Справочное руководство

Limited Edition 2005

http://www.mandrakesoft.com

Справочное руководство: **Limited Edition 2005**
Опубликовано Апрель 2005

NeoDoc (`http://www.neodoc.biz`)Camille Bégnis, Christian Roy, Fabian Mandelbaum, Roberto Rosselli del Turco, Marco De Vitis, Alice Lafox, John Rye, Wolfgang Bornath, Funda Wang, Patricia Pichardo Bégnis, Debora Rejnharc Mandelbaum, Mickael Scherer, Jean-Michel Dault, Lunas Moon, Céline Harrand, Fred Lepied, Pascal Rigaux, Thierry Vignaud, Giuseppe Ghibò, Stew Benedict, Francine Suzon, Indrek Madedog Triipus, Nicolas Berdugo, Thorsten Kamp, Fabrice Facorat, Xiao Ming, Snature , Guylhem Aznar, Pavel Maryanov, Annie Tétrault

Юридическое замечание

Тексты лицевых обложек

```
   Mandrakesoft April 2005
http://www.mandrakesoft.com/
Copyright © 1999-2005 by Mandrakesoft S.A. and Mandrakesoft Inc.
```

Главы, приведенные в таблице ниже, защищены отдельной лицензией. Проконсультируйтесь с таблицей и ссылками для получения более подробной информации об этих лицензиях.

Название главы	**Оригинальный копирайт**	**Лицензия**
Гл. 12	Benjamin Drieu , **APRIL** (`http://www.april.org/`)	GNU General Public License (GPL) (`http://www.gnu.org/copyleft/gpl.html`)

Утилиты, использованные при создании этого руководства

Это руководство написано в XML DocBook. Для управления совокупностью всех задействованных в проекте файлов использовалась Система совместного создания контента Borges (C3S) (`http://sourceforge.net/projects/borges-dms`). Исходные XML-файлы обрабатывались в xsltproc, `openjade` и `jadetex` с использованием собственной версии таблиц стилей Нормана Уолша (Norman Walsh). Снимки экрана получены при помощи `xwd` или GIMP и конвертированы при помощи

`convert` (из пакета ImageMagick). Все эти программы являются свободным программным обеспечением и большинство из них доступно в вашем дистрибутиве Mandrakelinux.

Содержание

Список таблиц

Предисловие

1. О Mandrakelinux

Mandrakelinux - это дистрибутив GNU/Linux, поддерживаемый компанией **MandrakeSoft** S.A., которая родилась в Интернете в 1998 году. Главной ее целью было и остается предоставление простой в использовании и дружественной системы GNU/Linux. Две опоры **MandrakeSoft** - это открытые исходные тексты и совместная работа.

1.1. Связь с сообществом Mandrakelinux

Следующие разнообразные Интернет-ссылки указывают на различные ресурсы, связанные с Mandrakelinux. Если вы хотите побольше узнать о компании **MandrakeSoft**, зайдите на наш веб-сайт (`http://www.mandrakesoft.com/`). Вы также можете заглянуть на веб-сайт дистрибутива Mandrakelinux (`http://www.mandrakelinux.com/`) и всего, что к нему относится.

Mandrakeexpert (`http://www.mandrakeexpert.com/`) - это платформа технической поддержки от **MandrakeSoft**. Она предлагает поделиться опытом, основываясь на доверии и вознаграждении других за их содействие.

Мы также приглашаем вас подписаться на различные списки рассылок (`http://www.mandrakelinux.com/ru/flists.php3`), в которых сообщество Mandrakelinux проявляет свою активность и проницательность.

Не забудьте также заглянуть на нашу страницу безопасности (`http://www.mandrakesoft.com/security`). На ней собраны все материалы, касающиеся безопасности дистрибутивов Mandrakelinux. Здесь вы найдете советы по безопасности и ошибкам, а также процедуры по обновлению ядра, различные списки рассылок по безопасности, на которые вы можете подписаться и Mandrakeonline. В общем все, о чем должен знать любой системный администратор или пользователь, заинтересованный в обеспечении безопасности.

