам, подобным назначению номера процессу или определения файлового дескриптора. Сторона, ожидающая запроса на установление соединения, порождает процесс-демон, связывающийся (bind) с определенным зарезервированным портом. *Демон (daemon)* — фоновый процесс UNIX, не имеющий управляющего терминала и связанного с жизнью терминала сеанса работы в системе. Такой процесс может выполняться бесконечно, управлять им можно только с помощью сигналов. Процесс-демон закрепляет за собой порт и переходит в режим ожидания обращений к этому порту. Это программа-сервер, ожидающая запроса на соединение. Программа, инициирующая обращение, заранее знает номер порта назначения, а номер порта отправителя она получает от системы практически случайным образом из диапазона нерезервируемых портов с высокими номерами. Номер уже занятого порта никогда не будет выдан другому процессу. Таким образом, получается уникальная пара: "порт с зарезервированным номером" — "порт с нерезервируемым номером", которая вместе с парой IP-адресов получателя и отправителя определяет линию передачи данных однозначно. Программа-сервер, получив запрос на соединение по прослушиваемому порту, производит, как правило, копию самой себя и снова возвращается в режим ожидания следующего запроса. Копия сервера, обслуживающая заданное соединение, не переходит в состояние бесконечного ожидания и "тихо умирает" после закрытия соединения.

Программы-получатели

Для того чтобы описанная выше схема работала, необходимо определить порты, которые прослушиваются серверами. Существует соглашение о хорошо известных портах. За каждым номером закреплено определенное название, т. е. название предоставляемого сервиса. Список хорошо известных сервисов в виде

```
имя_сервиса  номер_порта/транспортный_протокол
```

собран в файле /etc/services, фрагмент которого приведен в листинге 25.1.

Листинг 25.1. Содержимое файла /etc/services

```
#
# Network services, Internet style
#
# The latest IANA port assignments can be gotten from
#  http://www.iana.org/assignments/port-numbers
#
# The Well Known Ports are those from 0 through 1023.
# The Registered Ports are those from 1024 through 49151
# The Dynamic and/or Private Ports are those from 49152 through 65535
#
# $FreeBSD: src/etc/services,v 1.62.2.7 2001/08/01 20:17:40 dougb Exp $
#  From: @(#)services  5.8 (Berkeley) 5/9/91
#
# WELL KNOWN PORT NUMBERS
#
rtmp     1/ddp      #Routing Table Maintenance Protocol
tcpmux     1/tcp      #TCP Port Service Multiplexer
tcpmux     1/udp      #TCP Port Service Multiplexer
nbp     2/ddp      #Name Binding Protocol
.....
echo     7/tcp
echo     7/udp
discard     9/tcp      sink null
discard     9/udp      sink null
systat     11/tcp      users  #Active Users
systat     11/udp      users  #Active Users
daytime     13/tcp
daytime     13/udp
.....
chargen     19/tcp      ttytst source  #Character Generator
chargen     19/udp      ttytst source  #Character Generator
ftp-data     20/tcp      #File Transfer [Default Data]
ftp-data     20/udp      #File Transfer [Default Data]
ftp     21/tcp      #File Transfer [Control]
ftp     21/udp      #File Transfer [Control]
ssh     22/tcp     #Secure Shell Login
```

```
ssh    22/udp    #Secure Shell Login
telnet   23/tcp
telnet   23/udp
.....
smtp   25/tcp     mail  #Simple Mail Transfer
smtp   25/udp     mail  #Simple Mail Transfer
.....
time   37/tcp     timserver
time   37/udp     timserver
.....
tacacs   49/tcp    #Login Host Protocol (TACACS)
tacacs   49/udp    #Login Host Protocol (TACACS)
.....
domain   53/tcp     #Domain Name Server
domain   53/udp     #Domain Name Server
.....
#PROBLEMS!=============================================================
#   57/tcp     any private terminal access
#PROBLEMS!=============================================================
#   57/udp     any private terminal access
.....
bootps   67/tcp     dhcps  #Bootstrap Protocol Server
bootps   67/udp     dhcps  #Bootstrap Protocol Server
bootpc   68/tcp     dhcpc  #Bootstrap Protocol Client
bootpc   68/udp     dhcpc  #Bootstrap Protocol Client
tftp   69/tcp     #Trivial File Transfer
tftp   69/udp     #Trivial File Transfer
.....
finger   79/tcp
finger   79/udp
http   80/tcp     www www-http  #World Wide Web HTTP
http   80/udp     www www-http  #World Wide Web HTTP
.....
pop2  109/tcp     postoffice  #Post Office Protocol - Version 2
pop2  109/udp     postoffice  #Post Office Protocol - Version 2
```

```
pop3  110/tcp     #Post Office Protocol – Version 3
pop3  110/udp     #Post Office Protocol – Version 3

.....

auth  113/tcp     ident tap  #Authentication Service
auth  113/udp     ident tap  #Authentication Service

.....

ntp  123/tcp     #Network Time Protocol
ntp  123/udp     #Network Time Protocol

.....

netbios-ns  137/tcp     #NETBIOS Name Service
netbios-ns  137/udp     #NETBIOS Name Service
netbios-dgm  138/tcp     #NETBIOS Datagram Service
netbios-dgm  138/udp     #NETBIOS Datagram Service
netbios-ssn  139/tcp     #NETBIOS Session Service
netbios-ssn  139/udp     #NETBIOS Session Service

.....

imap  143/tcp     imap2 imap4 #Interim Mail Access Protocol v2
imap  143/udp     imap2 imap4 #Interim Mail Access Protocol v2

.....

irc  194/tcp     #Internet Relay Chat Protocol
irc  194/udp     #Internet Relay Chat Protocol

.....

ipx  213/tcp
ipx  213/udp

.....

ldap  389/tcp     #Lightweight Directory Access Protocol
ldap  389/udp     #Lightweight Directory Access Protocol

.....

https  443/tcp
https  443/udp

.....

router  520/udp  route routed  #local routing process (on site);

.....

irc-serv  529/tcp
irc-serv  529/udp

.....
```

```
#PROBLEMS!=============================================
doom  666/tcp     #doom Id Software
doom  666/udp     #doom Id Software
#PROBLEMS!=============================================
.....
# REGISTERED PORT NUMBERS
#
blackjack  1025/tcp   #network blackjack
blackjack  1025/udp   #network blackjack
.....
socks  1080/tcp
socks  1080/udp
checksum  1386/tcp   #CheckSum License Manager
checksum  1386/udp   #CheckSum License Manager
.....
#PROBLEMS!=============================================
orasrv  1525/tcp    #oracle
orasrv  1525/udp    #oracle
#PROBLEMS!=============================================
.....
bootserver  2016/tcp
bootserver  2016/udp
.....
nfsd  2049/tcp  nfs  # NFS server daemon
nfsd  2049/udp  nfs  # NFS server daemon
.....
cvsup  5999/tcp   #CVSup file transfer/John Polstra/FreeBSD
x11  6000/tcp   #6000-6063 are assigned to X Window System
x11  6000/udp
x11-ssh  6010/tcp   #Unofficial name, for convenience
x11-ssh  6010/udp
.....
font-service  7100/tcp   #X Font Service
font-service  7100/udp   #X Font Service
.....
```

Программа-сервер, как правило, "знает" порт, который она должна прослушивать, и, как правило, она может считать номер порта из опций командной строки или из параметров конфигурации. Стартующие сервисы ОС FreeBSD определяются в файле /etc/rc.conf, кроме того, любой сервис можно запустить вручную из командной строки. В UNIX первые 1024 порта являются привилегированными, они могут быть выделены только программам, исполняемым от имени суперпользователя.

Супердемон inetd

Множество демонов, "томящихся" в ожидании запросов, которые могут следовать с частотой раз в час, а то и в реже, напрасно расходует системные ресурсы. В большинстве UNIX-систем для вызова демонов сетевых служб используется супердемон inetd. Он играет роль сетевого диспетчера: регистрирует обращения по всем портам и, в соответствии со своим конфигурационным файлом, вызывает программу, обслуживающую запрошенный порт. Сервер, вызванный супердемоном, не становится демоном и прекращает свое существование после закрытия соединения. В результате, из множества сетевых демонов в оперативной памяти остается только один inetd.

Конфигурационный файл супердемона /etc/inetd.conf имеет формат:

```
Сервис тип_соединения протокол режим владелец полное_имя_файла
↳имя_программы
```

Конфигурационный файл супердемона /etc/inetd.conf выглядит так, как показано в листинге 25.2.

Листинг 25.2. Пример конфигурационного файла супердемона /etc/inetd.conf

```
# $FreeBSD: src/etc/inetd.conf,v 1.44.2.5 2001/08/04 16:06:44 rwatson Exp
$
#
# Internet server configuration database
#
# Define *both* IPv4 and IPv6 entries for dual-stack support.
# To disable a service, comment it out by prefixing the line with '#'.
# To enable a service, remove the '#' at the beginning of the line.
#
#ftp    stream  tcp     nowait  root    /usr/libexec/ftpd       ftpd -l
#ftp    stream  tcp6    nowait  root    /usr/libexec/ftpd       ftpd -l
#telnet stream  tcp     nowait  root    /usr/libexec/telnetd    telnetd
#telnet stream  tcp6    nowait  root    /usr/libexec/telnetd    telnetd
```

```
#shell  stream  tcp  nowait  root  /usr/libexec/rshd  rshd
#shell  stream  tcp6  nowait  root  /usr/libexec/rshd  rshd
#login  stream  tcp  nowait  root  /usr/libexec/rlogind  rlogind
#login  stream  tcp6  nowait  root  /usr/libexec/rlogind  rlogind
#finger  stream  tcp  nowait/3/10 nobody /usr/libexec/fingerd  fingerd -s
#finger stream  tcp6  nowait/3/10 nobody /usr/libexec/fingerd  fingerd -s
#exec  stream  tcp  nowait  root  /usr/libexec/rexecd  rexecd
#uucpd  stream  tcp  nowait  root  /usr/libexec/uucpd  uucpd
#nntp  stream  tcp  nowait  usenet  /usr/libexec/nntpd  nntpd
#
#ntalk  dgram  udp  wait  tty:tty  /usr/libexec/ntalkd  ntalkd
#tftp  dgram  udp  wait  nobody  /usr/libexec/tftpd  tftpd /tftpboot
#bootps  dgram  udp  wait  root  /usr/libexec/bootpd  bootpd
#
#daytime stream  tcp  nowait  root  internal
#daytime stream  tcp6  nowait  root  internal
#daytime dgram  udp  wait  root  internal
#daytime dgram  udp6  wait  root  internal
#time  stream  tcp  nowait  root  internal
#time  stream  tcp6  nowait  root  internal
#time   dgram  udp  wait  root  internal
#time   dgram  udp6  wait  root  internal
#echo  stream  tcp  nowait  root  internal
#echo  dgram  udp6  wait  root  internal
#echo  dgram  udp  wait  root  internal
#echo  stream  tcp6  nowait  root  internal
# example entry for the optional pop3 server
#
#pop3  stream  tcp  nowait  root  /usr/local/libexec/popper  popper
#
# example entry for the optional imap4 server
#
#imap4  stream  tcp  nowait  root  /usr/local/libexec/imapd  imapd
#
#smtp  stream  tcp  nowait  qmaild  /var/qmail/bin/tcp-env  tcp-env
/var/qmail/bin/qmail-smtpd
#
# Enable the following two entries to enable samba startup from inetd
```

```
# (from the Samba documentation).
#
#netbios-ssn stream tcp nowait root /usr/local/sbin/smbd smbd
#netbios-ns dgram udp wait root /usr/local/sbin/nmbd nmbd
```

В этом примере все (!) строки закрыты знаком комментария — это значит, что сервер никаких сетевых соединений через inetd принимать не будет. Но супердемон inetd ОС FreeBSD можно настроить гораздо тоньше. Супердемон имеет встроенный механизм tcp-wrappers, который дает возможность указать с точностью до отдельных хостов, кому можно предоставлять запрошенный сервис, а кому — нельзя. Файл /etc/host.allow содержит перечень ограничений сервиса для хостов, подсетей, доменов и поддоменов (листинг 25.3).

```
#
# hosts.allow access control file for "tcp wrapped" applications.
# $FreeBSD: src/etc/hosts.allow,v 1.8.2.5 2001/08/30 16:02:37 dwmalone
Exp $
#
# NOTE: The hosts.deny file is deprecated.
#       Place both 'allow' and 'deny' rules in the hosts.allow file.
#  See hosts_options(5) for the format of this file.
#  hosts_access(5) no longer fully applies.
# !!! This is an example! You will need to modify it for your specific
# !!! requirements!

# Start by allowing everything (this prevents the rest of the file
# from working, so remove it when you need protection).
# The rules here work on a "First match wins" basis.
ALL : ALL : allow

# Wrapping sshd(8) is not normally a good idea, but if you
# need to do it, here's how
#sshd : .evil.cracker.example.com : deny

# Protect against simple DNS spoofing attacks by checking that the
# forward and reverse records for the remote host match. If a mismatch
# occurs, access is denied, and any positive ident response within
```

```
# 20 seconds is logged. No protection is afforded against DNS poisoning,
# IP spoofing or more complicated attacks. Hosts with no reverse DNS
# pass this rule.
ALL : PARANOID : RFC931 20 : deny

# Allow anything from localhost.  Note that an IP address (not a host
# name) *MUST* be specified for portmap(8).
ALL : localhost 127.0.0.1 : allow
ALL : my.machine.example.com 192.0.2.35 : allow

# To use IPv6 addresses you must enclose them in []'s
ALL : [fe80::%fxp0]/10 : allow
ALL : [fe80::]/10 : deny
ALL : [3ffe:fffe:2:1:2:3:4:3fe1] : deny
ALL : [3ffe:fffe:2:1::]/64 : allow

# Sendmail can help protect you against spammers and relay-rapers
sendmail : localhost : allow
sendmail : .nice.guy.example.com : allow
sendmail : .evil.cracker.example.com : deny
sendmail : ALL : allow

# Exim is an alternative to sendmail, available in the ports tree
exim : localhost : allow
exim : .nice.guy.example.com : allow
exim : .evil.cracker.example.com : deny
exim : ALL : allow

# Portmapper is used for all RPC services; protect your NFS!
# (IP addresses rather than hostnames *MUST* be used here)
portmap : 192.0.2.32/255.255.255.224 : allow
portmap : 192.0.2.96/255.255.255.224 : allow
portmap : ALL : deny

# Provide a small amount of protection for ftpd
ftpd : localhost : allow
ftpd : .nice.guy.example.com : allow
ftpd : .evil.cracker.example.com : deny
ftpd : ALL : allow
```

```
# You need to be clever with finger; do _not_ backfinger!! You can easily
# start a "finger war".
fingerd : ALL \
  : spawn (echo Finger. | \
   /usr/bin/mail -s "tcpd\: %u@%h[%a] fingered me!" root) & \
  : deny

# The rest of the daemons are protected.
ALL : ALL \
  : severity auth.info \
  : twist /bin/echo "You are not welcome to use %d from %h."
```

Директивы файла hosts.allow имеют вид

```
Программа:Клиенты:Действие
```

Допускается слово ALL в двух первых разделах, что подразумевает все программы и любые хосты соответственно. Действие может быть обозначено ключевыми словами `allow` и `deny` или вызовом определенной программы. Более полную информацию можно получить, обратившись к системному руководству по host_access(5).

Практика предоставления сетевых сервисов только посредством супердемона inetd не всегда оправданна. В случаях, когда определенный сервис используется интенсивно, например, на специализированном FTP-сервере, использование супердемона inetd приводит к снижению эффективности работы сервера. В таких случаях лучше использовать демон ftp непосредственно.

Разрешение имен

Компьютеры находят друг друга по адресам, но людям гораздо легче запоминать имена. Для преобразования имен в адреса и обратно в Интернете поддерживается специальная служба Domain Name Service — DNS. Имена компьютеров формируются по правилам, которые никак не связаны с правилами формирования и распределения адресов. *Система имен* — это независимое от системы адресов иерархически организованное пространство (рис. 25.7).

Оно обслуживается множеством серверов DNS, хранящих записи имен в базах данных. Если сервер не находит нужное имя в своей базе, он передает запрос серверу более высокого уровня. Тот, в свою очередь, пересылает запрос либо подчиненному серверу, либо серверу более высокого уровня. Сервер, в ведении которого находится нужный домен, передает запрос вниз. Запрос передается серверами до тех пор, пока имя не будет найдено, либо не будет получен отрицательный ответ от авторитетного сервера об отсутст-

вии такого имени во вверенном ему домене. Все многообразие имен обеспечивается трех-четырехкомпонентными именами. Каждый сервер может передать другому (делегировать) ведение некоторой части той иерархии, за которую он отвечает. Несмотря на колоссальное количество и разбросанность серверов, имена разрешаются мгновенно благодаря продуманной схеме устройства пространства и отлично отработанной системе ведения записей в базах данных.

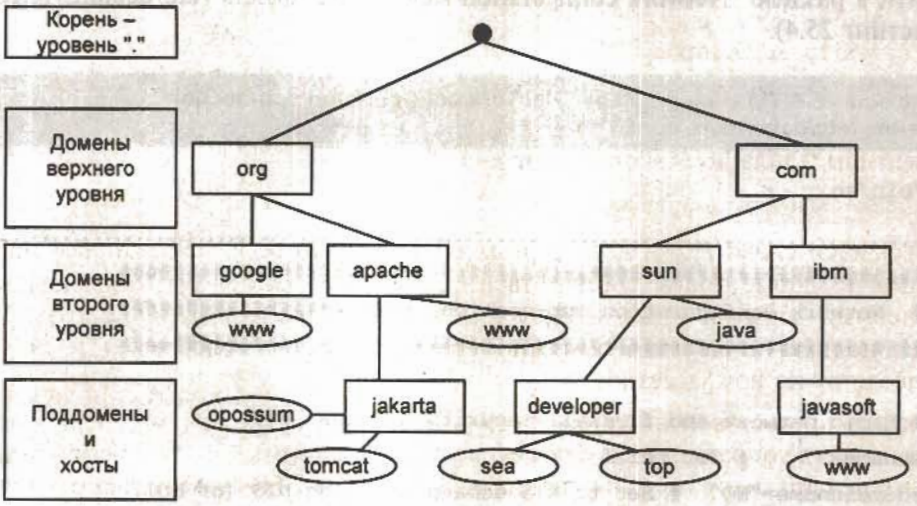


Рис. 25.7. Иерархия доменных имен

Каждый компьютер, пользующийся услугами DNS, является либо клиентом этой системы, либо клиентом и сервером одновременно. Для преобразования имен компьютеры вызывают подпрограмму `gethostbyname()` (это библиотечная подпрограмма, а не утилита), которая, анализируя конфигурацию системы, запрашивает адрес у сервера имен.

Информация о серверах имен для клиентских машин записывается в файле /etc/resolv.conf. Кроме адресов используемых серверов DNS, в нем записываются имена автоматически добавляемых доменов.

Файл /etc/hosts содержит сведения об именах компьютеров, записанные в простой форме. Иногда разумно применять его для хранения имен наиболее часто используемых соседей. Имя собственного компьютера можно установить утилитой hostname:

```
# hostname somehost.some.domain
```

или определить в параметрах стартовых скриптов (script — сценарий).

Сетевые настройки стартовых скриптов

Сетевые сервисы, как и другие системные службы, запускаются комплексом стартовых скриптов. Все настройки сети — специфические для данного компьютера величины — записываются в файл /etc/rc.conf в виде пар *имя_параметра=значение*. Параметры могут быть логической константой ("YES" или "NO") или строкой, определяющей, например, имя компьютера. Полный перечень всех параметров, относящихся к настройкам сети, можно найти в разделе "Network configuration sub-section" файла /etc/defaults/rc.conf (листинг 25.4).