1.2. Вступайте в Клуб

MandrakeSoft предлагает широкий спектр привилегий через свой Mandrakeclub (`http://www.mandrakeclub.com`):

- загрузка коммерческого программного обеспечения, обычно доступного только в коробочных версиях, такого как драйвера оборудования, коммерческие приложения, freeware и демо-версии;
- право голоса за новое программное обеспечение через систему голосования за RPM на добровольных основах;
- доступ к более чем 50 000 RPM-пакетов для всех дистрибутивов Mandrakelinux;
- получение скидок на продукты и услуги в Mandrakestore (`http://store.mandrakesoft.com`);
- доступ к списку лучших зеркал, доступных только для членов Клуба;
- чтение форумов и статей на нескольких языках.
- доступ к Базе знаний (`https://kb.mandrakeclub.com`) **MandrakeSoft** - сайт на базе wiki, содержащий документацию на разнообразные темы: администрирование, взаимодействие, поиск и устранение неисправностей и другие;
- чат с разработчиками Mandrakelinux в Club Chat (`https://www.mandrakeclub.com/user.php?op=clubchat`);
- повышение своих знаний GNU/Linux с помощью курсов электронного обучения от **MandrakeSoft** (`http://campus.mandrakesoft.com`)

Финансируя **MandrakeSoft** через Mandrakeclub, вы делаете непосредственный вклад в улучшение дистрибутива Mandrakelinux и помогаете нам предоставлять нашим пользователям самую лучшую настольную GNU/Linux-систему.

1.3. Подписка на Mandrakeonline

MandrakeSoft предлагает очень удобный способ для поддержания вашей системы в актуальном состоянии в автоматическом режиме, оберегая ее от ошибок и дыр в безопасности. Посетите веб-сайт Mandrakeonline (`https://www.mandrakeonline.net/`), чтобы больше узнать об этой услуге.

1.4. Приобретение продуктов Mandrakesoft

Пользователи Mandrakelinux могут приобрести продукты в нашем онлайновом интернет-магазине Mandrakestore (`http://store.mandrakesoft.com/`). В нем вы найдете не только программное обеспечение Mandrakelinux, операционные системы и загрузочные "live" CD (типа Move), но и также специальные заказные предложения, техническую поддержку, программное обеспечение сторонних разработчиков и под другими лицензиями, документацию, книги по GNU/Linux, а также другие полезности от **MandrakeSoft**.

1.5. Содействие Mandrakelinux

Опыт и знания многих талантливых людей, использующих Mandrakelinux, могут очень пригодиться при создании системы Mandrakelinux:

- **Сборка пакетов.** Система GNU/Linux в основном собрана из программ, загруженных из Интернета. Они должны быть собраны в пакеты для обеспечения их совместной работы.
- **Программирование.** Существует великое множество проектов, непосредственно поддерживаемых **MandrakeSoft**'ом: выберите для себя самый нужный проект и предложите свою помощь главным разработчикам.
- **Интернационализация.** Вы можете помочь нам с переводом веб-страниц, программ и соответствующей документации.

Загляните на страницу разрабатываемых проектов (`http://www.mandrakesoft.com/labs/`), чтобы больше узнать о том, как вы можете посодействовать развитию Mandrakelinux.

2. Об этом Справочном руководстве

Это *Справочное руководство* предназначено для людей, желающих лучше понять свою систему Mandrakelinux, и которые хотят полностью задействовать ее огромные возможности. Мы надеемся, что после прочтения этого руководства вам будет проще администрировать свою GNU/Linux-машину. Здесь представлен обзор трех частей, из которых состоит руководство, с кратким описанием составляющих их глав:

- В первой части (*Система Linux*) мы познакомим вас с командной строкой и различные вариантами ее использования. Мы также обсудим основы редактирования текста, что является неотъемлемой частью GNU/Linux.

  В первой главе (Гл. 1) мы ознакомим вас с понятием UNIX®, отдавая большее предпочтение миру GNU/Linux. Мы рассмотрим стандартные утилиты для работы с файлами, а также некоторые полезные возможности, предоставляемые shell'ом. Затем следует дополнительная глава (Гл. 2), в которой мы подробно рассмотрим управление жесткими дисками в GNU/Linux, а также их разметку. Очень важно, чтобы полностью разобрались с этими понятиями перед тем, как перейти к главе Гл. 3.

  В следующей главе рассматривается редактирование текста (Гл. 4). Так как большинство конфигурационных файлов UNIX® представлют собой текстовые файлы, вам рано или поздно придется их редактировать в ***текстовом редакторе*** . Вы узнаете, как пользоваться двумя наиболее известными текстовыми редакторами в мирах UNIX® и GNU/Linux: навороченным Emacs и старым добрым Vi, написанным Биллом Джоем (Bill Joy) еще в 1976 году.

  После этого вы будете в состоянии осуществлять базовое обслуживание своей системы. Следующие две главы демонстрируют практическое использование командной строки (Гл. 5) и в общих чертах управление процессами (Гл. 6).

- В части под названием *Linux изнутри* мы затронем ядро Linux и архитектуру файловой системы.

В Гл. 7 мы изучим организацию файлового дерева. Системы UNIX® имеют тенденцию очень сильно разрастаться, но при этом каждый файл находится на своем месте в определенном каталоге. После прочтения этой главы вы узнаете о том, где искать файлы в зависимости от их роли в системе.

Затем мы рассмотрим ***файловые системы*** и ***точки монтирования*** (Гл. 8). Мы дадим определения обоим этим терминам, а также поясним их на практических примерах.

Следующая глава посвящена файловым системам GNU/Linux (Гл. 9). После представления доступных файловых систем мы рассмотрим типы файлов и некоторые дополнительные понятия и утилиты типа inode и pipe. Эту главу мы заканчиваем рассмотрением принципов, лежащих в основе RAID (Redundant Array of Inexpensive Disks, матрица независимых дисковых накопителей с избыточностью) и EVMS (Enterprise Volume Management System), а также их базовой настройкой. В следующей главе (Гл. 10) представлена специальная (и виртуальная) файловая система GNU/Linux - `/proc`.

Следующая глава (Гл. 11) рассказывает о процедуре загрузки Mandrakelinux и о том, как ее эффективно использовать.

- В разделе *Продвинутое использование* мы рассмотрим действия, выполнять которые каждый день нет необходимости. Мы расскажем вам о этапах, необходимых для сборки и установки свободного программного обеспечения из исходных кодов в главе Гл. 12. Прочтение этой главы должно побудить вас попробовать сделать это, даже если на первый взгляд это выглядит пугающе. И, в заключение, информация, представленная в последней главе (Гл. 13), поможет вам добиться полной автономности GNU/Linux. После прочтения теории, изложенной в этой главе, и применения ее на практике вы сможете начать переводить пользователей Windows® на GNU/Linux (даже если вы еще не начали это делать!).

3. Примечание редактора

В философии open-source добровольное содействие всегда приветствуется! Обновление пакета документации к Mandrakelinux - это серьезная задача. Ваша помощь может быть выражена разными способами. Фактически команда документации постоянно ищет талантливых добровольцев, которые могли бы оказать помощь в выполнении следующих задач:

- написание или обновление;
- перевод;
- литературное редактирование;
- XML/XSLT-программирование.

Если у вас есть много времени, вы можете написать или обновить целую главу. Если вы говорите на иностранном языке, вы можете помочь нам с переводом наших руководств. Если у вас есть идеи о том, как улучшить содержимое - дайте нам знать. Если вы занимаетесь программированием и хотели бы помочь нам усовершенствовать Систему совместного создания контента Borges (C3S) (`http://sourceforge.net/projects/borges-dms`) - присоединяйтесь. И не стесняйтесь сообщать нам об опечатках, если вы их найдете, чтобы мы могли исправить их!