Листинг 25.4. Файл /etc/defaults/rc.conf

```
#!/bin/sh

.....
##################################################################
###  Network configuration sub-section  ########################
##################################################################

### Basic network and firewall/security options: ###
hostname=""    # Set this!
nisdomainname="NO"  # Set to NIS domain if using NIS (or NO).
dhcp_program="/sbin/dhclient"  # Path to dhcp client program.
dhcp_flags=""    # Additional flags to pass to dhcp client.
firewall_enable="NO"  # Set to YES to enable firewall functionality
firewall_script="/etc/rc.firewall" # Which script to run to set up the
firewall
firewall_type="UNKNOWN"  # Firewall type (see /etc/rc.firewall)
firewall_quiet="NO"  # Set to YES to suppress rule display
firewall_logging="NO"          # Set to YES to enable events logging
firewall_flags=""  # Flags passed to ipfw when type is a file
ip_portrange_first="NO"  # Set first dynamically allocated port
ip_portrange_last="NO"  # Set last dynamically allocated port
ipsec_enable="NO"  # Set to YES to run setkey on ipsec_file
ipsec_file="/etc/ipsec.conf"  # Name of config file for setkey
natd_program="/sbin/natd"  # path to natd, if you want a different one.
natd_enable="NO"           # Enable natd (if firewall_enable == YES).
natd_interface="fxp0"          # Public interface or IPaddress to use.
natd_flags=""                  # Additional flags for natd.
```

```
ipfilter_enable="NO"  # Set to YES to enable ipfilter functionality
ipfilter_program="/sbin/ipf -Fa -f"
    # program and how to specify the rules file,
    # see /etc/rc.network (pass1) for details
ipfilter_rules="/etc/ipf.rules"  # rules definition file for ipfilter,
see
    # /usr/src/contrib/ipfilter/rules for examples
ipfilter_flags="-E"  # should be *empty* when ipf is _not_ a module
    # (i.e. compiled into the kernel) to
    # avoid a warning about "already initialized"
ipnat_enable="NO"  # Set to YES for ipnat; needs ipfilter, too!
ipnat_program="/sbin/ipnat -CF -f" # program and how to specify rules
↳file
ipnat_rules="/etc/ipnat.rules"  # rules definition file for ipnat
ipnat_flags=""    # additional flags for ipnat
ipmon_enable="NO"  # Set to YES for ipmon; needs ipfilter, too!
ipmon_program="/sbin/ipmon"  # where the ipfilter monitor program lives
ipmon_flags="-Ds"  # typically "-Ds" or "-D /var/log/ipflog"
tcp_extensions="YES"  # Set to NO to turn off RFC1323 extensions.
log_in_vain="NO"  # YES to log connects to ports w/o listeners.
tcp_keepalive="YES"  # Enable stale TCP connection timeout (or NO).
# For the following option, you need to build your kernel with the
# TCP_DROP_SYNFIN option set.  Please refer to LINT for details.
tcp_drop_synfin="NO"  # Set to YES to drop TCP packets with SYN+FIN
    # NOTE: this violates the TCP specification
icmp_drop_redirect="NO"  # Set to YES to ignore ICMP REDIRECT packets
icmp_log_redirect="NO"  # Set to YES to log ICMP REDIRECT packets
network_interfaces="auto"  # List of network interfaces (or "auto").
ifconfig_lo0="inet 127.0.0.1"  # default loopback device configuration.
#ifconfig_lo0_alias0="inet 127.0.0.254 netmask 0xffffffff" # Sample alias
↳entry.
#ifconfig_ed0_ipx="ipx 0x00010010"  # Sample IPX address family entry.
#
# If you have any sppp(4) interfaces above, you might also want to set
# the following parameters.  Refer to spppcontrol(8) for their meaning.
sppp_interfaces=""  # List of sppp interfaces.
#sppp_interfaces="isp0"  # example: sppp over ISDN
#spppconfig_isp0="authproto=chap myauthname=foo myauthsecret='top secret'
hisauthname=some-gw hisauthsecret='another secret'"
gif_interfaces="NO"  # List of GIF tunnels (or "NO").
```

```
#gif_interfaces="gif0 gif1"  # Examples typically for a router.
    # Choose correct tunnel addrs.
#gifconfig_gif0="10.1.1.1 10.1.2.1"  # Examples typically for a router.
#gifconfig_gif1="10.1.1.2 10.1.2.2"  # Examples typically for a router.

# User ppp configuration.
ppp_enable="NO"  # Start user-ppp (or NO).
ppp_mode="auto"  # Choice of "auto", "ddial", "direct" or "dedicated".
    # For details see man page for ppp(8). Default is auto.
ppp_nat="YES"  # Use PPP's internal network address translation or NO.
ppp_profile="papchap"  # Which profile to use from /etc/ppp/ppp.conf.
ppp_user="root"  # Which user to run ppp as

### Network daemon (miscellaneous) & NFS options: ###
syslogd_enable="YES"  # Run syslog daemon (or NO).
syslogd_program="/usr/sbin/syslogd" # path to syslogd, if you want a dif-
ferent one.
syslogd_flags="-s"  # Flags to syslogd (if enabled).
inetd_enable="YES"  # Run the network daemon dispatcher (or NO).
inetd_program="/usr/sbin/inetd"  # path to inetd, if you want a different
↳one.
inetd_flags="-wW"  # Optional flags to inetd
#
# named.  It may be possible to run named in a sandbox, man security for
# details.
#
named_enable="NO"  # Run named, the DNS server (or NO).
named_program="named"  # path to named, if you want a different one.
named_flags=""    # Flags for named
#named_flags="-u bind -g bind"  # Flags for named

kerberos_server_enable="NO"  # Run a kerberos master server (or NO).
kadmind_server_enable="NO"  # Run kadmind (or NO) -- do not run on
    # a slave kerberos server
kerberos5_server_enable="NO"  # Run a kerberos 5 master server (or NO).
kerberos5_server="/usr/libexec/kdc"  # path to kerberos 5 KDC
kadmind5_server_enable="NO"  # Run kadmind (or NO)
kadmind5_server="/usr/libexec/k5admind"  # path to kerberos 5 admin
↳daemon
```

```
kerberos_stash=""  # Is the kerberos master key stashed?
rwhod_enable="NO"  # Run the rwho daemon (or NO).
rwhod_flags=""     # Flags for rwhod
amd_enable="NO"    # Run amd service with $amd_flags (or NO).
amd_flags="-a /.amd_mnt -l syslog /host /etc/amd.map /net /etc/amd.map"
amd_map_program="NO"  # Can be set to "ypcat -k amd.master"
nfs_client_enable="NO"  # This host is an NFS client (or NO).
nfs_client_flags="-n 4"  # Flags to nfsiod (if enabled).
nfs_access_cache="2"  # Client cache timeout in seconds
nfs_server_enable="NO"  # This host is an NFS server (or NO).
nfs_server_flags="-u -t -n 4"  # Flags to nfsd (if enabled).
single_mountd_enable="NO"       # Run mountd only (or NO).
mountd_flags="-r"  # Flags to mountd (if NFS server enabled).
weak_mountd_authentication="NO"  # Allow non-root mount requests to be
↳served.
nfs_reserved_port_only="NO"  # Provide NFS only on secure port (or NO).
nfs_bufpackets="DEFAULT"  # bufspace (in packets) for client (or DEFAULT)
rpc_lockd_enable="NO"  # Run NFS rpc.lockd (*broken!*) if nfs_server.
rpc_statd_enable="YES"  # Run NFS rpc.statd if nfs_server (or NO).
portmap_enable="NO"  # Run the portmapper service (or NO).
portmap_program="/usr/sbin/portmap"  # path to portmap, if you want a
↳different one.
portmap_flags=""  # Flags to portmap (if enabled).
rpc_ypupdated_enable="NO"  # Run if NIS master and SecureRPC (or NO).
keyserv_enable="NO"  # Run the SecureRPC keyserver (or NO).
keyserv_flags=""  # Flags to keyserv (if enabled).
rarpd_enable="NO"  # Run rarpd (or NO).
rarpd_flags=""     # Flags to rarpd.
xtend_enable="NO"  # Run the X-10 power controller daemon.
xtend_flags=""     # Flags to xtend (if enabled).
pppoed_enable="NO"  # Run the PPP over Ethernet daemon.
pppoed_provider="*"  # Provider and ppp(8) config file entry.
pppoed_flags="-P /var/run/pppoed.pid"  # Flags to pppoed (if enabled).
pppoed_interface="fxp0"  # The interface that pppoed runs on.
sshd_enable="NO"                # Enable sshd
sshd_program="/usr/sbin/sshd"  # path to sshd, if you want a different
↳one.
sshd_flags=""  # Additional flags for sshd.
```

```
### Network Time Services options: ###
timed_enable="NO"  # Run the time daemon (or NO).
timed_flags=""    # Flags to timed (if enabled).
ntpdate_enable="NO"  # Run ntpdate to sync time on boot (or NO).
ntpdate_program="ntpdate"  # path to ntpdate, if you want a different
↳one.
ntpdate_flags=""  # Flags to ntpdate (if enabled).
xntpd_enable="NO"  # Run ntpd Network Time Protocol (or NO).
xntpd_program="ntpd"  # path to ntpd, if you want a different one.
xntpd_flags="-p /var/run/ntpd.pid"  # Flags to ntpd (if enabled).

# Network Information Services (NIS) options: ###
nis_client_enable="NO"  # We're an NIS client (or NO).
nis_client_flags=""  # Flags to ypbind (if enabled).
nis_ypset_enable="NO"  # Run ypset at boot time (or NO).
nis_ypset_flags=""  # Flags to ypset (if enabled).
nis_server_enable="NO"  # We're an NIS server (or NO).
nis_server_flags=""  # Flags to ypserv (if enabled).
nis_ypxfrd_enable="NO"  # Run rpc.ypxfrd at boot time (or NO).
nis_ypxfrd_flags=""  # Flags to rpc.ypxfrd (if enabled).
nis_yppasswdd_enable="NO"  # Run rpc.yppasswdd at boot time (or NO).
nis_yppasswdd_flags=""  # Flags to rpc.yppasswdd (if enabled).

### Network routing options: ###
defaultrouter="NO"  # Set to default gateway (or NO).
static_routes=""  # Set to static route list (or leave empty).
gateway_enable="NO"  # Set to YES if this host will be a gateway.
router_enable="NO"  # Set to YES to enable a routing daemon.
router="routed"    # Name of routing daemon to use if enabled.
router_flags="-q"  # Flags for routing daemon.
mrouted_enable="NO"  # Do multicast routing (see /etc/mrouted.conf).
mrouted_flags=""  # Flags for multicast routing daemon.
ipxgateway_enable="NO"  # Set to YES to enable IPX routing.
ipxrouted_enable="NO"  # Set to YES to run the IPX routing daemon.
ipxrouted_flags=""  # Flags for IPX routing daemon.
arpproxy_all="NO"   # replaces obsolete kernel option ARP_PROXYALL.
forward_sourceroute="NO"  # do source routing (only if gateway_enable is
↳set to "YES")
accept_sourceroute="NO"  # accept source routed packets to us
```

```
### ATM interface options: ###
atm_enable="NO"     # Configure ATM interfaces (or NO).
#atm_netif_hea0="atm 1"  # Network interfaces for physical interface.
#atm_sigmgr_hea0="uni31"  # Signalling manager for physical interface.
#atm_prefix_hea0="ILMI"  # NSAP prefix (UNI interfaces only) (or ILMI).
#atm_macaddr_hea0="NO"  # Override physical MAC address (or NO).
#atm_arpserver_atm0="0x47.0005.80.999999.9999.9999.9999.999999999999.00"
# ATMARP server address (or local).
#atm_scsparp_atm0="NO"  # Run SCSP/ATMARP on network interface (or NO).
atm_pvcs=""     # Set to PVC list (or leave empty).
atm_arps=""     # Set to permanent ARP list (or leave empty).

### ISDN interface options: ###
isdn_enable="NO"  # Enable the ISDN subsystem (or NO).
isdn_fsdev="/dev/ttyv4"  # Output device for fullscreen mode (or NO for
↳daemon mode).
isdn_flags="-dn -d0x1f9"  # Flags for isdnd
isdn_ttype="cons25"  # terminal type for fullscreen mode
isdn_screenflags="NO"  # screenflags for ${isdn_fsdev}
isdn_trace="NO"     # Enable the ISDN trace subsystem (or NO).
isdn_traceflags="-f /var/tmp/isdntrace0"  # Flags for isdntrace

### Miscellaneous network options: ###
icmp_bmcastecho="NO"  # respond to broadcast ping packets
.....
```

Здесь же можно найти еще почти столько же параметров для протокола IPv6. Полное описание структуры файла rc.conf и разъяснение смысла всех его параметров можно найти в системном руководстве по файлу rc.conf.

Старт сетевых сервисов выполняется скриптом /etc/netstart, который считывает все назначения параметров как назначения переменных окружения:

```
if [ -f /etc/defaults/rc.conf ]; then
  . /etc/defaults/rc.conf
  source_rc_confs
elif [ -f /etc/rc.conf ]; then
  . /etc/rc.conf
fi
.....
```

Сначала считывается файл /etc/defaults/rc.conf, а затем функция source_rc_confs читает файлы /etc/rc.conf и /etc/rc.conf.local. Назначения па-

раметров, сделанные в этих файлах, заменят первоначальные значения. Вносить изменения в /etc/defaults/rc.conf не нужно: все настройки, отличные от данных по умолчанию, должны записываться в файл /etc/rc.conf:

```
# -- sysinstall generated deltas -- # Fri Nov  8 09:01:15 2002
# -- sysinstall generated deltas -- # Fri Nov  8 09:13:23 2002
# -- sysinstall generated deltas -- # Fri Nov  8 11:01:06 2002
# Created: Fri Nov  8 09:01:15 2002
# Enable network daemons for user convenience.
# Please make all changes to this file, not to /etc/defaults/rc.conf.
# This file now contains just the overrides from /etc/defaults/rc.conf.
kern_securelevel_enable="NO"
nfs_reserved_port_only="YES"
sendmail_enable="YES"
sshd_enable="YES"
usbd_enable="YES"
# -- sysinstall generated deltas -- # Fri Nov  8 17:25:14 2002
ifconfig_fxp0="inet 192.168.0.2  netmask 255.255.255.0"
moused_enable="YES"
defaultrouter="192.168.0.1"
portmap_enable="YES"
hostname="myhome.my.domain"
```

Если ограничить перенастройку системы только внесением изменений в файл /etc/rc.conf, изменения вступят в силу только после перезагрузки. Если изменить настройки командами, ничего не изменяя в /etc/rc.conf, новые настройки после перезагрузки пропадут.

Инсталляционная программа /stand/sysinstall записывает в этот файл внесенные пользователем изменения (конечно, только при наличии у пользователя необходимых прав). Кроме того, эта программа выполняет также все действия по обновлению настроек без перезагрузки, что очень удобно для не очень опытных пользователей. Однако сложные задачи, например, конфигурирование IP-псевдонима, с ее помощью выполнить нельзя.

Утилиты проверки работоспособности сети

*Простые утилиты ping, traceroute, netstat, tcpdump помогут найти неисправ*ности в настройках сети или сетевых установках компьютера. Эти программы не являются частью FreeBSD или какой-либо другой операционной системы. Все они были написаны отдельно и получили распространение

благодаря своей исключительной полезности. Они распространяются вместе с прикладной частью дистрибутивов всех UNIX, а их аналоги, пусть и в урезанном виде, работают во многих других ОС, поддерживающих TCP/IP.

Утилита ping посылает по заданному адресу ICMP-пакет "echo-request", на который получатель отвечает ICMP-пакетом "echo-reply". Каждый полученный ответ утилита ping выводит в виде строки, где указана длина пакета, IP-адрес заданного компьютера, порядковый номер пакета и время, затраченное на прохождение пакета туда и обратно. Без опции `-c` в командной строке программа образует бесконечный цикл, из которого выходят по одновременному нажатию клавиш <Crtl>+<C>. Посылка прекращается, а утилита ping сообщает число переданных и утраченных пакетов, подсчитывая процент потерь. С помощью этой утилиты можно не только выяснить, работает ли машина, которую тестируют утилитой ping, но и определить качество соединения, степень загруженности сети, готовность сети к экстремальным нагрузкам. С опцией `-R` можно получить некоторую информацию о маршруте пакета, но для более детального исследования пути пакета обычно используется программа traceroute.

Утилита traceroute выводит информацию о шлюзах, через которые IP-пакет проходит по пути к пункту своего назначения (листинг 25.5). Она посылает последовательность пакетов, время жизни которых позволяет им пройти от одного до 30 переходов. Маршрутизаторы уведомляют компьютер-отправитель об истечении времени жизни их пакета, сообщая о себе и трассируя путь IP-посылки к заданному адресу.

Листинг 25.5. Пример применения утилиты traceroute

```
$traceroute -n www.spb.ru
traceroute to www.spb.ru (193.125.192.167), 30 hops max, 38 byte packets
 1  192.168.103.97  0.187 ms  0.147 ms  0.140 ms
 2  196.16.197.71  1.156 ms  1.201 ms  1.326 ms
 3  196.16.94.68  6.115 ms  1.643 ms  1.788 ms
 4  194.19.255.165  3.114 ms  10.583 ms  6.851 ms
 5  194.85.36.42  37.656 ms  33.273 ms  31.410 ms
 6  194.85.177.35  23.965 ms  22.356 ms  17.916 ms
 7  193.125.15.129  13.455 ms  22.139 ms  31.794 ms
 8  193.125.192.167  23.403 ms  29.283 ms  19.246 ms
```

Утилита netstat — универсальная утилита проверки состояния всех аспектов работы в сети компьютера, на котором она выполняется. В этой главе уже рассматривалось применение этой утилиты для чтения таблицы маршрутизации ядра (`netstat -r [-n]`). Кроме этого, netstat можно использовать для

получения информации о конфигурации интерфейсов (`netstat -i [-n]`), проверки состояния сетевых соединений (`netstat -a [-n]`) и получения статических данных о различных сетевых протоколах (`netstat -s`). Утилита netstat без аргументов выдает информацию об активных портах протоколов UDP и TCP и доменных сокетах UNIX.

Чтобы получить информацию обо всех портах, нужно указывать опцию `-a`.

В графах `Recv-Q` и `Send-Q` показываются размеры очередей приема и передачи для установленного через порт соединения. Для протокола TCP указывается состояние соединения, чаще всего встречаются состояния ESTABLISHED (установлено), LISTENING (прослушивание) и TIME_WAIT (ожидание закрытия).

При получении информации о сетевых интерфейсах, `netstat -i` показывает количество исходящих и приходящих пакетов, количество ошибок в тех и других и число произошедших коллизий.

```
# netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR   TX-OK TX-ERR TX-DRP TX-OVR
Flg
eth0   1500   0  423843      0      0      1   90721      0      0      0
BMRU
eth1   1500   0       0      0      0      0       4      0      0      0
BMRU
lo    16436   0     122      0      0      0     122      0      0      0
LRU
```

Для получения статической информации по интерфейсам утилите netstat нужно указать интервал выдачи отчетов (в секундах):

```
#netstat -w 10
```

История сетевых соединений с момента перезагрузки отражается в счетчиках пакетов, разбросанных по программам. Утилита netstat с опцией `-s` выводит содержимое счетчиков пакетов и группирует их по протоколам (листинг 25.6).

Листинг 25.6. Пример выполнения утилиты netstat с опцией -s

```
# netstat -s
Ip:
    5125738 total packets received
    0 forwarded
    0 incoming packets discarded
    4056932 incoming packets delivered
```

```
    3977773 requests sent out
    2 fragments dropped after timeout
    35 reassemblies required
    15 packets reassembled ok
    2 packet reassembles failed
Icmp:
    5313 ICMP messages received
    34 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 991
        timeout in transit: 2524
        source quenches: 507
        echo requests: 1262
        echo replies: 17
        timestamp request: 3
        address mask request: 3
    5375 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 4109
        time exceeded: 1
        echo replies: 1262
        timestamp replies: 3
Tcp:
    25412 active connections openings
    0 passive connection openings
    11 failed connection attempts
    0 connection resets received
    15 connections established
    3766798 segments received
    3968179 segments send out
    92079 segments retransmited
    177 bad segments received.
    11569 resets sent
Udp:
    3778 packets received
    4073 packets to unknown port received.
    0 packet receive errors
    3882 packets sent
```

```
TcpExt:
    4 resets received for embryonic SYN_RECV sockets
    154 ICMP packets dropped because they were out-of-window
    2 ICMP packets dropped because socket was locked
    ArpFilter: 0
    180353 TCP sockets finished time wait in fast timer
    399 packets rejects in established connections because of timestamp
    28637 delayed acks sent
    651 delayed acks further delayed because of locked socket
    Quick ack mode was activated 5944 times
    1100832 packets directly queued to recvmsg prequeue.
    68565 packets directly received from backlog
    144538066 packets directly received from prequeue
    269209 packets header predicted
    362105 packets header predicted and directly queued to user
.....................................................................
```

Информацию о движении пакетов в сети в настоящий момент можно получить с помощью программы tcpdump.