За любой информацией о проекте документации Mandrakelinux обращайтесь, пожалуйста, к администратору документации (`mailto:documentation@mandrakesoft.com`) или посетите веб-страницу Проекта документации Mandrakelinux (`http://www.mandrakelinux.com/en/doc/project/`).

Пожалуйста, обратите внимание, что с июня 2004 написание документации Mandrakelinux и разработка проекта Borges ведется компанией NeoDoc (`http://www.neodoc.biz`).

4. Соглашения, используемые в этой книге

4.1. Соглашения по набору текста

Чтобы четко выделить специальные слова в потоке текста, мы используем различные виды оформления. В приведенной ниже таблице показаны примеры для каждого из специальных слов или группы слов с их оформлением и описанием их значения.

Форматированный пример	**Значение**
inode	Используется для выделения технических терминов, описываемых в Прил. А.
`ls -lta`	Используется для команд и их аргументов. Также применяется для опций и имен файлов (смотрите раздел Разд. 4.2.1).
ls(1)	Ссылка на страницу руководства (`man`). Чтобы прочитать страницу, просто наберите в командной строке `man 1 ls`.
`$ ls *.pid`	Форматирование, используемое для снимков текстовых областей, которые вы можете увидеть на своем экране, включая результаты выполнения команд, распечатки программы и т.д.
`localhost`	Буквенные данные, не подходящие под описание ни одной из ранее определенных категорий. Например, ключевое слово, взятое из конфигурационного файла.
OpenOffice.org	Определяет названия приложений. В зависимости от контекста, приложения и название команды могут быть одинаковыми, но иметь разное оформление. Например, большинство команд пишется в нижнем регистре, в то время как приложения обычно начинаются с большой буквы.
Файлы	Обозначает пункты меню или метки графического интерфейса. Подчеркнутая буква (если таковая присутствует) информирует вас о наличии “горячей” клавиши для быстрого доступа к пункту меню, который может получен путем нажатия клавиши **Alt** плюс сама буква.
SCSI-шина	Указывает на часть компьютера или сам компьютер.
Le petit chaperon rouge	Обозначает слова на иностранном языке.
Предупреждение!	Зарезервировано для особых предупреждений, чтобы подчеркнуть важность слов. Читается вслух.

Выделяет примечание. Обычно в нем содержится дополнительная информация об определенном контексте.

Обозначает подсказку. Это может быть общий совет о том, как выполнить определенное действие, или подсказки о полезных возможностях, которые могут облегчит вам жизнь, например, “быстрые” клавиши.

Будьте очень осторожны при встрече этого значка. Он всегда означает, что будет рассмотрена очень важная информация по определенной теме.

4.2. Основные соглашения

4.2.1. Краткий обзор команд

В приведенном ниже примере показаны символы, которые вы увидите при описании автором аргументов команды:

```
команда <не буквенный аргумент> [--option={арг1,арг2,арг3}] [опциональный арг. ...]
```

Эти соглашения являются стандартными и вы можете встретить их где угодно, например, в страницах руководства.

Знаки “<” (меньше) и “>” (больше) выделяют **обязательный** аргумент, который вы не копируете один в один, а заменяете на нужное вам значение. Например, <имя_файла> означает реальное имя файла. Если это имя `foo.txt`, вы должны ввести `foo.txt`, но не <foo.txt> или <имя_файла>.

Квадратные скобки (“[ ]”) выделяют необязательные аргументы, которые вы можете включать или не включать в свою команду.

Троеточие (“...”) означает произвольное число аргументов.

В фигурные скобки (“{ }”) заключаются аргументы, разрешенные для этого случая. Должен быть указан один из них.

4.2.2. Специальные формы записи

Время от времени вам будет предлагаться нажать, например, клавиши **Ctrl**-**R**, что означает, что вы должны, нажав и удерживая клавишу **Ctrl**, сразу после этого стукнуть по клавише **R**. То же самое касается клавиш **Alt** и **Shift**.

Также, в отношении к меню, переход к пункту меню Файл→Перезагрузить конфигурацию пользователя (**Ctrl-R**) означает: щелкнуть по надписи Файл, находящейся в меню (обычно расположенном в левом верхнем углу окна). Затем в появившемся меню нужно кликнуть по пункту Перезагрузить конфигурацию пользователя. Кроме того, для получения того же результата, вы можете воспользоваться комбинацией клавиш **Ctrl**-**R** (как описано выше).

4.2.3. Обычные пользователи системы

Всякий раз, когда это возможно, мы используем в наших примерах двух обычных пользователей:

Queen Pingusa	queen	Это наш пользователь по умолчанию, используемый в большинстве примеров этой книги.
Peter Pingus	peter	Этот пользователь может быть создан позже системным администратором, и изредка используется, дабы разнообразить текст.

Глава 1. Основные понятия системы UNIX

Название “UNIX®” некоторым из вас вероятно уже знакомо. Возможно, вы даже уже используете UNIX® на работе, тогда, вероятно, эта глава будет вам не очень интересна.

А для тех, кто еще никогда не работал с ней, прочтение этой главы является абсолютно необходимым. Понимание концепций, которые будут здесь представлены, позволит ответить на необычайно большое количество вопросов, обычно задаваемых новичками в мире **GNU/Linux**. Подобным образом некоторые из этих понятий станут хорошим ответом на большинство проблем, с которыми вы можете столкнуться в будущем.

1.1. Пользователи и группы

Так как они оказывают непосредственное влияние на все другие понятия, в этом разделе мы познакомим вас с понятием пользователей и групп, что является чрезвычайно важным.

Linux является действительно ***многопользовательской*** системой, и чтобы пользоваться своей машиной GNU/Linux, вы должны иметь на ней ***учетную запись***. Когда во время установки вы создавали пользователя, на самом деле вы создавали учетную запись пользователя. Напомним, что вам предлагалось ввести следующие данные:

- “настоящее имя” пользователя (которое на самом деле может быть чем угодно);
- имя ***логина***;
- и ***пароль***.

Два самых важных параметра здесь - это имя логина (обычно сокращается до логин) и пароль. Вы должны знать оба этих параметра для получения доступа к системе.

Когда вы создаете пользователя, также создается и группа по умолчанию. Как мы увидим позже, группы очень полезны, если вы хотите открыть общий доступ к файлам другим людям. Группа может содержать столько пользователей, сколько пожелаете, и это разделение является обычным делом для больших систем. Например, в университете вы можете иметь по одной группе на факультет, одну группу для преподавателей и так далее. Обратное тоже верно: пользователь может быть членом одной или нескольких групп. Преподаватель математики, например, может быть членом группы преподавателей и группы студентов, с которыми он работает.

Пока что мы рассмотрели только вводную информацию, теперь давайте узнаем, как же войти в систему.

Если графический интерфейс (X) автоматически запускается при загрузке, ваш стартовый экран будет выглядеть следующим образом Рис. 1-1.

Рисунок 1-1. Сеанс входа в систему в графическом режиме

Чтобы войти в систему, вы должны сначала выбрать из списка свою учетную запись. Появится новое диалоговое окно, предлагающее вам вести свой пароль . Обратите внимание, что пароль вам придется вводить вслепую, потому что символы, вводимые в поле для пароля, будут заменяться звездочками (*). Вы также можете выбрать тип своего сеанса (оконный менеджер). Как только вы будете готовы, нажмите кнопку Вход.

Если вы в "текстовом" режиме, вам будет представлено нечто похожее на это:

```
Mandrakelinux Release 10.2 (CodeName) for i586
Kernel 2.6.10-1mdk on an i686 / tty1
[имя_машины] login:
```

Для входа в систему введите свой логин в приглашении Login: и нажмите **Enter**. Затем программа входа в систему (login) выведет приглашение Password: и будет ожидать ввода пароля. Как и при входе в систему в графическом режиме, при консольном входе вводимые вами символы на экране не только не отображаются, но и даже не заменяются звездочками.