Утилита tcpdump следит за трафиком в сети, регистрируя и выводя на экран заголовки всех пакетов, соответствующих заданным условиям. Регистрируются в том числе и те пакеты, для которых хост-хозяин не является ни адресом назначения, ни адресом отправления. Чтобы программа могла выполняться, ядро ОС FreeBSD должно поддерживать прием всех пакетов физического сегмента сети. Это свойство обусловлено поддержкой псевдоустройства /dev/bpf. В конфигурационном файле ядра должна присутствовать строка:

```
pseudo-device  bpf  #Berkeley packet filter
```

Вызванная без аргументов, утилита tcpdump регистрирует все пакеты на наиболее используемом интерфейсе (листинг 25.7).

Листинг 25.7. Пример выполнения утилиты tcpdump без аргументов

```
#tcpdump
tcpdump: listening on eth0
19:42:12.717336 night.mare.any.org.ssh > 195.5.130.127.998: P
1426326995:1426327047(52) ack 322136 win 6432 (DF) [tos 0x10]
19:42:12.719237 night.mare.any.org.32836 > ns.any.org.domain:  54060+
PTR? 146.203.19.195.in-addr.arpa. (45) (DF)
```

```
19:42:12.721506 ns.any.org.domain > night.mare.any.org.32836:  54060*
1/2/2 (143) (DF)
19:42:12.722004 night.mare.any.org.32836 > ns.any.org.domain:  54061+
PTR? 66.194.19.195.in-addr.arpa. (44) (DF)
19:42:12.724071 ns.any.org.domain > night.mare.any.org.32836:  54061*
1/3/3 (174) (DF)
19:42:12.738386 195.5.130.127.998 > night.mare.any.org.ssh: . ack 52 win
8144 (DF)
19:42:12.898667 195.5.130.127.998 > night.mare.any.org.ssh: . ack 52 win
8144 (DF)
19:42:13.794885 195.5.130.127.998 > night.mare.any.org.ssh: P 1:21(20)
ack 52 win 8144 (DF)
19:42:13.797394 night.mare.any.org.ssh > 195.5.130.127.998: P 52:72(20)
ack 21 win 6432 (DF) [tos 0x10]
19:42:14.260226 195.5.130.127.998 > night.mare.any.org.ssh: P 21:41(20)
ack 72 win 8124 (DF)
```

Общий формат использования утилиты tcpdump:

```
tcpdump [ -adeflnNOpqRStvxX] [ -c count ] [ -F file ]
        [ -i interface ] [ -m module ] [ -r file ]
        [ -s snaplen ] [ -T type ] [ -w file ]
        [ -E algo:secret ] [ expression ]
```

Выражение — `expression` — позволяет ограничить регистрацию пакетов адресами источников или назначения, номерами портов, названиями используемых протоколов, параметрами уровня сетевого доступа. Отдельные выражения можно комбинировать с помощью логических операций `and`, `or` и `not`.

Глава 26

Маршрутизация и защита сети

Маршрутизация

Каким образом IP-пакеты находят дорогу к адресату? Как защитить локальную сеть от нежелательных посещений? Как десятки компьютеров могут пользоваться одним IP-адресом? Все эти задачи решаются, и ОС FreeBSD располагает всем необходимым для их решения.

Статическая и динамическая маршрутизация

Компьютер, имеющий выходы в разные сети, может пересылать данные из одной сети в другую. Процесс пересылки данных между сетями называют *маршрутизацией*, а устройство, выполняющее ее, — *маршрутизатором*. Маршрутизатор, соединяющий сети различного физического типа, было принято называть *шлюзом*, теперь же различие между терминами "маршрутизатор" и "шлюз" практически исчезло. Шлюз решает задачу маршрутизации для IP-пакетов, произведенных "третьими лицами" — компьютерами, находящимися в других сетях.

В листинге 26.1 приведена таблица маршрутизации для компьютера с двумя сетевыми картами fxp0 и fxp1, по которым осуществляется выход в сети с IP-адресами 214.91.204.64/27 и 214.91.202.224/27 соответственно. Кроме записей — отсылок к интерфейсам, в ней есть записи разрешения адресов (IP-адрес — MAC-адрес) и заглушки с MAC-адресами ff:ff:ff:ff:ff:ff. Таблица маршрутизации шлюза периодически обновляется постоянно работающим демоном маршрутизации. Таблица маршрутизации, приведенная в листинге 26.1, на первый взгляд, может быть легко сформирована статическим образом, однако это не так: маршруты этой таблицы могут исчезать и появляться вновь, статическая маршрутизация в таких случаях несостоятельна.

Листинг 26.1. Пример таблицы маршрутизации

```
# netstat -rn
Routing tables

Internet:
Destination        Gateway            Flags     Refs      Use  Netif Ex-
pire
default            214.91.204.65      UGSc     17656  8579697    fxp0
127.0.0.1          127.0.0.1          UH        4336   623603     lo0
172.16.100/27      214.91.204.79      UGSc         1     4432    fxp0
214.91.202.224     ff:ff:ff:ff:ff:ff  UHLWb        0     5875    fxp1 =>
214.91.202.224/27  link#2             UC          10        0    fxp1
214.91.202.225     0:2:b5:a3:86:aa    UHLW       193   284563     lo0
214.91.202.244     link#2             UHLW         0        3    fxp1
214.91.202.255     ff:ff:ff:ff:ff:ff  UHLWb        1     4522    fxp1
214.91.204.64/27   link#1             UC           5        0    fxp0
214.91.204.65      0:d2:b7:c:2f:d4    UHLW       124    70700    fxp0
1191
214.91.204.80      link#1             UHLW         1    28956    fxp0
214.91.204.95      ff:ff:ff:ff:ff:ff  UHLWb        0        1    fxp0
214.91.205.131     214.91.204.80      UGHD         0       16    fxp0
```

В случае единственного выхода в сеть задача маршрутизации решается просто: пакеты с адресами из всех сетей, кроме локальной, направляются к шлюзу default. Задача маршрутизатора часто осложняется неоднозначностью решения, т. к. одной и той же цели можно достичь разными путями. Сеть маршрутизаторов Интернета имеет древовидную структуру по краям, там, где они близки к конечным получателям, в центре же она скорее похожа на паутину. Если в начале пути пакет просто "поднимается" к маршрутизатору следующего уровня, то когда-нибудь он должен начать "спускаться". Понятно, как найти адресата в сети, в которую у шлюза есть непосредственный выход, а как найти маршрут к сети, если до нее остается еще десять маршрутизаторов?

Таблица маршрутизации шлюза поддерживается программой — демоном маршрутизации. Демоны обмениваются между собой информацией об известных им маршрутах по специальным протоколам. Простейший из демонов маршрутизации, routed, входит в состав всех UNIX-подобных ОС. Он использует только Routing Information Protocol (RIP). Оба — и демон, и протокол — далеки от совершенства. Демон routed работает в широковещательном режиме и повышает нагрузку на сеть. Качество маршрута протокол RIP

оценивает количеством переходов; цель, находящаяся на расстоянии более чем в 16 переходов, считается недостижимой, поэтому протокол RIP непригоден для больших сетей. Кроме того, демон routed не поддается настройке и вещает обо всех известных ему маршрутах.

Другой широко распространенный демон маршрутизации — gated — кроме протокола RIP поддерживает и другие протоколы маршрутизации: OSPF (Open Shortest Path First), EGP (Exterior Gateway Protocol) и BGP (Border Gateway Protocol). Протоколы RIP и OSPF используются для внутренней маршрутизации, а EGP и BGP — для внешней.

Трансляция адресов

Соглашение о приватных адресах

В адресном пространстве сети Интернет специальным соглашением (RFC 1918) выделены группы адресов для приватных сетей. Область видимости приватной сети ограничена ближайшим выпускающим шлюзом, маршруты к приватным сетям никогда не объявляются (не анонсируются) маршрутизаторами. Таким образом, за каждым маршрутизатором можно построить сеть с миллионами компьютеров, не заняв ни одного адреса Интернета.

Приватные сети исключительно удобны для создания корпоративных сетей, т. к. информация, размещенная в них, недоступна извне. Приватная сеть — это часть сети Интернет, использующая анонсируемые адреса, о которых сеть Интернет не знает.

Но у этой медали есть и обратная сторона: не только приватные сети недоступны для внешнего мира, но и мир недоступен для них. Чтобы преодолеть это ограничение и открыть дорогу к открытым источникам информации Интернета компьютерам из корпоративных сетей, была изобретена трансляция адресов — Network Address Translation (NAT, RFC 1631). Трансляция адресов NAT, поддерживаемая маршрутизатором, переписывает (транслирует) IP-адреса источника и приемника по заданным правилам. Получается, что поток пакетов, изначально имевших адрес источника, скажем, 192.168.0.3, превращается в поток пакетов от источника с анонсируемым адресом шлюза. Ответные пакеты направляются по этому адресу, но передаются уже маршрутизатором компьютеру с адресом 192.168.0.3. Соединение устанавливается, клиент корпоративной сети получает доступ к открытым ресурсам Интернета.

В подавляющем большинстве случаев NAT настраивается именно на такую схему работы: из корпоративной сети выход в Интернет есть, а из Интернета корпоративная сеть недоступна, но схемы могут быть разными.

ОС FreeBSD располагает двумя альтернативными средствами для организации трансляции адресов: демон natd, использующий divert-сокеты ядра

FreeBSD с пакетным фильтром IPFW, и механизм IPNAT, использующий пакетный фильтр IPFILTER ядра.

Трансляция адресов демоном natd

Демон трансляции адресов NAT — natd — принимает входящие IP-пакеты, меняет в них адрес источника и отправляет их обратно в выходной поток IP-пакетов, направленный к адресу назначения. Демон natd меняет IP-адрес источника и его порт так, чтобы можно было определить, куда направлять ответные данные, запрошенные исходным потоком. Действие демона natd основано на divert-сокетах, поддерживаемых ядром FreeBSD.

Примечание

Divert — "отклоняющий" — сокет, который игнорирует IP-адрес принимаемого пакета, поддерживая соединение только по номеру порта. Если включен механизм, направляющий пакеты к порту, связанному с divert-сокетом, то понятно, как можно "разнести" сетевые соединения от разных компьютеров по набору таких сокетов.

Организовать трансляцию адресов можно на маршрутизаторе, у которого есть, как минимум, два интерфейса: один с анонсируемым IP-адресом, и второй, присоединенный к локальной сети с приватными адресами. Тогда вся локальная сеть сможет разделить доступ к открытым ресурсам сети Интернет с "natd-шлюзом".

Для работы с демоном natd ядро системы должно быть сконфигурировано со следующими опциями:

- `options IPFIREWALL`;
- `options IPDIVERT`.

Они не присутствуют в ядре GENERIC, и для их подключения требуется перекомпиляция ядра. При желании, если можно устанавливать брандмауэр открытым при старте и требуется вести протоколирование через службу системной регистрации сообщений syslog, можно добавить еще две опции:

- `options IPFIREWALL_DEFAULT_TO_ACCEPT`;
- `options IPFIREWALL_VERBOSE`.

В файл /etc/rc.conf нужно добавить следующие параметры (табл. 26.1).

Таблица 26.1. Параметры поддержки natd

Параметр	Назначение
`gateway_enable="YES"`	Включает поддержку маршрутизации
`firewall_enable="YES"`	Включает поддержку брандмауэра при загрузке

Таблица 26.1 (окончание)

Параметр	Назначение
`firewall_type="OPEN"`	Устанавливает первоначальное состояние брандмауэра открытым (пропускать всех)
`natd_interface="fxp0"`	Указывает имя внешнего интерфейса (выходящего в сеть Интернет)
`natd_flags=""`	Указывает дополнительные опции демона natd, используемые при старте (см. natd(8))

Чтобы ввести в действие назначенные параметры, не обязательно производить перезагрузку. Поддержку маршрутизации можно включить командой

```
# sysctl net.inet.ip.forwarding=1
```

а демон natd стартовать из командной строки:

```
# natd -interface fxp0
```

Все компьютеры, находящиеся за маршрутизатором, должны иметь приватные адреса и маршрут по умолчанию default, указывающий на внутренний интерфейс шлюза.

Например, внешний интерфейс natd-шлюза — fxp0 с IP-адресом 216.19.103.15, а внутренний интерфейс ed0 имеет неанонсируемый адрес 192.168.0.1. Компьютеры-клиенты A и B из внутренней сети имеют адреса 192.168.0.2 и 192.168.0.3 и маршрут default, указывающий на 192.168.0.1. Тогда старт демона natd на интерфейсе fxp0

```
# natd -interface fxp0
```

обеспечит клиентам A и B возможность установления соединений с внешним миром — открытыми ресурсами сети Интернет.

Чтобы демон natd работал, никаких специальных модификаций его внешнего интерфейса не требуется.

Недостаток приведенной выше конфигурации демона natd заключается в недоступности его клиентов из внешней части сети Интернет. Клиенты локальной сети, обслуживаемой демоном natd, могут устанавливать исходящие соединения, но не могут принимать входящие. Это представляет проблему, если клиенты пытаются запускать интернет-сервисы. Ее можно решить, перенаправив некоторые интернет-порты natd-шлюза на компьютер-клиент из локальной сети. Образец конфигурации демона natd для открытия ресурсов приватной сети для Интернета представлен в следующем примере.

Например, на вышеупомянутом компьютере-клиенте A работает сервер Internet Relay Chat (IRC, служба поддержки переговоров сети Интернет), а

на компьютере-клиенте B — Web-сервер. Чтобы эти сервисы были доступны из внешней части сети Интернет, порты 6667 (irc) и 80 (http) natd-шлюза должны быть перенаправлены на соответствующие машины во внутренней сети. Указания о перенаправлении включаются в опции демона natd. Их синтаксис таков:

```
-redirect_port proto targetIP:targetPORT[-targetPORT]
 [aliasIP:]aliasPORT[-aliasPORT]
 [remoteIP[:remotePORT[-remotePORT]]]
```

Для примера, приведенного выше, указания должны быть такими:

```
-redirect_port tcp 192.168.0.2:6667 6667
-redirect_port tcp 192.168.0.3:80 80
```

Это перенаправит собственные порты natd-шлюза на соответствующие порты клиентских машин.

Аргумент `redirect_port` может указывать диапазон портов. Например, аргумент

```
-tcp 192.168.0.2:2000-3000 2000-3000
```

перенаправит все соединения с портами в диапазоне номеров от 2000 до 3000 к портам 2000—3000 компьютера-клиента A.

Опции можно указать непосредственно при старте демона natd, или же поместить в значение переменной `natd_flags=` в файле `/etc/rc.conf`.

Перенаправление адресов полезно, если в наличии есть несколько незанятых IP-адресов, но они должны быть приписаны одной машине. Тогда демон natd может назначить каждому клиенту из приватной сети его собственный внешний IP-адрес. При переписывании исходящих пакетов от внутренней сети демон natd присвоит им их собственный внешний адрес и перенаправит весь входящий трафик на приватный адрес клиента локальной сети. Такое перенаправление называется *статической трансляцией адресов*. Например, IP-адреса 202.14.143.1, 202.14.143.2 и 202.14.143.3 принадлежат natd-шлюзу. IP-адрес 202.14.143.1 может использоваться как внешний адрес natd-шлюза, тогда как IP-адреса 202.14.143.2 и 202.14.143.3 прикрепляются клиентам A и B во внутренней сети. В этом случае опция `-redirect` записывается в следующем формате:

```
-redirect_address localIP publicIP
```

где аргументы имеют значения:

- `localIP` — внутренний IP-адрес клиента;
- `publicIP` — внешний IP-адрес, назначенный клиенту.

Для нашего примера аргументы должны выглядеть так:

```
-redirect_address 192.168.0.2 202.14.143.2
-redirect_address 192.168.0.3 202.14.143.3
```

Как и в случае перенаправления портов, опция `-redirect_address` может быть помещена в переменную `natd_flags` файла /etc/rc.conf. При перенаправлении адресов нет необходимости делать перенаправление для портов, т. к. все данные для внешнего IP-адреса перенаправляются клиенту. Внешний перенаправляемый адрес на natd-шлюзе должен находиться в активном состоянии и быть псевдонимом для его внешнего интерфейса.

IPNAT — альтернативный механизм трансляции адресов

Второй способ организации трансляции адресов основан на применении альтернативного механизма фильтрации пакетов, IPFILTER, также встраиваемом в ядро FreeBSD. Для включения IP-фильтра также необходимо перекомпилировать ядро, установив следующие опции:

- `options IPFILTER`;
- `options IPFILTER_LOG`.

В этом случае для настройки трансляции адресов используется утилита ipnat. Правила обращения с пакетами записываются в таблицу ядра из указанного файла или со стандартного ввода, разбираются и, если их синтаксис верен, моментально принимаются к исполнению. Синтаксис правил трансляции адресов NAT с IP-фильтром гораздо сложнее синтаксиса, используемого демоном трансляции адресов. Каждое правило разделяется знаком "->" на две части. Левая часть используется для отождествления пакета, над которым планируется производить действия, правая часть описывает адрес, который пакет получит после произведения действия. Первое слово правила указывает на действие, которое должно быть произведено над соответствующим пакетом. Например,

```
map de0 10.1.0.0/16 -> 201.2.3.4/32
```

или так

```
map de0 from 10.1.0.0/16 to any -> 201.2.3.4/32
```

Сравнивать можно только IP-адреса и номера портов. Действий может быть четыре (табл. 26.2).

Таблица 26.2. Директивы действия IPNAT

Указатель действия	Назначение
map	Используется для привязки адреса или сети к другой по нерегулируемой круговой схеме
rdr	Используется для перенаправления пакета с одной пары "порт-адрес" на другую
bimap	Назначает двунаправленную трансляцию адресов между внешним и внутренним IP-адресами
map-block	Устанавливает статическую трансляцию IP, основанную на алгоритме сопоставления транслируемых адресов диапазону адресов назначения

Для реализации задачи трансляции трафика приватной сети в Интернет посредством механизма трансляции адресов IPNAT достаточно внести следующий набор правил в файл /etc/ipnat.rules:

```
map de0 10.1.0.0/16 -> 0/32 portmap tcp/udp 10000:65000
map de0 10.1.0.0/16 -> 0/32
```

и с помощью утилиты ipnat внести их в таблицу трансляции адресов ядра:

```
#/sbin/ipnat -f </etc/ipnat.rules
```

Брандмауэры

Локальные сети не всегда базируются на приватных адресах. Многие организации успели получить в свое распоряжение блоки анонсируемых сетей и построить корпоративные по своей сути сети на маршрутизируемых адресах. Кроме того, разумно ограничить доступ к открытым сервисам: на Web-сервер, например, порт 80 (http) должен быть доступен со стороны внешних сетей, а вот порт 23 (telnet) открывать на нем для всей сети Интернет совсем не обязательно. В подобных ситуациях часто встает вопрос об ограничении доступа из внешнего мира к внутренней сети. Для этого используется механизм, называемый "firewall" (брандмауэр, пожарная стена). Термин брандмауэр (firewall) применяется и к устройствам защиты сети на аппаратном уровне, и к программным средствам защиты разного принципа действия. Брандмауэр ОС FreeBSD подразумевает регулирование установки сетевых соединений посредством фильтрации IP-пакетов на уровне ядра. Использование *IP-фильтров на входном маршрутизаторе* анонсируемой локальной сети способно поднять уровень ее защищенности до уровня защищенности приватной сети без трансляции адресов.

Организация брандмауэра, как это уже можно было заметить, была необходима для трансляции адресов. Как уже отмечалось ранее, ОС FreeBSD предлагает два различных механизма пакетной фильтрации: IPFW и IPFILTER.