Обратите внимание, что вы несколько раз можете войти в систему с одной и той же учетной записью в дополнительных ***консолях*** и в X. Каждый открытый вами сеанс не зависит от других, и даже имеется возможность открывать одновременно несколько сеансов X (однако это не рекомендуется, т.к. при этом расходуется много ресурсов). По умолчанию в Mandrakelinux есть шесть ***виртуальных консолей*** в дополнение к одной, зарезервированной для графического интерфейса. Вы можете переключиться в любую из них, нажав последовательность клавиш **Ctrl-Alt-F<n>**, где <n> - номер консоли, в которую вы хотите переключиться. По умолчанию графический интерфейс находится в 7-й консоли. Следовательно, чтобы переключиться во вторую консоль, вам необходимо нажать клавиши **Ctrl**, **Alt** и **F2**.

Во время установки DrakX также спрашивал у вас пароль для специального пользователя: `root`. Это системный администратор, которым вероятнее всего будете вы сами. Для безопасности вашей системы очень важно, чтобы учетная запись `root` всегда была защищена хорошим и трудным для угадывания паролем!

Если вы будете постоянно работать под ***root***'ом, то очень просто допустить ошибку, которая сделает вашу систему непригодной к использованию: одна единственная ошибка может привести к этому. В частности, если вы не установили пароль для учетной записи `root`, тогда **любой** пользователь может внести изменения в **любую** часть вашей системы (и даже другой операционной системы на вашей машине!). Очевидно, что это не самая лучшая идея.

Стоит обратить внимание, что внутренне система не идентифицирует вас по имени логина. Вместо этого она использует уникальный номер, присвоенный этому имени: *идентификатор пользователя* (User ID, UID) . Аналогично каждая группа идентифицируется по своему *идентификатору группы* (Group ID, GID), а не по имени.

1.2. Основы работы с файлами

По сравнению с Windows® и большинством других ***операционных систем***, в GNU/Linux работа с файлами организована совсем по-другому. В этом разделе мы рассмотрим большинство явных различий. Для получения дополнительной информации прочтите, пожалуйста, Гл. 9.

Основные различия являются прямым следствием того факта, что Linux - это многопользовательская система: каждый файл является исключительной собственностью одного пользователя и одной группы. Еще один момент о пользователях и группах, который мы не упомянули, состоит в том, что каждый из них владеет личным каталогом (называемым ***домашним каталогом***). Пользователь является владельцем этого каталога и всех создаваемых в нем файлов. Также обратите внимание, что с ними также ассоциируется группа, которая является основной группой, к которой принадлежит пользователь. Как было сказано ранее (см.), пользователь может быть членом нескольких групп одновременно.

Однако, если бы это было единственным понятием владения файлом, пользы от этого было бы мало. Как владелец файла, пользователь может устанавливать **права** на файлы. Эти права распределяются между тремя категориями пользователей: **владельцем** файла; всеми пользователями, являющимися членами **группы**, ассоциированной с файлом (также называемой ***группой владельца***), но не являющимися владельцами; и **остальными**, куда входят все остальные пользователи, которые не являются ни владельцами, ни членами группы владельца.

Существует три разновидности прав:

1. Права на *чтение* (`Read, r`): пользователю разрешается читать содержимое файла. По отношению к каталогу это означает, что пользователь может просмотреть его содержимое (т.е. список файлов этого каталога).

2. Права на *запись* (`Write, w`): разрешает изменять содержимое файла. По отношению к каталогу право на запись дает пользователю возможность добавлять или удалять файлы из этого каталога, даже если он не является владельцем этих файлов.
3. Права на *выполнение* (`eXecute, x`): разрешает запуск файла (обычно только исполняемые файлы имеют этот тип прав доступа). По отношению к каталогу это дает пользователю возможность ***проходить*** его, что означает войти в этот каталог или пройти сквозь него. Обратите внимание, что это отличается от доступа на чтение: вы в состоянии пройти через каталог, но прочитать его содержимое все-таки не можете!