Брандмауэр IPFW является традиционной частью программного обеспечения, поставляемого с ОС FreeBSD. Это система фильтрации и учета пакетов, поддерживаемая ядром FreeBSD, снабженная управляющей утилитой ipfw(8).

Для включения механизма фильтрации IP-пакетов IPFW ядро должно быть скомпилировано со следующими опциями:

- `options IPFIREWALL;`
- `options IPFIREWALL_VERBOSE;`
- `options IPFIREWALL_VERBOSE_LIMIT=10.`

Вторая и третья опции включают возможность протоколирования проходящих пакетов и лимитируют объем получаемого log-файла. Если переменные файла /etc/rc.conf будут иметь значения `firewall_enable="YES"` и `firewall_type="OPEN"`, то брандмауэр IPFW включится после ближайшей перезагрузки. Очень важен тип брандмауэра. Значение `"OPEN"` отключает фильтрацию пакетов, и после рестарта система будет поддерживать все соединения. Значения `"CLOSED"` или `"UNKNOWN"` приведут к запрету всех соединений, полностью отключив компьютер от сети. Для включения фильтров нужно определить правила, по которым пакеты будут или не будут пропускаться ядром. Набор правил разбирается следующим образом: первое из прочитанных правил, подходящее для данного пакета, выполняется, после чего чтение правил прекращается. Действия, производимые над пакетом при фильтрации, сводятся к ликвидации пакета, передаче пакета и даже посылке ICMP-сообщения производителю пакета. Учет пакетов позволяет следить за трафиком, производимым клиентскими машинами сети.

Правила фильтрации пакетов строятся в зависимости от IP-адресов источника и назначения, и номеров портов, а также, от типа пакета (UDP, TCP, ICMP и т. д.).

Синтаксис правил брандмауэра IPFW весьма замысловат с виду, но структура его относительно проста. Пакет можно пропустить к месту назначения (`allow`), запретить пропускать (`deny`), отклонить (`reject` — запрет с посылкой отправителю ICMP-уведомления `"host/port unreachable"`) и подсчитать (`count`). Для идентификации и определения порядкового номера правила используется `index` — целое число от 1 до 65 534. Порядок следования правил важен, т. к. чтение таблицы правил прекращается после первого совпадения параметров пакета с указанным диапазоном параметров правила.

Для обновления таблицы правил фильтрации используется утилита ipfw в следующем формате:

```
ipfw [-N] command [index] [new_rule]
```

Единственная опция `-N` превращает адреса и номера портов в имена хостов и сервисов. Параметр `command` означает действие, которое нужно произвести, `new_rule` — текст нового правила, `index` — номер, под которым правило помещается или уже помещено в таблицу правил фильтрации. Действий может быть всего четыре (табл. 26.3).

Таблица 26.3. Команды утилиты ipfw

Команда	Описание
`add`	Добавляет новое правило, текст которого должен быть приведен
`delete`	Удаляет указанное индексом правило
`flush`	Удаляет все записи в таблице
`list`	Выводит на экран все записи или указанную индексом запись
`clear`	Обнуляет счетчики всех протоколирующих правил или указанного индексом правила

Содержание нового правила `new_rule` определяется следующим образом:

```
action [log] protocol addresses [options]
```

Индекс `index`, предшествующий правилу, число от 1 до 65 534, указывает на место в таблице фильтрации, куда правило будет помещаться или где оно уже размещено. Если индекс после команды `add` не указан, новое правило будет помещено после последнего правила с автоматически вычисляемым индексом. Индекс 65 535 всегда занят персистентным (неудаляемым) правилом, завершающим свод правил брандмауэра IPFW. Обычно это правило `"deny all"`.

Параметр `action` может иметь четыре значения (табл. 26.4).

Таблица 26.4. Директивы действия правил IPFW

Значение параметра	Действие
`reject`	Ликвидирует пакет, отослав отправителю ICMP-сообщение `"host/port unreachable"`
`allow`	Пропускает пакет по назначению
`deny`	Ликвидирует пакет без сообщений
`count`	Обновляет значение счетчика пакетов, не выполняя операций `allow` или `deny` для пакетов из этого правила

Параметр `protocol` может принимать значения:

- `icmp` — ICMP-пакет;
- `tcp` — TCP-пакет;
- `udp` — UDP-пакет.

Параметр `addresses` указывается следующим образом:

```
from address/mask [port] to address/mask [port] [via interface]
```

Порт можно указывать только для протоколов TCP или UDP. Адрес `address/mask` может быть указан в виде `address`, или `address/prefix_length`, или `address:mask-pattern`.

Например:

```
192.216.222.1 или 192.216.222.1/24 или 192.216.222.1:255.255.255.0
```

Ключевое слово `any` соответствует любому адресу. Порты перечисляются либо через запятую: `port [,port [,port [...]]]`, либо указывается диапазон: `port-port`.

Опции `options` могут принимать значения, перечисленные в табл. 26.5.

***Таблица 26.5.** Опции правил IPFW*

Опция	Пакеты, соответствующие опции
`Frag`	Соответствует пакетам, не являющимся первыми фрагментами дейтаграммы
`In`	Соответствует входящим пакетам
`Out`	Соответствует исходящим пакетам
`ipoptions spec`	Соответствует пакетам, IP-заголовки которых содержат разделенный запятыми список опций, указанных в параметре `spec`. Допустимыми значениями являются: `ssrr` (`strict source route`), `lsrr` (`loose source route`), `rr` (`record packet route`) и `ts` (`time stamp`). Предшествующий значению знак ! означает отсутствие указанной опции
`Established`	Соответствует пакетам, являющимся частью уже установленного TCP-соединения (т. е. имеющим установленные биты RST (Restart, перезагрузка соединения) или ACK (Acknowledgement, подтверждение приема)). Можно оптимизировать производительность брандмауэра, помещая правила `established` в начале цепи
`Setup`	Соответствует пакетам, пытающимся установить TCP-соединение (бит SYN (Synchronization, синхронизация номеров очереди) установлен, а бит ACK (Acknowledgement, подтверждение приема) — нет)

Таблица 26.5 (окончание)

Опция	Пакеты, соответствующие опции
`tcpflags flags`	Соответствует пакетам, TCP-заголовки которых содержат разделенный запятыми список `flags`. Допускаются флаги `fin`, `syn`, `rst`, `psh`, `ack` и `urg`. Предшествующий значению знак `!` означает отсутствие указанного флага
`icmptypes types`	Соответствует пакетам, ICMP-тип которых присутствует в списке `types`. Список может быть представлен любой комбинацией диапазонов и/или индивидуальных типов, разделенных запятыми. Обычно используются типы: 0 — эхо-ответ (ответ ping), 3 — цель недостижима, 5 — перенаправление, 8 — эхо-запрос (запрос ping), и 11 — время превышено (время жизни — TTL — закончилось, как в traceroute(8))

Приведем примеры добавления готовых правил. Вот правило, ликвидирующее все пакеты от хоста evil.crackers.org, направленные на порт Telnet хоста nice.people.org:

```
# ipfw add deny tcp from evil.crackers.org to nice.people.org 23
```

Следующий пример запрещает и записывает весь TCP-трафик от всей сети (класса C) crackers.org ко всем портам хоста nice.people.org:

```
# ipfw add deny log tcp from evil.crackers.org/24 to nice.people.org
```

Если кто-то не хочет, чтобы люди посылали свои X-сессии на его внутреннюю сеть (класса C), ему следует применить команду:

```
# ipfw add deny tcp from any to my.org/28 6000 setup
```

Чтобы прочитать введенные правила, утилита ipfw используется с ключевым словом `list`:

```
# ipfw [-a] [-t] [-N] l[ist]
```

В этом случае опции имеют значения, приведенные в табл. 26.6.

Таблица 26.6. Опции чтения таблицы правил IPFW

Опция	Назначение
`-a`	При чтении показывает значения счетчика для правила. Эта опция — единственный способ увидеть счетчики
`-t`	Показывает время последнего совпадения для каждого входа. Вывод времени совпадения несовместим с синтаксисом ввода утилиты ipfw(8)
`-N`	Пытается использовать имена хостов и сервисов вместо IP-адресов и номеров портов

С помощью специального ключевого слова утилита ipfw удаляет все введенные правила:

```
# ipfw flush
```

кроме последнего, 65 535-го. Последнее правило зависит от типа брандмауэра и может запрещать все:

```
# ipfw list
65535 deny ip from any to any
```

Флаг `firewall_type="OPEN"` вставляет перед ним правило глобального разрешения:

```
65534 allow ip from any to any
```

поэтому с этим флагом система остается открытой при старте.

Брандмауэр IPF основан на применении альтернативного механизма фильтрации пакетов IPFILTER. Этот брандмауэр несколько сложнее в применении, но гибче, надежнее и эффективнее, чем брандмауэр IPFW. Для брандмауэра IPF ядро должно быть скомпилировано с двумя следующими опциями:

- `options IPFILTER`;
- `options IPFILTER_LOG`.

Управляется брандмауэр IPF утилитой ipf(8).

Так же как и для брандмауэра IPFW, правила записываются в таблицу ядра и изменяются управляющей утилитой. Все пакеты проверяются на соответствие образцам, но список правил прочитывается до конца, и судьба пакета определяется последним правилом, для которого соответствие было установлено, но только если правило не было помечено ключевым словом `qwick`. Правило с пометкой `qwick` выполняется сразу, прекращая дальнейшее чтение таблицы. Заполнять таблицу правил фильтрации удобнее всего, считывая правила из файла. Например, так:

```
# ipf -f /etc/ipf.rules
```

Для очищения таблицы правил фильтрации используется опция `-F`, произошедшая от слова "flush". Не вдаваясь в подробности описаний правил для брандмауэра IPF (как это было сделано для брандмауэра IPFW), приведем несколько примеров, которые должны прояснить их синтаксис. Так, чтобы запретить все входящие и исходящие пакеты, применяется набор из двух правил, например:

```
block in all
block out all
```

Чтобы пропустить все входящие пакеты от хоста firewall к любому назначению:

```
pass in from firewall to any
```

Укажем в правиле интерфейс:

```
block in on le0 from localhost to any
```

Установим ограничения по протоколам и портам, т. е. например, не будем принимать входящий трафик для NFS:

```
block in on le0 proto tcp/udp from any to any port = 2049
```

Примечание

NFS — протокол поддержки сетевых файловых систем (Network File System), использующих протоколы транспортного уровня UDP (User Datagram Protocol) и TCP (Transfer Control Protocol) и порт с номером 2049.

Запретим прием фрагментированных пакетов:

```
block in all with frag
```

Выберем для фильтрации диапазон портов — в данном случае заблокируем все порты X-терминалов:

```
block in proto tcp from any to any port 5999 >< 6010
```

Ограничим некоторые TCP-флаги (SYN или SYN-ACK):

```
block out on le0 proto tcp from any to 10.1.0.0/16 flags S/SA
```

Как построить фильтрующий брандмауэр

Все пакетные фильтры требуют осторожности в обращении, хорошего понимания основ TCP/IP, задач защиты сети и правил фильтрации. Ни в коем случае нельзя начинать работу с брандмауэром в режиме удаленного доступа — правила принимаются к исполнению мгновенно и первая же ошибка может привести к потере соединения. Не вносите при удаленном доступе старт не отлаженного брандмауэра в стартовые скрипты — можно получить "невменяемую" машину после перезагрузки.

Самые общие советы:

- заблокировать все входящие TCP-пакеты к портам ниже 1024;
- заблокировать весь входящий UDP-трафик;
- заблокировать весь входящий трафик к портам, используемым X-серверами (6000—6063);
- проверить, какие порты используют внутренние серверы (например, SQL) и заблокировать те, которые выпадают из диапазона 1—1024.

Эти советы нужно воспринимать как предварительные рекомендации. Правила своего брандмауэра вам придется выбирать самим.

Защита терминальной службы

Несмотря на успех системы графического пользовательского интерфейса (GUI — Graphical User Interface) X Window, все UNIX сохраняют ориентацию

на применение командной строки как главного пользовательского и административного интерфейса. Главный элемент всех реализаций X Window — окно псевдотерминала xterm, или подобный ему программный продукт. На заре развития сети Интернет интерфейс командной строки был единственным. Поэтому одними их первых служб прикладного уровня стали программы, реализующие протокол Telnet, и их эквиваленты rsh (Remote Shell) и rlogin (Remote Login), позволявшие работать с удаленной системой в режиме терминального доступа. С одной стороны, все они получили огромную популярность, т. к. с ними исчезало расстояние до очень удаленных серверов, а с другой стороны, они стали одной из зияющих прорех в безопасности, т. к. использовали передачу по сети незашифрованных данных, в том числе, и паролей. С появлением разного рода "снифферов" (от англ. *sniffer* — вынюхиватель, шпион) — программ, подслушивающих, как утилита tcpdump, все протокольные переговоры в физическом сегменте сети, бедствие приняло характер стихийного. Надо было спасаться. И тогда был создан протокол SSH (Secure Shell). Этот замечательный протокол не только поддерживает полные аналоги терминального интерфейса в зашифрованном варианте, но делает безопасным запуск X-приложений удаленной системы, т. к. передает данные для X-сервера по шифрованному туннелю. В состав FreeBSD 4.4 входит пакет OpenSSH, первоначально разработанный для ОС OpenBSD — другие реализации SSH постепенно переходят к распространению на платной основе.

Демон sshd прослушивает 22-й порт и для каждого клиента порождает новый экземпляр самого себя (при желании можно организовать также его старт и посредством супердемона inetd). В файл конфигурации стартовых скриптов /etc/rc.conf нужно внести параметр `sshd_enable="YES"`. В файле /etc/inetd.conf строку вызова демона telnet нужно отметить как комментарий:

```
#telnetstream tcp     nowait root    /usr/libexec/telnetd telnetd
```

и перезапустить супердемон inetd:

```
#killall -HUP inetd
```

Для полной уверенности следует включить на сервере поддержку одного из брандмауэров и заблокировать 23-й порт как для входящих TCP-пакетов, так и для исходящих. Желательно проделать все то же и для FTP (соединениям по File Transfer Protocol, 20-й и 21-й порты), т. к. в состав пакетов программ поддержки ssh всегда входит утилита scp, поддерживающая удаленное копирование файлов.

Протокол SSH версии 2 — OpenSSH — поддерживает все версии протоколов — позволяет организовать перенаправление по шифрованному каналу соединений к любому порту TCP/IP: так, можно защитить получение почты с серверов по протоколам IMAP (Internet Message Access Protocol) и POP3 (Post Office Protocol), передающим всю информацию в открытом виде и некоторые другие сервисы.

Глава 27

FreeBSD и основные сетевые сервисы

В этой главе приводится обзор основных сетевых служб Интернета с точки зрения их настройки для работы под управлением ОС FreeBSD. Почти все программы, описываемые в данной главе, не являются особенностью ОС FreeBSD. Они выпускаются для всех UNIX-подобных операционных систем, а некоторые, например Apache, реализованы и для Windows. Специфика заключается только во встраивании службы в системное окружение, размещении и настройке стартовых скриптов и управляющих утилит.

Прочитав эту главу, вы поймете, что можно сделать с вашим сервером, если удастся подключить его к сети. Практически по каждому разделу этой главы написана не одна книга, и профессиональное использование описываемых здесь сервисов требует гораздо более глубокого изучения.

Соединение "точка-точка" (PPP)

Point to Point Protocol (PPP, RFC 1331) — это протокол двухточечной связи, чаще всего используемый для выхода в сеть Интернет через модем. PPP — это протокол инкапсуляции пакетов, позволяющий передавать мультипротокольные пакеты по одному каналу. В протокол PPP входят три компонента: протокол управления каналом (Link Control Protocol, LCP), протоколы управления сетью (Network Control Protocols, NCP) и процедура инкапсуляции дейтаграмм для их передачи по последовательному каналу. Протокол PPP относится к протоколам уровня сетевого доступа. Управление им вынесено на прикладной уровень только потому, что решение об установлении соединения принимается пользователем в произвольный момент времени, т. к. применяемое в соединении "точка-точка" средство физической связи — телефонная линия — может использоваться другими устройствами, не связанными с компьютером.

Определение

Процесс помещения блоков данных одного уровня в блоки данных другого или того же уровня называют *инкапсуляцией*.

ОС FreeBSD предлагает два варианта программ, поддерживающих протокол PPP — это демон pppd, поддерживаемый ядром (kernel PPP), и программа ppp, выполняющаяся в пространстве пользовательских процессов (user PPP). Для работы демона pppd ядро системы должно быть скомпилировано с поддержкой псевдоустройства ppp (оно входит в ядро GENERIC), пользовательская программа ppp требует поддержки ядром псевдоустройства tun (оно тоже присутствует в ядре GENERIC). Протокол PPP можно использовать как для установления соединения своего компьютера с сетью, так и для подключения других компьютеров к сети через свой компьютер. Обе программы (pppd и ppp) и на стороне клиента, и на стороне сервера выполняются в режиме демона, постоянно присутствуя в памяти компьютера либо в ожидании звонка клиента, либо поддерживая уже установленное по последовательной линии соединение. Но, в отличие от демонов прикладного уровня, они никогда не создают своих копий: один демон — одна телефонная линия (клиент), 10 телефонных линий — 10 демонов (сервер). Демон на стороне клиента инициирует соединение с сервером, выполняя дозвон по телефону провайдера. Демон на стороне сервера ожидает звонка, принимает звонок, проводит аутентификацию обратившегося пользователя. После установления соединения оба демона одинаково поддерживают передачу данных по установившемуся соединению.

Для того чтобы настроить PPP-соединение на стороне клиента, необходимо получить у интернет-провайдера телефоны удаленного доступа, свои регистрационные данные (login и password), IP-адреса его DNS-серверов, тип используемой аутентификации и информацию о назначении IP-адреса клиенту. Информация о DNS-серверах провайдера должна быть записана в файле /etc/resolv.conf клиента:

```
# cat /etc/resolv.conf
search  my.default.domain
nameserver  105.10.13.1
nameserver  210.22.34.5
```

Сервер, подключающий PPP-клиентов, может использовать разные схемы аутентификации: аутентификация по имени и паролю, или аутентификация по протоколам PAP (Password Authentication Protocol) или CHAP (Challenge Handshake Authentication Protocol). Сервер может назначать клиенту при подключении один и тот же, известный клиенту, статический IP-адрес или динамический, тот, который в данный момент не занят. Эти детали имеют значение для конфигурационных файлов демонов протокола PPP.

Конфигурирование демона pppd

Демон pppd имеет множество опций, которые записываются в файле /etc/ppp/options и могут указываться в командной строке при запуске демона. Приведем пример для динамического адреса и аутентификации по имени и паролю:

```
# cat /etc/ppp/options
/dev/cuaa0  115200
crtscts
modem
connect  "/usr/bin/chat -f /etc/ppp/chat.script"
noipdefault
silent
domain  my.default.domain
defaultroute
```

Здесь `/dev/cuaa0 115200` определяет последовательный порт, к которому подключен модем (в данном случае COM1), и максимальную скорость передачи модема. Опция `crtscts` устанавливает аппаратный контроль над приемом/передачей данных. Опция `modem` включает проверку наличия сигнала Carrier Detect (CD) от модема перед открытием порта. Опция `connect` указывает имя и параметры программы дозвона (демон pppd не имеет встроенной функции дозвона), утилита /usr/bin/chat с конфигурационным файлом /etc/ppp/chat.script. На динамическое назначение адреса указывает опция `noipdefault`. Опция `silent` заставляет демона pppd ожидать прихода LCP-пакетов (LCP — Link Control Protocol). Имя домена провайдера указывается опцией `domain`. На время работы PPP-соединения опция `defaultroute` назначает маршрут default в таблице маршрутизации на адрес PPP-сервера. Если IP-адрес назначается статически, вместо опции `noipdefault` указывается комбинация `x.x.x.x:y.y.y.y`, где `x.x.x.x` — IP-адрес, назначенный клиенту провайдером, а `y.y.y.y` — IP-адрес сервера провайдера.