Возможны любые комбинации этих прав. Например, вы можете разрешить только чтение файла для себя и запретить доступ для всех других пользователей. Как владелец файла вы также можете изменить его группу (только если вы член устанавливаемой группы).

Давайте рассмотрим пример файла и каталога. Ниже представлено выполнение команды `ls -l` в ***командной строке***:

```
$
ls -l
total 1
-rw-r-----   1 queen    users           0 Jul  8 14:11 a_file
drwxr-xr--   2 peter    users        1024 Jul  8 14:11 a_directory/
$
```

Результаты выполнения команды `ls -l` (слева направо):

- Первые десять символов представляют тип файла и назначенные ему права. Первый символ - это тип файла: если это обычный файл, вы увидите тире (`-`). Если это каталог, крайним левым символом будет `d`. Существуют и другие типы файлов, которые мы обсудим позже. Следующие девять символов представляют собой права доступа для данного файла. Эти девять символов на самом деле являются тремя группами из трех прав. Первая группа представляет права владельца файла; следующие три символа касаются всех пользователей, принадлежащих к группе владельца; и последние три символа относятся ко всем остальным. Знак тире (`-`) означает, что права доступа не установлены.
- Далее следует количество ссылок на файл. Позже мы увидим, что уникальный идентификатор файла - это не имя, а его номер (***номер inode***), и существует возможность иметь на диске несколько имен для одного файла. Для каталога количество ссылок имеет специальное значение, что также будет рассмотрено несколько позже.
- Следующая часть информации - это имя владельца файла и имя группы.
- И, наконец, далее показаны размер файла (в ***байтах***), время его последнего изменения и имя самого файла или каталога в самом конце строки.

Давайте поближе рассмотрим права доступа для каждого из этих файлов: сначала мы должны отбросить первый символ, представляющий тип файла, и для файла `a_file` мы получим следующие права: `rw-r-----`. Ниже представлена схема организации прав:

- первые три символа (`rw-`) - это права владельца, которым в данном случае является queen. Следовательно, queen может читать файл (`r`), изменять его содержимое (`w`), но не может запускать его (`-`).
- следующие три символа (`r--`) относятся к любому пользователю, кроме queen, который является членом группы `users`. Он будет в состоянии прочитать файл (`r`), но не сможет ни записать, ни выполнить его (`--`).
- последние три символа (`---`) относятся к любому пользователю, кроме queen и всех кто входит в группу `users`. Эти пользователи вообще не имеют никаких прав на этот файл.

Для каталога `a_directory` права выглядят так `rwxr-xr--`, отсюда:

- peter, как владелец каталога, может получить список находящихся в нем файлов (`r`), добавить или удалить файлы из этого каталога (`w`) и может пройти через него (`x`);
- Каждый пользователь, кроме peter, который входит в группу `users`, будет в состоянии получить список файлов в этом каталоге (`r`), но не сможет удалить или добавить файлы (`-`), а также сможет проходить его (`x`).

- Любой другой пользователь сможет только получить список содержимого этого каталога (`r`). Но поскольку у него нет прав `wx`, он не сможет записать файлы или войти в каталог.

Есть **одно** исключение из этих правил - `root`. `root` может изменять атрибуты (права доступа, владельца и группу) всех файлов, даже если он не является владельцем, и поэтому сможет сделать себя владельцем файла! `root` может читать файлы, для которых у него нет прав на чтение, проходить через каталоги, к которым у него, будь он обычным пользователем, не было бы доступа и т.д. И если `root`'у не хватает прав, ему нужно просто добавить их. `root` имеет полный контроль над системой, что влечет за собой определенный уровень доверия к человеку, знающего его пароль.

И в заключение, не стоит беспокоиться из-за различий между именами файлов в мирах UNIX® и Windows®. Первый - UNIX® - предоставляет значительно большую гибкость и имеет меньше ограничений.