Программа /usr/bin/chat автоматизирует диалог с модемом, используя сценарий диалога из файла /etc/ppp/chat.script, например, такой как в листинге 27.1.

```
ABORT BUSY
ECHO OFF
SAY "Dialling your ISP...\n"
'' ATDT5551212
```

```
TIMEOUT 120
SAY "Waiting up to 2 minutes for connection ... "
CONNECT ''
SAY "Connected, now logging in ...0
ogin: account
ssword: pass
$ SAY "Logged in OK ...0 etc ...
```

При использовании PAP- и CHAP-аутентификации имя пользователя, имя сервера и пароль пользователя на серверной стороне хранятся в файлах /etc/ppp/pap-secrets и /etc/ppp/chap-secrets соответственно. Опциями демона pppd `refuse-chap`, `require-chap`, `refuse-pap` и `require-pap` сервер может установить требуемый тип аутентификации.

Опция `persist` заставляет демон pppd на стороне клиента поддерживать соединение постоянно. Опция `demand` обеспечивает установку PPP-соединения, опция `idle n` обеспечивает разрыв соединения в случае простоя в течении n секунд.

Когда все опции установлены, можно просто запустить демон pppd, и как только модем дозвониться и установит соединение, PPP-интерфейсом можно пользоваться. При работе демона pppd в режиме соединения по требованию можно использовать скрипты (сценарии) /etc/ppp/ip-up и /etc/ppp/ip-down для выполнения действий, сопровождающих установление и прекращение связи. Таким образом можно подключать, например, проверку новых поступлений на почтовом сервере, запуская демона Fetchmail из сценария `ip-up` и останавливая его в `ip-down`.

Настройка пользовательской программы ppp

Часть кода демона pppd выполняется ядром, что затрудняет работу с ним при настройке и отладке. ОС FreeBSD включает альтернативный вариант программы поддержки протокола PPP, выполненный в виде пользовательского процесса, использующего туннелирующее устройство ядра. По функциональности программа ppp ни в чем не уступает демону pppd, и даже обладает некоторыми дополнительными возможностями. Пользовательский ppp так же, как и pppd, может использоваться как демон и на стороне клиента, и на стороне сервера. Кроме того, он легко конфигурируется для поддержки трансляции адресов.

Программа ppp использует конфигурационный файл /etc/ppp/ppp.conf. Образец конфигурационного файла поставляется вместе с системой (листинг 27.2), но для реальной работы его нужно настраивать.

```
#####################################################################
# PPP  Sample Configuration File
# Originally written by Toshiharu OHNO
# Simplified 5/14/1999 by wself@cdrom.com
#
# See /usr/share/examples/ppp/ for some examples
#
# $FreeBSD: src/etc/ppp/ppp.conf,v 1.2.2.5 2001/07/13 10:55:23 brian Exp
$
#####################################################################

default:
 set log Phase Chat LCP IPCP CCP tun command
 ident user-ppp VERSION (built COMPILATIONDATE)

 # Ensure that "device" references the correct serial port
 # for your modem. (cuaa0 = COM1, cuaa1 = COM2)
 #
 set device /dev/cuaa1

 set speed 115200
 set dial "ABORT BUSY ABORT NO\\sCARRIER TIMEOUT 5 \
          \"\" AT OK-AT-OK ATE1Q0 OK \\dATDT\\T TIMEOUT 40 CONNECT"
 set timeout 180                  # 3 minute idle timer (the default)
 enable dns                       # request DNS info (for resolv.conf)

papchap:
 #
 # edit the next three lines and replace the items in caps with
 # the values which have been assigned by your ISP.
 #
 set phone PHONE_NUM
 set authname USERNAME
 set authkey PASSWORD

 set ifaddr 10.0.0.1/0 10.0.0.2/0 255.255.255.0 0.0.0.0
 add default HISADDR                        # Add a (sticky) default route
```

Здесь все параметры назначаются ключевыми словами `set` или `enable`. В конфигурационном файле имеются следующие параметры:

- `device` — указывает последовательный порт, к которому подключен модем;
- `log` — определяет перечень протоколируемых событий;
- `speed` — задает скорость последовательного порта;
- `dial` — определяет диалог модема при дозвоне;
- `ifaddr` — задает IP-адреса хоста (при статическом назначении адреса, при динамическом назначении адреса нужно оставить то значение, которое есть) и шлюза провайдера и сетевую маску (ее не нужно менять, если провайдер не указал другую маску);
- `timeout` — задает допустимое время простоя, по истечении которого соединение разрывается;
- `default HISADDR` — указывает маршрут по умолчанию (default) для таблицы маршрутизации;
- `enable DNS` — задает проверку DNS-серверов, указанных в файле /etc/resolv.conf/;
- `phone` — определяет номер телефона сервера провайдера;
- `authname` — указывает логин на сервере провайдера;
- `authkey` — указывает пароль на сервере провайдера.

Этот файл настраивает программу ppp на использование PAP- или CHAP-аутентификации. Если эту настройку нужно отключить, то из конфигурационного файла следует удалить всю секцию `papchap` вместе с опциями `set phone`, `set authname`, `set authkey`, а после опции `set dial` добавить строки:

```
provider:
    set phone "5551212"
set login "TIMEOUT 10 \ "\ " \ "\ " gin: login sword: password"
```

Слова `login` и `password` нужно заменить на действительные имя входа и пароль. После них нужно будет еще добавить команду, которая переведет программу ppp из командной оболочки в режим поддержки соединения по протоколу PPP — обычно эту команду сообщает провайдер. Этот сценарий подобен сценарию chat — он передает пары "ожидаемый ответ" — "передаваемая строка", которые отправляются серверу только после исполнения им предыдущей команды.

Для запуска программы ppp с PAP- или CHAP-аутентификацией можно ввести команду:

```
# ppp -background papchap
```

Если нужно производить аутентификацию в командной оболочке, то следует применить команду:

```
# ppp -foreground provider
```

Если нужно разрешить доступ к программе ppp для непривилегированных пользователей, файл /etc/ppp/ppp.conf должен содержать строку `"allow user"`.

Для запуска программы ppp в режиме соединения по требованию или постоянного соединения в файл /etc/rc.conf нужно добавить параметр `ppp_enable="YES"`. При этом автоматически активизируется трансляция адресов на PPP-интерфейсе, если же это нежелательно, то в конфигурационном файле /etc/rc.conf нужно установить параметр `ppp_nat="NO"`. Чтобы отказаться от PAP- и CHAP-аутентификации, следует установить флаг `ppp_profile="provider"`. Для организации постоянного соединения в файл /etc/rc.conf нужно поместить строку `ppp_mode="ddial"`. Команды, выполняемые при разрыве и возобновлении PPP-соединения, помещаются в файлах /etc/ppp/ppp.linkup и /etc/ppp/ppp.linkdown.

Службы e-mail

Общие сведения о SMTP

SMTP (Simple Mail Transfer Protocol) — один из самых первых протоколов сети Интернет, он появился вместе с электронной почтой в 1971 году. Электронная почта (Electronic mail, иначе e-mail) предназначалась для посылки текстовых сообщений с одного компьютера на другой и никаких суперсложных задач перед ней не ставилось. Поэтому протокол SMTP до удивления прост. Демон приема почтовых сообщений ожидает обращения, прослушивая 25-й порт. Посылающая сторона может непосредственно обратиться к почтовому серверу получателя:

```
$ telnet your.address.org 25
```

После простого обмена приветствиями и выяснения адресов отправителя и получателя отправитель помещает текст письма. Точка в начале пустой строки служит сигналом конца посылки. После сообщения `"QUIT"` клиента обмен прекращается. На первый взгляд, никаких дополнительных действий предпринимать не надо, и даже клиентская программа для посылки не нужна. Но не все так просто: еще нужно определить адрес сервера получателя по почтовому адресу, что не всегда легко сделать в уме, а также установить с сервером получателя контакт, что не всегда возможно. Сервер SMTP разбирает адрес получателя письма, обращаясь к службе DNS, выясняет адрес сервера получателя и пытается передать ему письмо. Если сервер получателя

в данный момент недоступен, сообщение помещается в очередь, и через определенный период времени попытки отправить его повторяются. Настройка и содержание почтового сервера весьма обременительны, поэтому даже те хосты, которые для отправки электронной почты используют программу sendmail, не всегда занимаются рассылкой почты сами, а передоверяют ее SMTP-серверам ISP.

Почтовые агенты

Программы, действующие на разных этапах передачи сообщений электронной почты, можно разделить на две категории:

- агенты пересылки почты (Mail Transfer Agent, MTA);
- пользовательские почтовые агенты (Mail User Agent, MUA).

Пути следования почты всегда одинаковы: MUA → MTA → ... → MTA → → MUA. Таким образом, данные (содержание письма) поступают к почтовому агенту пользователя (например, программе pine). Отправленное письмо передается программе передачи почтовых сообщений, такой как sendmail. По пути своего следования к почтовому серверу получателя письмо переходит от одного агента пересылки почты к другому. Почтовый сервер получателя по требованию клиента передает письмо почтовому агенту пользователя, например программе Microsoft Outlook Express.

Среди программ-агентов пересылки почты самой известной и распространенной является sendmail, написанная еще в 1983 году Эриком Олманом и непрерывно совершенствующаяся компанией Sendmail (коммерческие версии) и проектом Sendmail, выпускающим бесплатную версию программы на общественных началах. Исходный текст программы распространяется свободно, все ОС UNIX, в том числе, и FreeBSD, включают программу sendmail по умолчанию. Программы Postfix (автор Вьетс Венема) и Qmail (автор Дэн Бернстайн) разработаны как альтернатива программе sendmail для UNIX, как более быстрые, более простые и более защищенные. Из агентов пересылки почты MTA для других ОС можно выделить Microsoft Exchange.

Наиболее популярными пользовательскими почтовыми агентами для ОС UNIX являются программы pine, elm и mutt, ведущие свое происхождение от самых простых пользовательских почтовых агентов mail и mailx, появившихся на заре существования UNIX. Все они ориентированы на использование их из командной строки, и потому весьма неприхотливы. Программы pine и mutt могут обращаться не только к локальному почтовому ящику пользователя, но и к ящику, расположенному на почтовом сервере, используя протокол POP3 (Post Office Protocol).

Настройка программы sendmail

Файлы настройки почтового сервиса ОС FreeBSD размещаются в каталоге /etc/mail. Конфигурационный файл sendmail.cf программы sendmail уже давно не подлежит редактированию вручную: его синтаксис слишком сложен и неудобен для этого (вносить изменения непосредственно в конфигурационный файл можно только в случае крайней необходимости). Все изменения вносятся в файл мастера конфигурации /etc/mail/freebsd.mc (Master Config). Файл написан на языке препроцессора m4, править его следует, только руководствуясь инструкциями из файла /usr/share/sendmail/cf/README. Для получения конфигурационного файла к mc-файлу нужно подгрузить файл cf.m4. Файл makefile из каталога /etc/mail помогает сделать это автоматически:

```
# cd /etc/mail
# make cf
cp freebsd.mc .mc
/usr/bin/m4 -D_CF_DIR=/usr/share/sendmail/cf/
↳ /usr/share/sendmail/cf/m4/cf.m4 .mc>.cf
```

и установить его на нужное место командой

```
# make install
install -c -m 644 .cf /etc/mail/sendmail.cf
```

Файл aliases содержит список почтовых псевдонимов, по которым входящая почта перенаправляется на другие адреса или передается в файлы или программы:

```
andy:              Andrew@mail.ru
Andrey.Fpezer:     andy
log:               ">>/root/mail.log"
otpusk:            "|/usr/local/bin/vacation"
```

Программа sendmail получает список псевдонимов из ассоциативной таблицы псевдонимов alias.db, которую необходимо обновить после внесения изменений в файл aliases:

```
# make aliases
/usr/sbin/sendmail -bi
/etc/mail/aliases: 34 aliases, longest 12 bytes, 316 bytes total
```

База данных доступа access.db содержит правила ограничения доступа к SMTP-серверу для отдельных хостов, подсетей, доменов и адресов. Правила записываются в файле access и вносятся в базу данных командой

```
# make maps
/usr/sbin/makemap hash access.db < access
```

Если сервер обслуживает несколько доменов, принимая почту для разных групп адресов, необходимо перечислить все имена сервера, записанные в зонах обслуживающего ваш компьютер серверов службы доменных имен DNS, в файле local-host-names (в предыдущих версиях он назывался sendmail.cw). Программа sendmail читает файл local-host-names только при старте, после изменения этого файла необходимо заставить программу sendmail перечитать вновь файлы конфигурации:

```
# make restart
/bin/kill -HUP `head -1 /var/run/sendmail.pid`
```

Web-сервер

В отличие от программ почтового сервиса, Web-сервер в комплект базового программного обеспечения ОС FreeBSD не входит и должен устанавливаться дополнительно. В настоящее время самым популярным, свободно распространяемым и практически не имеющим конкурентов Web-сервером является Apache, "A PatCHy sErver", дополненный и исправленный наследник одного из самых первых HTTP-серверов, HTTP-сервера NCSA (National Center for Supercompitung Applications). Apache выпускается для очень многих и разных платформ. Его можно собрать и установить из архива исходных файлов почти на любой ОС. Для ОС FreeBSD Web-сервер Apache выпущен в виде порта apache-1.3.20, включенного в дополнительное программное обеспечение дистрибутива. В настоящее время параллельно распространяются и поддерживаются две версии этого сервера: традиционная ветвь с номером версии 1.3 (последний выпущенный вариант 1.3.27) и первые выпуски версии 2.0 (последний вариант 2.0.49). В дистрибутив ОС FreeBSD включены порты с Web-сервером Apache версии 1.3.20 и beta-версия Apache2, apache-2.0.16-beta.

Как работает HTTP-сервер?

В ответ на запрос клиента сервер передает ему файлы по протоколу HTTP (HyperText Transfer Protocol, RFC 1945). Первоначально предполагалось оперативно передавать текстовые и гипертекстовые (HTML) файлы без авторизации и аутентификации клиента. Протокол HTTP не поддерживает сеансов — соединение закрывается сразу после выполнения запроса. В листинге 27.3 показано, как в ответ на обращение клиента с запросом GET сервер отвечает передачей файла и его заголовков, содержащих массу полезной информациию

```
$ telnet www.some.http-server.com 80
Connected to www.some.http-server.com.
```

```
Escape character is '^]'.
GET /index.html HTTP/1.0
Host: www.some.http-server.com
HTTP/1.1 200 OK
Date: Sat, 25 Jan 2003 11:55:04 GMT
Server: Apache/1.3.27 (Unix) mod_jk Resin/2.1.6 PHP/4.2.3 rus/PL30.16
Content-Length: 1399
Last-Modified: Mon, 02 Jul 2001 11:10:09 GMT
ETag: "4ab90-577-3b405691-koi8-r"
Connection: close
Content-Type: text/html; charset=koi8-r
Expires: Thu, 01 Jan 1970 00:00:01 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
 <HEAD>
  <TITLE>Test Page for Apache Installation</TITLE>
 </HEAD>
<!-- Background white, links blue (unvisited), navy (visited), red
(active) -->
 <BODY
  BGCOLOR="#FFFFFF"
  TEXT="#000000"
  LINK="#0000FF"
  VLINK="#000080"
  ALINK="#FF0000"
 >
<P>
If you can see this, it means that the installation of the
<A HREF="http://apache.lexa.ru/">Russian Apache web server</A> software
on this system was successful.  You may now add content to this directory
and replace this page.

<P><HR WIDTH="50%" SIZE="8">

<H2 ALIGN="CENTER">Seeing this instead of the website you expected?</H2>


<P>
This page is here because the site administrator has changed the configu-
ration of this web server.
```

```
Please  <STRONG>contact the person responsible for maintaining this
server with questions.</STRONG>  The Apache Software Foundation, which
wrote the web server software this site administrator is using, has noth-
ing to do with maintaining this site and cannot help resolve configura-
tion issues.

<P><HR WIDTH="50%" SIZE="8">


<P>
The Apache <A HREF="manual/index.html">documentation</A> has been in-
cluded with this distribution.

<P>
You are free to use the image below on an Apache-powered web server.
Thanks for using Apache!


<DIV ALIGN="CENTER"><a href="http://apache.lexa.ru"><IMG SRC="ra-
powered.gif" ALT=""></a></DIV>
</BODY>
</HTML>
```

Кроме команды в запрос включается много дополнительной информации, благодаря которой по HTTP-соединению удается имитировать сеансы связи и авторизацию пользователя в рамках сеанса. В запросе также сообщается информация о программе-клиенте, на этом строится автоматическое перекодирование текста при передаче файлов. Способность сервера выполнять программы сделало повсеместным внедрение динамических страниц, в которых вид передаваемого документа зависит от параметров, вводимых клиентом, или других изменяющихся факторов. Использование протокола HTTP вышло далеко за рамки своего первоначального предназначения: по этому протоколу в виде бинарных файлов передаются изображения и звук, исполняемые элементы и архивные файлы. Версия протокола HTTP/1.0 ограничивалась всего несколькими видами запросов клиента: `GET`, `POST` и `HEAD`. В следующей, расширенной, версии протокола HTTP/1.1 (RFC 2048) к ним добавились `PUT` и `DELETE`, и клиенты получили возможность не только получать файлы с Web-серверов, но и загружать их (по англ. *upload*) туда. Возможности протокола HTTP расширились настолько, что Web-интерфейс стал почти стандартным методом заполнения счетов и заявок, поиска в справочных системах, складского учета и много другого.

Функциональность сервера Apache обеспечивается программой httpd, которая при запуске создает несколько перманентно присутствующих в оперативной памяти экземпляров самой себя. Демон httpd прослушивает

80-й порт (или другой, это зависит от параметров конфигурации). Демонов должно быть несколько, т. к. для каждого нового запроса создается новый экземпляр демона (серверы Apache-2 используют другой механизм саморазмножения). При старте сервер Apache создает сразу несколько экземпляров программы httpd: один принадлежит суперпользователю, остальные — непривилегированному пользователю (для ОС FreeBSD — пользователю nobody). Суперпользовательский процесс (головной демон) прослушивает привилегированный порт 80 (именно поэтому он должен принадлежать суперпользователю) и управляет остальными демонами. Минимальное число подчиненных демонов определяется в конфигурационным файле.

Web-сервер Apache представляет собой очень гибкую систему с несколькими уровнями настроек. Первый уровень — при сборке сервера, позволяет статически включить при компиляции в программу httpd поддержку тех или иных свойств, например, поддержку динамических модулей. Второй уровень приспособляемости определяется набором динамических модулей, который задается администратором сервера и может сокращаться и дополняться в зависимости от того, какие модули нужны в настоящий момент. И третий уровень приспособляемости — конфигурационный файл сервера, содержащий директивы времени исполнения. Несколько десятков директив времени исполнения определяют имя сервера, прослушиваемые порты, список загружаемых динамических модулей, структуру каталогов сервера и уровень их доступности. Web-сервер строит свое дерево файлов, которое может не иметь ничего общего с реальной файловой структурой, в которой расположены документы сервера. Один httpd-демон может поддерживать несколько серверов с различными именами, они называются виртуальными.

Расположение файлов сервера Apache

После установки сервера Apache из порта все файлы, связанные с сервером, располагаются в подкаталогах /usr/local. Полный перечень файлов пакета можно получить посредством утилиты pkg_info, он занимает несколько страниц.

```
# pkg_info -L apache-1.3.20
Information for apache-1.3.20:
Files:
```

В приведенном далее списке оставлены только наиболее значимые файлы.

- Страницы системных руководств для программ и утилит Apache:

  ```
  /usr/local/man/man1/dbmmanage.1.gz
  /usr/local/man/man1/htdigest.1.gz
  /usr/local/man/man1/htpasswd.1.gz
  /usr/local/man/man8/ab.8.gz
  ```

```
/usr/local/man/man8/apachectl.8.gz
/usr/local/man/man8/apxs.8.gz
/usr/local/man/man8/httpd.8.gz
/usr/local/man/man8/logresolve.8.gz
/usr/local/man/man8/rotatelogs.8.gz
```

- Утилиты пользовательского уровня:

```
/usr/local/bin/dbmmanage
/usr/local/bin/htdigest
/usr/local/bin/htpasswd
```

- Образцы конфигурационных файлов:

```
/usr/local/etc/apache/access.conf.default
/usr/local/etc/apache/httpd.conf.default
/usr/local/etc/apache/magic.default
/usr/local/etc/apache/mime.types.default
/usr/local/etc/apache/srm.conf.default
```

- Стартовый скрипт:

```
/usr/local/etc/rc.d/apache.sh
```

- Файлы заголовков Apache:

```
/usr/local/include/apache/ap.h
.....
```

- Динамически загружаемые модули:

```
/usr/local/libexec/apache/httpd.exp
/usr/local/libexec/apache/libproxy.so
.....
```

- Программы и утилиты административного уровня:

```
/usr/local/sbin/ab
/usr/local/sbin/apachectl
/usr/local/sbin/apxs
/usr/local/sbin/httpd
/usr/local/sbin/logresolve
/usr/local/sbin/rotatelogs
```

- Каталог первых страниц "пустого" сервера:

```
/usr/local/share/doc/apache/README.rus
/usr/local/share/doc/apache/apache_pb.gif
.....
```

- Каталог HTML-страниц руководства к Apache:

```
/usr/local/share/doc/apache/manual/LICENSE
/usr/local/share/doc/apache/manual/bind.html
.....
```

- CGI-скрипты, поставляемые вместе с Apache:

```
/usr/local/www/cgi-bin.default/printenv
/usr/local/www/cgi-bin.default/test-cgi
```

Во время своей работы Web-сервер Apache сам создает некоторые файлы. Это файл с номером процесса головного сервера и файлы для записи сообщений о поступивших запросах (access_log) и произошедших ошибках (error_log). Имена этих файлов задаются в конфигурационном файле, по умолчанию /var/run/apache.pid, /usr/local/etc/apache/log/access_log и /usr/local/etc/apache/log/error_log. Как правило, все неудачи с запуском или другие случаи неожиданного поведения сервера отражаются в файле error_log. Файл apache.pid нужен для управления сервером: управляющие утилиты посылают сигналы головному демону, для этого они должны знать номер его процесса.

Утилиты пакета Apache

Прежде всего нужно отметить скрипт /usr/local/sbin/apachectl, обеспечивающий управление демоном: старт, останов, рестарт, перечитывание конфигурационного файла и проверка его синтаксиса — все эти задачи решаются с помощью команды `apachectl` с разными опциями. Утилита /usr/local/sbin/apxs нужна для создания динамически загружаемых модулей сервера — очень полезная вещь для расширения возможностей сервера. Утилита /usr/local/sbin/rotatelogs, поставленная в cron — службу выполнения заданий по расписанию, — автоматизирует ротацию log-файлов сервера, достигающих без этого при большой загруженности сервера необъятных размеров. Утилиты /usr/bin/htpasswd, /usr/bin/htdigest и /usr/bin/dbmmanage служат для назначения паролей пользователям зон ограниченного доступа для разных методов авторизации.

Директивы времени исполнения

Гибкость настроек Web-сервера Apache способна поразить воображение: изменение нескольких директив времени исполнения в конфигурационном файле сервера может преобразить поведение сервера до неузнаваемости. Список директив времени исполнения содержит несколько десятков имен, которые, при необходимости, нужно изучать специально (описание runtime-директив входит в пакет документации, поставляемый вместе с сервером Apache). Их следует разделить на две категории: глобальные и местного зна-

чения. Наиболее важными для сервера являются директивы: `ServerRoot`, задающая каталог, в котором находится файл httpd — файл-программа сервера; `DocumentRoot`, определяющая каталог-вершину иерархии файловой структуры документов; `ServerType`, описывающая тип запуска сервера; `Listen`, назначающая порт, который сервер будет прослушивать. Директивы локального характера помещаются внутрь группы директив, относящихся к части файловой (или адресной) иерархии сервера, например:

```
<Directory "/var/www/docs/secure">
  Options       None
  AllowOverride  None
</Directory>
```

и определяют круг доступа, перечень разрешенных операций только для указанной части иерархии документов.

FTP-сервер

Как и протокол HTTP, протокол FTP (File Transfer Protocol, RFC 959) разрабатывался для передачи файлов по сети. Однако цель, с которой он создавался, была другой: протокол FTP ориентирован на передачу больших двоичных файлов и поэтому совсем не похож на протокол HTTP. Протокол FTP имеет ряд характерных отличий:

- поддерживает сеансы связи;
- требует авторизации клиента на сервере;
- поддерживает взаимодействие с файловыми системами сервера и клиента;
- устанавливает два соединения в рамках одного сеанса связи: одно для передачи команд, другое — для передачи данных.

Общая беда всех FTP-демонов — многочисленные прорехи в безопасности, причиной которых отчасти являются недостатки протокола и сложность возлагаемых на FTP-сервер задач. Базовое программное обеспечение ОС FreeBSD включает достаточно сложный и хорошо защищенный FTP-сервер, готовый к работе при настройках по умолчанию. Базовый демон имеет стандартное имя ftpd, прослушивает 21-й порт и, при отсутствии специальных указаний, стартует через посредство супердемона inetd. Чтобы исключить посредство inetd, демон ftpd должен запускаться с опцией `-D`. Непосредственное прослушивание 21-го порта FTP-демоном имеет смысл при интенсивном использовании этого сервиса на сервере, если, скажем, хост является специализированным FTP-сервером. Запись сообщений демона ftpd производится через службу регистрации системных событий syslog по статье log_ftp. Опция `-S` демона ftpd включает протоколирование в файле /var/log/ftpd всех передач файлов через анонимный сервер.

Назначение и размещение файлов FTP-сервера

Без специальных указаний FTP-сервер предоставляет услуги только пользователям, зарегистрированным на хосте сервера — это *авторизованный сервер*. При необходимости FTP-сервер можно настроить так, чтобы он обслуживал сторонних, не зарегистрированных в системе, клиентов — *анонимный сервер*. Настроить FTP-сервер в ОС FreeBSD как анонимный можно с помощью программы /stand/sysinstall при установке системы или в любое другое время. Общие файлы конфигурации и файлы конфигурации авторизованного FTP-сервера традиционно размещаются в каталоге /etc, файловые структуры анонимного сервера создаются в каталоге /var/ftp.

Глобальные файлы конфигурации приведены далее.

- Файл /etc/ftpusers содержит список пользователей, которым FTP-доступ к серверу запрещен. Файл обычно содержит список системных пользователей: root, bin, adm, tty и др. Используя записи в этом файле, можно запретить доступ по протоколу FTP не только отдельным пользователям, но и целым группам пользователей:

```
# $FreeBSD: src/etc/ftpusers,v 1.6 1999/08/27 23:23:41 peter Exp
$
#
# list of users disallowed any ftp access.
# read by ftpd(8).
root
toor
daemon
operator
bin
tty
.....................................
nobody
@wheel
```

  Запись `@wheel` запрещает вход по протоколу FTP всем членам группы wheel.

- Файл /etc/ftpchroot — содержит список пользователей, для которых выполняется переназначение корневого каталога, т. к. это делается для пользователей анонимного сервера. Символ "@" предваряет запись для группы.
- Файл /etc/welcome — содержит текст приветственного сообщения, выводимого перед приглашением к авторизации при обращении пользователей к серверу.

- Файл /etc/ftpmotd — содержит текст второго приветственного сообщения, которое выводится на экран для успешно зарегистрировавшихся пользователей.
- Файл /etc/shells — содержит список командных интерпретаторов, с которыми допускается регистрация в системе. Предотвращает регистрацию в системе пользователей без командного интерпретатора, служит дополнительным ограничителем доступа по протоколу FTP.
- Файл /var/run/nologin — если этот файл существует, то вход в систему невозможен, и не только по протоколу FTP. Содержание файла /var/run/nologin выводится на экран и соединение прекращается. Этот способ хорош для временного ограничения входа в систему для всех пользователей.
- Файл /etc/ftphosts — файл конфигурации виртуальных серверов, подобных виртуальным хостам HTTP-сервера. Файл имеет формат таблицы со следующими столбцами:
  - `hostname` — IP-адрес виртуального хоста. В отличие от сервера Apache, демон ftpd не способен различать виртуальные хосты по именам. Даже если в этом поле указано имя виртуального хоста, демон ftpd все равно будет использовать соответствующий этому имени IP-адрес;
  - `user` — имя пользователя виртуального хоста. Домашний каталог этого пользователя будет использоваться как область доступа по протоколу FTP для анонимного пользователя;
  - `statfile` — файл статистики использования виртуального сервера;
  - `welcome` — файл первого приветствия (по умолчанию /etc/ftpwelcome);
  - `motd` — файл второго приветствия (по умолчанию /etc/ftpmotd).

Для анонимного FTP-сервера отводится каталог /var/ftp. Для анонимного FTP-входа создается учетная запись в файле /etc/passwd для пользователя с именем ftp и домашним каталогом /var/ftp, принадлежащим пользователю root. Для пользователя ftp утилитой chroot вершина файловый системы, т. е. каталог /, устанавливается в его домашнем каталоге. Все файлы вне каталога /var/ftp для анонимного пользователя не существуют. Поэтому в каталоге /var/ftp создается аналог настоящей файловой структуры, необходимый для деятельности в рамках протокола FTP. Там создается подкаталог etc (в действительности, /var/ftp/etc) с файлами passwd и group. Файлы /var/ftp/etc/passwd и /var/ftp/etc/group содержат записи только тех пользователей и групп, которые нужны для перевода идентификаторов пользователя и группы в имя и название группы владельца файлов при выполнении *команды* `ls`. *Поле пароля файла passwd не используется и не должно содер*жать настоящих данных. Файл /var/ftp/etc/ftpmotd содержит приветствие для анонимных пользователей. В подкаталоге pub в произвольном порядке размещаются файлы, предоставляемые клиентам для чтения. Подкаталог

incoming с модификатором доступа 1777 открывается для записи файлов клиентами, если это зачем-то нужно серверу (sticky bit закрепляет право уничтожения файла в общедоступном каталоге за его владельцем). Обычно еще создается подкаталог bin с утилитами ls и date, но для демона ftpd ОС FreeBSD он не нужен — эти утилиты встроены в программу демона.

Управление FTP-доступом

Служба FTP потенциально опасна для сервера. Открытие FTP-доступа всегда сопряжено с повышением риска взлома системы. В первую очередь опасность представляет передача незашифрованных паролей при авторизации на сервере. Во вторую — ежемесячно появляющиеся в сети Интернет "эксплойты" (программы диверсионного характера, использующие дефекты программного обеспечения системы для получения прав привилегированного пользователя или других деяний, компрометирующих систему), применяющие очередную фантастическую прореху, открытую то в одном, то в другом FTP-сервере. К сожалению, FTP-сервис чрезвычайно широко распространен и тяжело поддается замене более безопасными аналогами. Рекомендуется использовать зашифрованный вариант FTP, использующий протокол SSH. Пакет OpenSSH содержит серверную оболочку /usr/libexec/sftp-server, имитирующую действия обычного FTP-сервера при обращении клиента с программой sftp. Корень проблемы незаменимости обычного протокола FTP состоял в недоступности sftp-клиентов для ОС Windows — настольной операционной системы подавляющего большинства пользователей FTP-серверов. Но положение меняется и SSH-клиенты для ОС Windows стали появляться: в частности, пакет SSH Communication Security для Windows, комплекс Putty и др.

Если же открытие FTP-сервиса все же необходимо, следует попытаться ограничить доступ к нему брандмауэром, пропуская запросы на соединения по протоколу FTP только от известных клиентов сервера.

Сервер DNS

Настройка сервера службы доменных имен (DNS) — одна из самых сложных и тонких задач сетевого администрирования, и этой теме посвящена не одна книга. Постараемся кратко и с максимальной простотой обрисовать типичную постановку сервера имен в системе BIND (Berkeley Internet Name Domain), традиционного для UNIX-сервера DNS.

Структура системы доменных имен

Иерархия доменных имен строится в порядке, обратном расположению цепочки доменов в именах компьютеров. Так, элементы имени somehost.somenet.org при следовании в порядке подчиненности расположатся

так: org → somenet → somehost. Вершиной иерархии является элемент ".", обычно в написании имен компьютеров опускаемый. Нижний элемент иерархии, тот, с которого начинается имя компьютера, и является собственно именем компьютера, все остальные элементы являются именами доменов. Каждый домен обслуживается, условно говоря, своим сервером имен. Сервер домена отвечает за имена компьютеров и имена подчиненных доменов, перекладывая на их серверы (делегируя) ответственность за имена их компьютеров и их поддоменов. В действительности же домен, как правило, обслуживается группой серверов, синхронизирующих информацию по определенным правилам. Таким образом, серверам, обслуживающим вершину иерархии ".", нужно знать только то, какие серверы обслуживают домены верхнего уровня. Доменов верхнего уровня не так уж и много, чтобы несколько десятков серверов корневого уровня (они так называются), разбросанных по всему миру, не справились с ней.

Примечание

К доменам верхнего уровня относятся имена доменов государств — ru, uk, pl и т. п. — и имена доменов верхнего уровня в США, назначавшиеся не по государственной, а по ведомственной принадлежности: com, edu, gov, mil.

Серверы, отвечающие за домены верхнего уровня, содержат информацию об именах компьютеров в своем домене (например, **www.ru**) и серверах, отвечающих за подчиненные домены. Область ответственности сервера имен — домен за вычетом поддоменов — называется *зоной*. Зона обычно описывается одним текстовым файлом, записи в котором делаются по строго определенным правилам.

При такой организации — сверху вниз, всегда есть тот, кто знает ответственного за нижележащую зону, поэтому найти нужную запись сравнительно легко. Промежуточные результаты поиска оседают в памяти (кэше) сервера, и прежде чем начать новый поиск, сервер поищет нужные сведения там. По принципу проведения поиска серверы разделяются на:

- рекурсивные;
- нерекурсивные.

Рекурсивный сервер возвращает клиенту окончательный ответ — как положительный, так и отрицательный, организуя процедуру опроса сам. *Нерекурсивный сервер* возвращает только положительный ответ, если он его знает, перекладывая организацию опроса других серверов на клиента. Серверы имен обязаны иметь "дублера" — сервер, который копирует информацию своего напарника на случай временного выхода первичного сервера из строя. Сервер-"дублер" называется *вторичным сервером*.

В системе DNS решаются две задачи.

- Прямого поиска — определения адресов компьютеров по их именам.
- Обратного поиска — определения имен компьютеров по адресам.

Файлы и программы пакета BIND

Пакет BIND входит в состав ОС FreeBSD по умолчанию, его основные программные компоненты:

- /usr/sbin/named — демон сервера имен, прослушивающий 53-й порт в ожидании поступления запросов;
- /usrsbin/ndc — программа управления демоном сервера имен, она вызывается для запуска, останова, контроля и перезагрузки сервера;
- /etc/namedb — каталог — база данных сервера. Файлы конфигурации и определения зон должны находиться в этом каталоге или его подкаталогах;
- /etc/namedb/named.conf — главный конфигурационный файл сервера, определяющий сферу его полномочий.

Файлы зон не могут поставляться вместе с пакетом — каждая зона уникальна и потому следует изучить правила, по которым они создаются. Зона описывается набором записей разного типа, число которых ограничено, а формат — строго определен.

Записи типа A обеспечивают перевод имен в адреса и имеют следующий формат:

```
Имя_компьютера  [время жизни]  [класс]  A  IP-адрес
```

Первое и последнее поля записи в комментариях не нуждаются, а все остальные следует пояснить:

- `[время жизни]` — время в секундах, в течение которого данная запись может считаться действительной в кэше сервера-посредника;
- `[класс]` — класс сети, по умолчанию IN (Интернет), другие типы сетей не встречаются почти никогда;
- `A` — тип записи (в данном случае это тип A).

Записи типа PTR, обеспечивающие решение задачи обратного поиска, имеют более замысловатую структуру:

```
адрес  [время жизни]  [класс]  PTR  полностью_определенное_имя.
```

Замысловатость записей типа PTR определяется названием зоны, в которой они делаются: IN-ADDR.ARPA. Поле, имеющее название `полностью_определенное_имя`, должно содержать полное имя компьютера — со всеми доменными приставками.

Примечание

Символ ".", завершающий запись типа PTR, является обязательной частью записи, а не концом предложения или опечаткой.

Запись типа NS описывает серверы, ответственные за данную зону, или, как принято их называть, *авторитетные серверы* для данной зоны:

```
зона [время жизни] [класс] NS полностью_определенное_имя.
```

Запись типа SOA (Start Of Authority) — стартовая запись зоны, отмечающая начало зоны, имеет наиболее сложный формат. Трудно поверить, но она почти укладывается в общую схему записей системы DNS:

```
<имя> [время жизни] [класс] <тип> <значение>
```

Первое поле заполняется именем зоны, которое может вводиться по-разному. Самый простой способ указать его — это записать имя зоны с точкой в конце:

```
somenet.org.
```

Время можно опустить, а класс и тип записи понятны:

```
somenet.org. IN SOA names.somenet.org. root.somenet.org.
```

Вся сложность формата записи типа SOA сосредоточена в поле `<значение>`, т. к. оно содержит несколько имен и набор параметров:

- имя сервера авторитетного сервера имен (`names.somenet.org.`);
- адрес электронной почты администратора сервера, в котором символ "@" заменен точкой: `root.somenet.org.`;
- параметры, заключаются в круглые скобки и определяются порядком следования. Их названия обычно приводятся как комментарий, напоминающий смысл параметра:
  - `serial` — последовательный номер, указывающий на последнее обновление записей зоны. Ни первичный, ни вторичные серверы не произведут обновление информации, не обнаружив увеличения этого параметра;
  - `Refresh` — задает интервал времени (в секундах), по истечении которого вторичные серверы должны будут перечитать информацию о содержимом зоны на основном сервере;
  - `Retry` — интервал времени (в секундах), по истечении которого вторичные серверы обязаны повторить неудавшуюся попытку считать содержимое зоны на основном сервере;
  - `Expire` — интервал времени (в секундах), по истечении которого от момента последнего перечитывания зоны с основного сервера, вторичные серверы прекращают поддержку дубликата зоны;
  - `Minimum` — определяет время жизни записей о ресурсах, принимающееся по умолчанию.

Этого набора записей достаточно, чтобы понять примеры файла зон прямого (листинг 27.4) и обратного поиска (листинг 27.5), приведенные далее.

Листинг 27.4. Пример файла зоны прямого поиска

```
# cat somenet.org
$TTL 3600
  somenet.org.   IN  SOA  names.somenet.org.  root.somenet.org (
            200206071  ; Serial
            10800    ; Refresh
            3600     ; Retry
            604800   ; Expire
            86400)   ; Minimum
; DNS servers
@         IN  NS  names.somenet.org.
@         IN  NS  ns1.somenet.org.
; Hosts names
  localhost    IN  A  127.0.0.1
  ns1       IN  A  95.19.131.161
  somehost     IN  A  95.19.131.162
  mail       IN  A  95.19.131.163
  another     IN  A  165.20.41.12
  @       IN  A  95.19.131.161
; Aliases
  www       IN  CNAME  @
  ftp       IN  CNAME  somehost
; MX records
  @       IN  MX  10  mail
```

Здесь, кроме уже рассмотренных записей типа SOA, A, и NS, есть записи типа CNAME и MX, специальный символ `@` и переменная `$TTL`. Символ `@` всегда заменяется именем зоны. Все имена, кроме заканчивающихся символом ".", дополняются именем зоны. Знак `$` вводит глобальные переменные, или директивы, в частности `$TTL` устанавливает время жизни для записей, в которых поле `[время жизни]` опущено. Смысл записей CNAME и MX относительно прост:

- CNAME — *определяют имя-псевдоним для уже поименованного адреса;*
- MX — указывают на хосты, использующиеся для доставки электронной почты в данной зоне.

Файл зоны обратного поиска содержит запись типа SOA, записи NS и множество записей PTR (листинг 27.5).

Листинг 27.5. Пример файла зоны обратного поиска

```
# cat 131.19.95.in-addr.arpa
$TTL 3600
131.19.95.in-addr.arpa.  IN  SOA  names.somenet.org. root.somenet.org (
            200206071  ; Serial
            10800    ; Refresh
            3600     ; Retry
            604800   ; Expire
            86400)   ; Minimum
; DNS servers
@         IN  NS  names.somenet.org.
@         IN  NS  ns1.somenet.org.
;
161     IN  PTR  ns1.somenet.org.
162     IN  PTR  somehost.somenet.org.
163     IN  PTR  mail.somenet.org.
```

Замысловатость записей типа PTR определяется тем, что для поиска адресов по именам и для поиска имен по адресам используется один и тот же механизм.

Конфигурационный файл named.conf

Файл конфигурации демона сервиса имен в последних версиях BIND состоит из нескольких описательных блоков с С-подобным синтаксисом (листинг 27.6).

Листинг 27.6. Пример конфигурационного файла сервиса имен named.conf

```
# cat /etc/named/named.conf
logging {
  category lame-servers { null; };
  category cname  { null; };
};
options {
  directory "/etc/namedb";
};
```

```
zone "somenet.org" {
  type master;
  file "somenet.org";
};
zone 131.19.95.in-addr.arpa" {
  type master;
  file "131.19.95.in-addr.arpa";
};
zone "othernet.com" {
  type slave;
  file slaves/othernet.com";
  masters { 191.19.34.67; };
};
zone "." {
  type hint;
  file "named.boot";
};
zone "0.0.127.in-addr.arpa" {
  type master;
  file "localhost.rev";
};
```

Блоки `logging` и `options` описывают настройки сервера, блоки `zone` — поддерживаемые зоны. Каждая зона хранится, как правило, в файле, имя которого совпадает с названием зоны. Файл зоны обратного поиска для адреса закольцовывающегося интерфейса должен присутствовать обязательно; его можно создать с помощью специального скрипта /etc/namedb/make-localhost. Один и тот же сервер может быть первичным для одних зон (type master) и вторичным для других (type slave). Зона "." является зоной предположительного размещения серверов корневого уровня, список которых содержится в файле /etc/namedb/named.boot. Теперь можно стартовать сервер имен:

```
# /usr/sbin/ndc start
```

После любых изменений конфигурационного файла сервер имен должен быть перезапущен:

```
# /usr/sbin/ndc reload
```

Если сервер имен должен работать постоянно, нужно внести в файл /etc/rc.conf параметр `named_enable="YES"`.

Сетевая файловая система

Network File System (NFS, RFC 1094) позволяет компьютерам использовать файловые системы других компьютеров как свои собственные. Все ОС UNIX монтируют удаленные файловые системы так же легко, как свои собственные, без всяких ограничений, связанных с принадлежностью к одному физическому сегменту или одному домену. При этом разрыв соединения с файловым сервером проявляется только в ожидании завершения операций с "сетевыми" файлами, после восстановления связи или перезагрузки сервера можно продолжать работу с ним как ни в чем не бывало. Сервер же на потерю клиента не отреагирует никак.

Сетевая файловая система применяется там, где нужна централизация ресурсов или поддержка единообразия без обременительного копирования. Например, сеть рабочих станций монтирует каталог /home центрального сервера и пользователи сети, вне зависимости от станции регистрации, получают доступ к одним и тем же файлам своего домашнего каталога. Если для авторизации сеть использует службу NIS (Network Information Service), то права доступа и принадлежность файлов на всех хостах будут одинаковыми. Временное объединение файловых систем по протоколу NFS очень удобно для копирования больших файлов или создания резервных копий. Один из вариантов инсталляции ОС FreeBSD предусматривает монтирование дистрибутива по протоколу NFS с сервера инсталляции.

Служба NFS базируется на протоколе UDP: это как раз тот случай, когда сокращение объема контролирующего обмена предпочтительнее, чем надежность и управляемость соединения. Целостность данных обеспечивается самим протоколом NFS, и не возлагается на протокол транспортного уровня. Протокол NFS это, в сущности, набор протоколов, который строится на средствах XDR (External Data Representation) и RPC (Remote Procedure Calls), изобретенных, как и протокол NFS, фирмой Sun Microsystems.

Сетевая файловая система состоит из комплекса компонентов: протокола и сервера монтирования и комплекта демонов, координирующих базовый сервис протокола NFS. Сетевой компьютер может быть NFS-сервером, NFS-клиентом, а может быть и тем, и другим одновременно.

Конфигурирование сервера NFS

Благодаря общей простоте процедуры конфигурации сервера в ОС FreeBSD, настройка NFS-сервера сведена к минимуму операций, а именно для этого в файл /etc/rc.conf необходимо добавить следующую строку:

```
nfs_server_enable="YES"
```

Значение `"YES"` должно быть также назначено параметру `portmap_enable`, т. к. демон portmap необходим для функционирования протокола RPC

(Remote Procedure Call). Этих действий достаточно для запуска демонов, поддерживающих протокол NFS, после перезагрузки.

Прежде чем ее произвести, нужно решить, какие файловые системы и каким клиентам следует позволить использовать. Разрешение на использование файловых систем по протоколу NFS в ОС FreeBSD называется *экспортированием*, а информация об экспортируемых файловых системах записывается в файле /etc/exports. В этом файле перечисляются каталоги, к которым предоставляется доступ, а также пользователи и системы, имеющие право на доступ к ним. Файл имеет формат:

```
<список каталогов> <опции экспортирования> <список клиентов>
```

Например, следующая строка разрешает использование каталога /home всем пользователям хоста dieu.sait.ou.com:

```
/home  -rw -alldirs  dieu.sait.ou.com
```

Принадлежность файлов экспортируемой файловой системы на удаленном клиенте определяется идентификатором пользователя — UID (User ID), т. е. числовым значением, используемым системой UNIX для идентификации пользователя (информация о принадлежности файлов записывается в файловой системе). Если на сервере UID равно 1527 и ему соответствует имя пользователя sasha, а в системе клиента UID так же равно 1527 и соответствует пользователю vova, то vova и будет владельцем файлов пользователя sasha на клиентской машине. Поэтому, следует позаботиться о соответствии регистрационных номеров пользователей на сервере и клиенте. Принципиально важно определить пользователя root на клиенте для экспортируемой файловой системы. Для этого используется опция `-maproot`, приписывающая права пользователя root пользователю с заданными идентификационными номерами на удаленной системе. Другая версия опции `map`, это опция `-mapall`, что называется, одним махом сводит права всех удаленных пользователей экспортируемой системы к правам указанного пользователя. Список клиентов может быть перечислением отдельных хостов, адресом подсети с указанной маской или сетевой группой. Сетевые группы определяются в файле /etc/network и позволяют объединить в одну группу очень разные объекты сети — от подсетей и доменов до отдельных пользователей на разных хостах. В следующем примере файловые системы /usr и /opt экспортируются для подсети 212.19.34.0 с привязкой прав всех пользователей удаленных клиентов к правам пользователя nobody:

```
/usr /opt -mapall=nobody -network 212.19.34.0 -mask 255.255.255.0
```

Если файл /etc/exports не существует или недоступен для чтения при старте сетевых сервисов, демоны, поддерживающие протокол NFS, не запускаются.

Демоны NFS

Особенностью NFS является наличие у nfsd — основного демона сервиса — группы поддержки в виде демона монтирования mountd. Именно демон mountd читает файл /etc/exports и обслуживает запросы на удаленное монтирование. Этот демон запускается автоматически вместе с основным демоном nfsd.

Демон nfsd обслуживает запросы удаленных клиентов; и каждому клиенту нужен свой экземпляр демона. Поэтому при запуске демона nfsd необходимо указать число его копий, порождаемых при старте. Число экземпляров демона зависит от количества клиентов, одновременно обращающихся к серверу, при этом недостаточное число экземпляров препятствует оперативному обслуживанию клиентов, а слишком большое число экземпляров чрезмерно загружает процессор. И то, и другое понижает эффективность сетевой файловой системы. ОС FreeBSD по умолчанию запустит четыре экземпляра демона nfsd, т. к. именно это число указано в переменной `nfs_server_flags="-u -t -n 4` в файле /etc/defaults/rc.conf. Опции `-u` и `-t` разрешают обслуживать клиентов, подключающихся как по протоколу UDP, так и по протоколу TCP. Опция `-h` демона nfsd назначает IP-адрес для корректного обслуживания клиентских запросов по протоколу UDP NFS-сервером с несколькими интерфейсами.

Конфигурирование клиента NFS

Для поддержки протокола NFS со стороны клиента, в сущности, ничего, кроме запроса на монтирование удаленной файловой системы, не нужно:

```
# mount_nfs server_home:/home /shome
```

Если монтирование пройдет успешно, никаких сообщений не появится и файловая система станет доступной клиенту как каталог /shome. Освободить ресурс можно следующей командой:

```
# umount /shome
```

Для успешного размонтирования необходимо освободить все файлы и каталоги /shome, иначе будет получено сообщение "device busy" ("устройство занято"). Утилита mount_nfs имеет множество опций, определяющих разные нюансы монтирования. Если удаленную файловую систему планируется использовать часто, имеет смысл внести строку с параметрами монтирования в файл /etc/fstab:

```
# Device            Mountpoint     Fstype     Options  Dump  Pass#
server_home:/home   /shome  nfs    rw,noauto  0        0
```

Если ресурс планируется использовать постоянно и желательно монтировать удаленную файловую систему при загрузке клиента, то в графе `Options`

нужно удалить опцию noauto и добавить опцию -b. Опция -b предотвращает блокировку процесса загрузки при неудаче удаленного монтирования, переводя в фоновый режим попытки смонтировать недоступные сетевые файловые системы.

Для улучшения производительности сетевой файловой системы на сервере клиента запускается демон nfsiod, обслуживающий запросы асинхронного ввода/вывода. Как и демон nfsd на сервере, демон nfsiod на клиенте запускается в нескольких экземплярах. Это число записывается в параметр nfs_client_flags="-n 6". По умолчанию число демонов так же равно 4.

Демон автоматического монтирования

Еще удобнее подключать удаленные NFS-ресурсы с помощью демона автоматического монтирования amd. Он поддерживает монтирование "по требованию", т. е. тогда, когда удаленный ресурс кому-то понадобится. Для запуска демона при старте системы достаточно в конфигурационный файл /etc/rc.conf добавить параметр amd_enable="YES" и, если нужно, строку опций, с которыми демон amd стартует, т. е. amd_flags.

Начальное значение параметра amd_flags:

```
-a /.amd_mnt -l syslog /host/ /etc/amd.map /net /etc/amd.map
```

Файл /etc/amd.map содержит правила и опции монтирования удаленных систем. В вышеприведенном примере монтирование выполняется в каталогах /host и /net по одной схеме, т. к. для обоих каталогов используется один и тот же файл привязки /etc/amd.map:

```
# $FreeBSD: src/etc/amd.map,v 1.8 1999/09/13 17:09:07 peter Exp $
#
/defaults  type:=host;fs:=${autodir}/${rhost}/host;rhost:=${key}
*          opts:=rw,grpid,resvport,vers=2,proto=udp,nosuid,nodev
```

Опция -a задает каталог действительных точек монтирования, в данном случае это /.amd_mnt. Остальные файловые структуры создаются символическими ссылками.

Демон автоматического монтирования не обращается к файлу /etc/fstab, точки монтирования сетевых файловых систем определяются динамически по правилам, заданным в его основном конфигурационном файле и в файлах привязки автоматического монтирования, которых может быть несколько. По умолчанию конфигурационным файлом является /etc/amd.conf; файл привязки по умолчанию один — /etc /amd.map.

Тогда когда демон amd смонтирует удаленную файловую систему server_host:/home как локальный каталог /host/server_host/home, то при попытке перейти в него (cd /host/server_host/home) или прочитать его ог-

лавление (`ls /host/server_host/home`) ненужные удаленные ресурсы освобождаются автоматически.

Безопасность и NFS

Сетевая файловая система представляет серьезную угрозу безопасности как для сервера, так и для клиента. Самая большая опасность заключается в получении системой клиента привилегированного доступа к экспортируемым файлам. Даже если привилегированный доступ на экспортируемые файловые системы не предоставляется, угроза для сервера всегда существует. Всегда следует тщательно продумывать политику экспортирования и следить за состоянием файла /etc/exports.

Сервер Samba

Протокол NFS — прекрасное средство разделения файловых ресурсов для UNIX-машин, однако для совместной работы с файлами в другой среде оно не подходит. Подавляющее большинство систем персонального пользования работают под управлением разных версий ОС Windows, которые практически не поддерживают протокол NFS, продвигая собственные средства разделения ресурсов. В этой ситуации, весьма напоминающей положение Магомета и горы, сообщество разработчиков ОС UNIX поступило, как Магомет, пойдя на встречу пользователям ОС Windows и создав систему Samba (**http://www.samba.org**). Первоначально проект разрабатывал Эндрю Триджелл (Andrew Tridgell), теперь этот проект поддерживается сообществом разработчиков UNIX. Пакет Samba дает UNIX-серверам возможность участвовать в процессах разделения ресурсов ОС Windows, используя протоколы Server Message Block (SMB) и Common Internet File System (CIFS). Пакетом поддерживается поиск доступных ресурсов, защита подключений по принципам доменной организации ресурсов и подключения пользователей, а также услуги сетевых служб печати. Пакет Samba выпускается для многих клонов ОС UNIX, например, ОС FreeBSD 4.4 включает в виде порта версию 2-0.10 сервера Samba.

Другой порт, реализующий файловую систему SMBFS (Server Message Block File System), обеспечивает FreeBSD клиентскую часть сетевой среды ОС Windows.

Особенности протокола SMB

Преимущество протокола SMB перед NFS заключается в поддержке автоматического выявления серверов (browsing). Список всех доступных серверов, наблюдаемый при открытии окна **Сетевое окружение** в среде ОС Windows, является результатом широковещательного опроса, в процессе которого ка-

ждый хост периодически запрашивает "главный браузер", обнаруживая свое присутствие в сети. Все машины, присутствующие в сети, строят свои списки просмотра на основе данных этой широковещательной рассылки. Компьютеры в списке представляются своими "именами протокола NetBIOS", длина которых не должна превышать 15 символов. В других ОС, в частности, в ОС FreeBSD, в качестве имен протокола NetBIOS используются имена хоста, сокращенные при необходимости до 15 символов. Недостатком протоколов SMB и NetBIOS является ориентация на широковещательные запросы, что ограничивает сферу их действия ближайшим маршрутизатором. Для связи зон совместного использования ресурсов ОС Windows в различных сетях используется протокол Windows Internet Name Service (WINS), отчасти устраняющий эту проблему.

Доступ к общим ресурсам протокола SMB контролируется на нескольких уровнях. Самый верхний уровень ограничивает доступ содержимого любых файлов на базе IP-адреса или пароля. Кроме этого, для каждого отдельного ресурса могут быть установлены ограничения прав доступа на уровне хостов и паролей. Аутентификация пользователей по паролю может осуществляться централизованно (доменная организация защиты) или автономно тем хостом, который предоставляет ресурс.

Система Samba позволяет ограничивать доступ на всех этих уровнях, выполнять функции "главного браузера" (в контексте рабочей группы) или контроллера домена.

Установка и конфигурирование сервера Samba

В пакет Samba-2.0.10 ОС FreeBSD 4.4 входит около 400 файлов — значительную часть из них составляют страницы руководств и разного рода документация в формате HTML. Все файлы размещаются в подкаталогах /usr/local.

Средства администрирования находятся в /usr/local/bin:

- /usr/local/bin/addtosmbpass;
- /usr/local/bin/convert_smbpasswd;
- /usr/local/bin/make_printerdef;
- /usr/local/bin/make_smbcodepage;
- /usr/local/bin/make_smbpasswd;
- /usr/local/bin/make_unicodemap;
- /usr/local/bin/nmblookup;
- /usr/local/bin/rpcclient;
- /usr/local/bin/smbclient;
- /usr/local/bin/smbpasswd;

- /usr/local/bin/smbspool;
- /usr/local/bin/smbstatus;
- /usr/local/bin/smbtar;
- /usr/local/bin/testparm;
- /usr/local/bin/testprns.

Демоны помещаются в каталоге /usr/local/sbin:

- /usr/local/sbin/nmbd;
- /usr/local/sbin/smbd;
- /usr/local/sbin/swat.

Кодовые страницы — в каталоге /usr/local/etc/codepages, документация (кроме системных руководств) — в каталоге /usr/local/share/doc/samba. Примеры конфигурационных файлов и пояснения к ним — в каталоге /usr/local/share/examples/samba. В каталоге /usr/local/share/swat — документация к демону swat — поддержке графического интерфейса конфигурирования демона smbd. В файл /usr/local/private/smbpasswd записываются пароли пользователей системы Samba.

Заготовки для конфигурационного файла демона Samba и для стартового скрипта — файлы /usr/local/etc/rc.d/samba.sh.sample и /usr/local/etc/smb.conf.default должны быть переименованы или скопированы в /usr/local/etc/rc.d/samba.sh и /usr/local/etc/smb.conf соответственно. В самом простом варианте запуска Samba-сервера в файле /usr/local/etc/smb.conf необходимо исправить параметр `workgroup`, поставив название рабочей группы или домена, в которую входит сервер:

```
#workgroup = имя домена или рабочей группы NT
workgroup = ITGROUP
```

Теперь можно запускать демоны системы Samba:

```
#/usr/local/etc/rc.d/samba.sh start
```

Скрипт приводит в действие двух демонов: smbd — демона сервера данных, обслуживающего запросы подключенных клиентов ОС Windows, и nmbd — демона службы имен протокола NetBIOS. В отличие от nfsd, демон smbd в единственном экземпляре справляется с обслуживанием всех клиентов. Демон nmbd отвечает за преобразование имен протокола NetBIOS в IP-адреса.

Предполагается, что оба демона работают через посредство супердемона inetd, что разумно, если обращения к серверу Samba происходят от случая к случаю, нерегулярно. В конфигурационном файле /etc/inetd.conf должен быть отменен комментарий в следующих строках:

```
#netbios-ssn stream tcp nowait root /usr/local/sbin/smbd smbd
#netbios-ns dgram udp wait root /usr/local/sbin/nmbd nmbd
```

Если же сервер эксплуатируется постоянно, желательно организовать его старт в режиме демона, используя опцию -D.

Правила конфигурирования Samba-сервера весьма непросты; но файл-заготовка конфигурационного файла smb.conf.default содержит много комментариев и примеров конфигурирования различных сервисов Samba. Кроме того, есть системное руководство к файлу smb.conf (man smd.conf) и руководство к конфигурационному файлу в виде HTML-страниц. Прежде чем вносить изменения в конфигурационный файл сервера, с этими руководствами необходимо ознакомиться с тем, чтобы, меняя значения параметров, хорошо понимать, к каким последствиям это может привести.

Файл smb.conf состоит из разделов, каждый из которых отвечает за настройки либо своего уровня, либо отдельного сервиса или группы сервисов. Начало каждого раздела отмечается названием раздела в квадратных скобках: [global], [homes], [printers] и др.

В разделе [global] сосредоточены назначения глобальных параметров, определяющие поведение Samba-сервера в целом; первый из них, workgroup, определяет название рабочей группы, в которую сервер должен входить. Здесь же определяются схема аутентификации пользователей ОС Windows и способ управления доступом.

Раздел [homes] соответствует встроенному ресурсу, позволяющему пользователям PC подключаться к своим домашним каталогам на Samba-сервере. Этот сервис включен в конфигурацию по умолчанию.

```
[homes]
  comment = Home Directories
  browseable = no
  writeable = yes
```

При определении прав доступа к ресурсам система Samba руководствуется правилами, принятыми в ОС Windows NT, трудно сопоставимыми с правилами формирования прав в ОС UNIX. Так, строка browseable = no означает, что домашние каталоги пользователей недоступны тем, кто не является их владельцем. А строка writeable = yes относится уже к владельцу каталога и открывает этот каталог ему на запись.

Раздел [printers] описывает другой специальный ресурс, открывающий доступ к принтерам, доступным серверу:

```
[printers]
  comment = All Printers
  path = /var/spool/samba
  guest ok = no
  browseable = no
  writeable = no
  printable = yes
```

Чтобы разрешить всем клиентам пользоваться принтерами, в блоке параметров `[printers]` нужно изменить значение параметра `guest ok` с `no` на `yes`.

В файле smb.conf.default есть примеры настройки публичного, группового и персонального сервисов.

Уровень безопасности системы Samba определяется глобальным параметром `security`, который может принимать 4 значения: `user`, `server`, `share` и `domain`. Однако для управления доступом используются две принципиально разные схемы, которые можно определить как:

- управление доступом на уровне пользователей;
- управление доступом на уровне ресурсов.

При управлении доступом на уровне пользователей права пользователя на доступ к ресурсам определяются по имени пользователя. Клиенту достаточно успешно пройти один раз идентификацию "имя пользователя/пароль", чтобы получить доступ ко всем общим ресурсам. При управлении доступом на уровне ресурсов пароль назначается *ресурсу*. Если ресурс объявлен общедоступным (`public = yes`), пароль не потребуется, но если ресурс приписан пользователю john, то потребуется пароль пользователя john. Если пароль будет введен правильно, доступ к ресурсу предоставляется, несмотря на то, что пользователь входит в систему с именем george. Значения `user` и `share` параметра `security` определяют уровни управления доступом. Значение `server` только переадресует проверку соответствия "имя пользователя/пароль" другому серверу. Значение `domain` устанавливается для Samba-сервера, имитирующего поведение сервера домена NT.

При управлении доступом на уровне пользователей требуется соответствие имени пользователя на PC, где он проходит регистрацию по правилам ОС Windows, имени на сервере Samba, который ищет его в своем файле паролей. Имена должны либо совпадать, либо в файле /usr/local/etc/smbusers.map должно быть установлено соответствие имен ОС UNIX и Windows по следующему формату:

```
<имя_пользователя_unix> = <имя_пользователя_windows>...
```

Одному имени в ОС UNIX может быть сопоставлено несколько имен Windows, и это еще одна форма создания групповых ресурсов на сервере Samba.

При установке пакета Samba для всех пользователей системы сервера по файлу /etc/master.passwd создаются пароли в файле /usr/local/private/smbpasswd. Однако этот файл для совместимости содержит пароли систем LanManager (LANMAN) и Windows NT, выполненные в виде 16-ти символов "X", что не дает пользователю возможности зарегистрироваться в системе Samba-сервера до установки действительного пароля командой `smbpasswd`.

Другим источником трудностей при управлении доступом на уровне пользователей является шифрование паролей пользователей, т. к. система

Samba, ориентированная на ОС Windows 95 и первоначальные версии ОС Windows NT, по умолчанию не шифрует пароли пользователей от клиентов Windows. В более поздних версиях операционных систем Windows 98/ME и Windows NT4.0/2000 пароли пользователей шифруются обязательно и для клиентов этих систем Samba-сервер должен включать шифрование паролей параметром `encrypt passwords = yes` в разделе `[global]`.

Следует упомянуть, что в современные версии системы Samba включается комплекс программ и документов SWAT — Samba Web Administration Tool, предоставляющий средства конфигурирования Samba-сервера на основе Web-интерфейса. Это очень удобный, полезный, "красивый" и очень опасный сервис. Однако на него стóит взглянуть хотя бы потому, что он содержит ссылки на всю документацию к системе Samba в HTML-формате. Для этого нужно предпринять ряд действий, т. к. разработчики ОС FreeBSD сделали все возможное для предотвращения его автоматического старта. Итак, нужно внести в файл /etc/services имя `swat`:

```
swat    901/tcp
```

внести в конфигурационный файл /etc/inetd.conf строку

```
swat  stream  tcp  nowait  root  /usr/local/sbin/swat  swat
```

и перезапустить супердемон inetd:

```
# killall -HUP inetd
```

После этого при наведении HTTP-браузера на URL **http://localhost:901** можно получить начальную страницу комплекса SWAT с приглашением зарегистрироваться. Имя пользователя root и его действительный пароль позволяют успешно пройти авторизацию и производить конфигурирование Samba-сервера в графической среде со всей подручной документацией. Нужно помнить, что пароль при регистрации комплекса SWAT передается по сети без шифрования, что при его начальной установке для пользователя root очень опасно.

Безопасность сервера Samba

Сервер Samba обеспечивает работу сетевых сервисов ОС Windows, следовательно, предназначен для работы в исключительно доверительной обстановке, как это определяется сетевой политикой фирмы Microsoft. Его истинное место — в закрытой брандмауэром корпоративной сети на приватных адресах. Пожалуй, в этих условиях можно даже конфигурировать его посредством комплекса SWAT, если дополнительно закрыть 901-й порт сервера для всех адресов, кроме 127.0.0.1, и переопределить принадлежность файла smb.conf специальному пользователю без командного интерпретатора и домашнего каталога. Тогда под угрозой оказывается только благополучие Samba-сервера, а не всей системы.

Поддержка DHCP

Dynamic Host Configuration Protocol (DHCP) используется для передачи динамически назначаемых IP-адресов и другой информации о конфигурации сети клиентам DHCP-сервера. Динамические адреса, выделяемые DHCP-серверами на непродолжительное время, дают возможность компьютерам оперативно перемещаться из одной сети в другую, избегая назначения "мертвых" IP-адресов и имен. Это экономит адресное пространство и административные затраты на переназначение "блуждающих" адресов. Протокол DHCP также позволяет подключать к сети большое число временных клиентов с незначительным количеством IP-адресов там, где клиенты присоединяются к сети через телефонные линии и модемы.

Базовое программное обеспечение ОС FreeBSD 4.4 включает клиента протокола DHCP — dhclient. DHCP-сервер поставляется в комплекте портов и может быть легко установлен.

Основы протокола DHCP

Толчком к разработке протокола DHCP послужила проблема оперативного конфигурирования компьютеров в сетях, число хостов в которых составляло сотни, а то и тысячи клиентов, оснащенных стеками TCP/IP. К тому времени уже существовал стандарт пересылки IP-адресов (и загрузочных файлов) с серверов загрузки на отдельные рабочие станции. Это был Bootstrap Protocol (BOOTP, RFC 951), разработанный для поддержки процессов загрузки по сети бездисковых рабочих станций и устройств типа сетевых принтеров. Однако протокол BOOTP не соответствовал поставленной задаче, т. к. IP-адреса назначались статически, а таблицы разрешения аппаратных адресов велись вручную. Требовалось такое средство, которое бы динамически распределяло адреса между рабочими станциями по мере необходимости, не допуская повторения и "простоя" адресов. Инициатором разработки такого средства была фирма Microsoft, и в результате ее усилий появился протокол DHCP (RFC 1531, 1532, 1541 и 1542).

Клиентский компьютер, настроенный на получение параметров IP от DHCP-сервера при каждом перезапуске пытается связаться с DHCP-сервером своей локальной сети. Клиент либо запрашивает новые параметры конфигурации сети, либо пытается подтвердить те, которые он получил ранее. В ходе сеанса связи клиент направляет свои пакеты на 67-й порт, сервер отвечает клиенту на его 67-й порт. В качестве транспортного протокола используется UDP, а на уровне приложений используются пакеты протокола BOOTP. В процессе "переговоров" с сервером клиент может принять предложенный сервером адрес, отклонить его или заявить об освобождении используемого им адреса. Сервер может предложить адрес, подтвердить или не подтвердить назначение принятого клиентом адреса. В тех случаях, когда

клиент не может установить связь ни с одним сервером, он имеет право использовать полученные ранее параметры, пока не истечет срок их действия.

Настройка клиента DHCP

Для работы клиента DHCP в ОС FreeBSD необходимо иметь ядро, поддерживающее устройство Berkeley Packet Filter (BPF). Ядро GENERIC включает поддержку этого устройства; чтобы убедиться в этом, нужно найти в файле конфигурации ядра строку

```
device  bpf
```

Устройство BPF используется для работы анализаторов IP-пакетов в сети и дает возможность считывать информацию о "чужих" пакетах, переводя ядро системы в режим "promiscuous mode".

Чтобы включить поддержку клиента DHCP, необходимо указать интерфейс, IP-адрес которого будет назначаться по протоколу DHCP. Если, например, для этого предполагается использовать интерфейс ep0, то в файл /etc/rc.conf необходимо внести строку

```
ifconfig_ep0="DHCP"
```

Других назначений для DHCP-интерфейса в файле не должно быть. Систему следует перезагрузить, чтобы установки протокола DHCP были учтены. Теперь при каждом старте системы программа dhclient будет пытаться получить во временное пользование динамически назначаемый IP-адрес и другие параметры сети. Программа dhclient имеет несколько опций, изменяющих ее поведение. Их можно задать в параметре `dhcp_flags = "flags"` файла /etc/rc.conf. Другие параметры, определяющие сценарий запроса IP-адреса, задаются в файле /etc/dhclient.conf, образец которого приведен далее (листинг 27.7).

Листинг 27.7. Пример конфигурационного файла /etc/dhclient.conf

```
send host-name "andare.fugue.com";
send dhcp-client-identifier 1:0:a0:24:ab:fb:9c;
send dhcp-lease-time 3600;
supersede domain-name "fugue.com home.vix.com";
prepend domain-name-servers 127.0.0.1;
request subnet-mask, broadcast-address, time-offset, routers,
  domain-name, domain-name-servers, host-name;
require subnet-mask, domain-name-servers;
timeout 60;
retry 60;
```

```
reboot 10;
select-timeout 5;
initial-interval 2;
script "/etc/dhclient-script";
media "-link0 -link1 -link2", "link0 link1";
reject 192.33.137.209;

alias {
  interface "ep0";
  fixed-address 192.5.5.213;
  option subnet-mask 255.255.255.255;
}

lease {
  interface "ep0";
  fixed-address 192.33.137.200;
  medium "link0 link1";
  option host-name "andare.swiftmedia.com";
  option subnet-mask 255.255.255.0;
  option broadcast-address 192.33.137.255;
  option routers 192.33.137.250;
  option domain-name-servers 127.0.0.1;
  renew 2 2000/1/12 00:00:01;
  rebind 2 2000/1/12 00:00:01;
  expire 2 2000/1/12 00:00:01;
}
```

Сервер DHCP

Свободно распространяемая версия сервера DHCP включена в набор портов ОС FreeBSD. Это пакет isc-dhcp3 раздела net. После установки пакета образец стартового скрипта будет помещен в каталог /usr/local/etc/rc.d в файл с названием isc-dhcpd.sh.sample. Если его переименовать хотя бы в isc-dhcpd.sh, то демон службы динамического распределения адресов будет пытаться стартовать при каждой перезагрузке. Как и все стартовые скрипты локальных служб, файл isc-dhcpd.sh может управляться параметрами `start`, `stop` и `restart`. Если файл конфигурации /usr/local/etc/dhcpd.conf существует и не содержит ошибок, то демон dhcpd благополучно начнет работу. Правильный конфигурационный файл демона можно получить, скопировав

с именем dhcpd.conf образец файла /usr/local/etc/dhcpd.conf.sample и внеся в него необходимые изменения (листинг 27.8).

Листинг 27.8. Пример файла конфигурации демона dhcpd

```
# dhcpd.conf
#
# Sample configuration file for ISC dhcpd
#

# option definitions common to all supported networks...
option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
#authoritative;

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).
log-facility local7;

# No service will be given on this subnet, but declaring it helps the
# DHCP server to understand the network topology.

subnet 10.152.187.0 netmask 255.255.255.0 {
}

# This is a very basic subnet declaration.

subnet 10.254.239.0 netmask 255.255.255.224 {
  range 10.254.239.10 10.254.239.20;
  option routers rtr-239-0-1.example.org, rtr-239-0-2.example.org;
}

# This declaration allows BOOTP clients to get dynamic addresses,
# which we don't really recommend.
```

```
subnet 10.254.239.32 netmask 255.255.255.224 {
  range dynamic-bootp 10.254.239.40 10.254.239.60;
  option broadcast-address 10.254.239.31;
  option routers rtr-239-32-1.example.org;
}

# A slightly different configuration for an internal subnet.
subnet 10.5.5.0 netmask 255.255.255.224 {
  range 10.5.5.26 10.5.5.30;
  option domain-name-servers ns1.internal.example.org;
  option domain-name "internal.example.org";
  option routers 10.5.5.1;
  option broadcast-address 10.5.5.31;
  default-lease-time 600;
  max-lease-time 7200;
}

# Hosts which require special configuration options can be listed in
# host statements.   If no address is specified, the address will be
# allocated dynamically (if possible), but the host-specific information
# will still come from the host declaration.

host passacaglia {
  hardware ethernet 0:0:c0:5d:bd:95;
  filename "vmunix.passacaglia";
  server-name "toccata.fugue.com";
}

# Fixed IP addresses can also be specified for hosts.  These addresses
# should not also be listed as being available for dynamic assignment.
# Hosts for which fixed IP addresses have been specified can boot using
# BOOTP or DHCP.  Hosts for which no fixed address is specified can only
# be booted with DHCP, unless there is an address range on the subnet
# to which a BOOTP client is connected which has the dynamic-bootp flag
# set.
host fantasia {
  hardware ethernet 08:00:07:26:c0:a5;
```

```
  fixed-address fantasia.fugue.com;
}

# You can declare a class of clients and then do address allocation
# based on that.  The example below shows a case where all clients
# in a certain class get addresses on the 10.17.224/24 subnet, and all
# other clients get addresses on the 10.0.29/24 subnet.

class "foo" {
  match if substring (option vendor-class-identifier, 0, 4) = "SUNW";
}

shared-network 224-29 {
  subnet 10.17.224.0 netmask 255.255.255.0 {
    option routers rtr-224.example.org;
  }
  subnet 10.0.29.0 netmask 255.255.255.0 {
    option routers rtr-29.example.org;
  }
  pool {
    allow members of "foo";
    range 10.17.224.10 10.17.224.250;
  }
  pool {
    deny members of "foo";
    range 10.0.29.10 10.0.29.230;
  }
}
```

Конфигурационный файл демона протокола DHCP — dhcpd — описывает структуру обслуживаемой сети, динамику обмена адресов и политику обслуживания подсетей. Все его директивы разделяются на две категории: параметры и декларации. Параметры определяют количественные и качественные характеристики сервера, декларации же описывают топологию сети и политику выделения адресов для различных подсетей. Все декларации придется переписать, а параметры — серьезно обдумать, т. к., скорее всего, политика и правила в реальной сети будут совсем другими. Главными дек-

ларациями являются декларации подсетей с указанием диапазона выделяемых для них IP-адресов:

```
subnet 10.254.239.0 netmask 255.255.255.224 {
  range 10.254.239.10 10.254.239.20;
  option routers rtr-239-0-1.example.org, rtr-239-0-2.example.org;
}
```

Протокол DHCP предусматривает также обслуживание клиентов по протоколу BOOTP:

```
subnet 10.254.239.32 netmask 255.255.255.224 {
  range dynamic-bootp 10.254.239.40 10.254.239.60;
  option broadcast-address 10.254.239.31;
  option routers rtr-239-32-1.example.org;
}
```

Послесловие

Обзор владений леди Free подошел к концу. Надеюсь, вы не очень пожалели о затраченном на него времени. Вне зависимости от того, будете ли вы использовать FreeBSD в своей практической работе или нет, совсем не вредно ознакомиться с тем, что происходит в параллельных мирах.

Ведь ценность FreeBSD не только в ней как таковой. Эта система оказала в свое время едва ли не решающее влияние на весь мир UNIX-систем. И продолжает его оказывать: симптоматично, что многие новые (и прогрессивные!) дистрибутивы Linux возникли под явным идейным влиянием FreeBSD.

Так, дистрибутив Gentoo (**http://www.gentoo.org**), разрабатываемый упомянутым чуть выше Дэниэлом Роббинсом, заимствовал из FreeBSD и плоскую структуру стартовых файлов, и даже систему портов, и, по признанию разработчика, многое, многое другое.

Тем более, что, как мы пытались показать, сама по себе FreeBSD вполне самодостаточная система. Более чем подходящая для решения очень широкого круга задач — перечислять их не будем, дабы не пересказывать содержание книги.

Предметный указатель

A

a2kfcnv 417
AbiWord 490
aumix 411

B

bash 324, 343
Blackbox 439
bladeenc 414
BTX loader 513

C

cdparanoia 413
cdplay 410
csh 324

D

d1489 417
dagrab 413
deco 345

F

file 284
FreeBSD:
- дистрибуция 30
- история 22
- лицензия 32
- номенклатура накопителей 55
- определение 19
- требования к аппаратуре 62
- установка 72
- файловая система 53

fromdos 417
fromwin 417

G

GIMP 503
Gnumeric 501
gPhoto 507
GQview 507
grip 508
GRUB 516

I

IceWM 433
inode 186

K

keymap 246
Ksnapshot 507

L

lame 414
LILO 514
links 409
linux-base 520
lynx 408
Lyx 493

M

manual pages 47
mapchan 419
Midnight Commander 344

mp32ogg 415
mp3blaster 411
mpg123 410

O

ogg 411
ogg123 412
oggenc 412
ogginfo 412

P

Partition 57
pm-пакеты 519

R

rmdir 289
rusconv 417

S

scr2png 405
sh 342
Shell 323
slice 57
Soft Updates 211
syscons 229
sysinstall (программа установки и настройки) 73

T

tcsh 324, 343
todos 417
towin 417

V

vi 371
vidcontrol 234
vorbis-tools 412

W

waveplay 414
wavplay 414
WindowMaker 431

X

X Window System 422
XFce 436
XFree86 60
  настройка 125
  установка 124
xmms 507
XV 507

Z

zsh 325, 343, 349

А

Автодополнение 327
Автокоррекция команд 329
Аргумент 259

Б, В, Г

Буфер истории команд 332
Встроенные команды оболочки 329
Группировка команд 269

И

Интегрированные визуальные (называемые также не совсем точно графическими) среды 435
Интегрированные среды 435
Интерфейс командной строки 257
История команд 328

К

Каталоги 190
Командная оболочка 42
Команды 43, 258
banner 315
bc 316
bzcat 304
bzip2 294
cal 317
cat 276, 300, 308
cd 279
chgroup 289
chmod 289
chown 289
cmp 304
colrm 318
column 319
cp 285
cpio 290
dd 286
df 314
diff 304
echo 313
egrep 310
expand 320
fgrep 310
find 295
fmt 320
fold 320
grep 310
gzip 293
head 303
history 332
less 301
lesskey 302
ln 278
locate 280
ls 281
mkdir 277
mkfifo 278
mknod 278
more 301
mv 287
ncal 318
od 320
patch 308
printenv 314
pwd 279
rm 288
sort 321
split 309
stat 284
tail 303
tar 290
tee 277
touch 276, 289
uniq 322
wc 322
whereis 280
which 279
write 322
zcat 304
zcmp 308
zdiff 308
zmore 304
Конвейеризация команд 269
Консоль:
виртуальная 41
системная 40

М

Менеджеры окон 428
Метасимволы 263

Н

Наращиваемый поиск в истории команд 333

О

Оконная система X 60, 422
  установка 123, 422
Опция 259

П

Пакеты:
  установка программ 109
Перенаправление ввода/вывода 266
Порты:
  установка программ 115
Права доступа 59
Процесс 171
  атрибуты 174
Псевдонимы 338

Р

Разделы дисковые 57
Регулярные выражения 262
Редактирование командной строки 328

С

Слайс 57
Ссылки:
  жесткие 187
  символические 191
Сценарии оболочки 270

Т

Текстовые редакторы 366
  ee 50
  jed 379
  joe 384
  le 380
  lpe 368
  NEdit 447
  vi 371
  Vim 372

У

Управление задачами 329, 335
Управляющие последовательности 326
Учетные записи 217

Ф

Файлы 189
  классификация 189
  обычные 189
Файлы устройств 190

Я

Ядро:
  регенерация 136
  функции 134

Алексей Федорчук
Алексей Торн

FreeBSD

Установка, настройка, использование

В книге приводится описание операционной системы FreeBSD, которая сегодня является наиболее зрелой и технологически совершенной среди семейства открытых и свободных UNIX-подобных систем. FreeBSD снискала широкую популярность в сфере интернет- и интранет-технологий, сетевых и коммуникационных служб. Однако в неменьшей степени она пригодна для использования в качестве универсальной настольной системы общего назначения. Именно эту, менее освещенную особенность системы FreeBSD авторы и постарались донести до читателя.

- Командная строка и команды
- Текстовые редакторы
- Консольная графика и мультимедиа
- Маршрутизация и защита сети

ИНТЕРНЕТ-МАГАЗИН
www.computerbook.ru

ISBN 5-94157-200-X

Уровень пользователя	Средний/высокий
Категория	Операционные системы

9 785941 572007