- Имя файла может содержать любые символы, включая непечатаемые, за исключением ASCII-символа 0, который означает конец строки, и /, который является разделителем каталога. Кроме того, вследствие чувствительности к регистру в UNIX® файлы `readme` и `Readme` будут разными, потому что под буквами `r` и `R` в системах на базе UNIX® подразумеваются два **разных** символа.
- Как вы могли заметить, имя файла не обязательно должно иметь расширение, если только вам не захочется так называть свои файлы. В GNU/Linux расширения файлов не определяют их содержимого, а также на большинстве операционных систем. Тем не менее, так называемые "расширения файлов" довольно удобны. В UNIX® точка (`.`) - это просто один из символов, но он также имеет одно специальное назначение. В UNIX® файлы с именами, начинающимися с точки, являются "скрытыми"[1]; это также касается и каталогов, чьи имена начинаются с `.`

Однако нет стоит беспокоиться из-за того, что многие графические приложения (файловые менеджеры, офисные приложения и т.п.) все-таки используют файловые расширения для распознавания своих файлов. Это хорошая идея - использовать расширения в именах файлов для тех приложений, которые их поддерживают.

1.3. Процессы

Под ***процессом*** понимается копия выполняемой программы и ее ***окружение***. Здесь мы только упомянем наиболее важные различия между GNU/Linux и Windows® (для получения более подробной информации обратитесь, пожалуйста, к Гл. 6).

Наиболее важное различие напрямую связано с понятием **пользователя**: каждый процесс выполняется с правами пользователя, который его запустил. Внутри себя система идентифицирует процессы по уникальному номеру, называемому *идентификатором процесса* или PID (proccess ID) . Из этого PID система знает, кто (т.е. какой пользователь) запустил процесс, некоторые части другой информации, и ей нужно только проверить достоверность процесса. Давайте возьмем наш пример с `a_file`. Пользователь peter будет иметь возможность открыть этот файл в ***режиме только для чтения***, но не в ***режиме чтения-записи***, т.к. его права доступа к файлу запрещают это. И опять же, исключением из этого правила является `root`.

Благодаря этому, GNU/Linux практически неуязвима для вирусов. Для своей работы вирусы должны заражать исполняемые файлы. Как у пользователя, у вас нет доступа на запись в уязвимые системные файлы, таким образом, существенно снижается риск. Вообще говоря, вирусы очень редки в мире UNIX®. Существует всего лишь несколько малоизвестных вирусов для Linux, и они безобидны, если выполняются под обычным пользователем. Только один пользователь может повредить систему запуском этих вирусов: `root`.

Довольно интересно, что антивирусное программное обеспечение для GNU/Linux таки существует, но в основном для файлов DOS/Windows®! Зачем же нужны антивирусные программы, работающие в GNU/Linux, но ориентированные на DOS/Windows®? Все чаще и чаще вы будете встречать системы

1. По умолчанию скрытые файлы не будут отображаться в файловом менеджере до тех пор, пока вы не скажете ему сделать это. В терминале вы должны ввести команду `ls -a`, чтобы увидеть все файлы, включая скрытые. Зачастую в них содержится конфигурационная информация. В качестве примера взгляните на файлы `.mozilla` или `.openoffice` из своего каталога `home/`.

GNU/Linux, работающие в как файл-серверы для машин Windows® при помощи пакета программ Samba (смотрите главу Общий доступ к файлам и принтерам в книге *Руководство по администрированию сервера*).

Linux упрощает контроль процессов. Один из методов - это "сигналы", которые позволяют вам приостановить или убить процесс путем отправки ему соответствующего сигнала. Однако отправлять сигналы вы можете только своими собственным процессам. За исключением `root`'а, Linux и системы на базе UNIX® не позволят вам отправить сигналы процессам, запущенным другими пользователями. В главе Гл. 6 вы узнаете, как получать PID процесса и отправлять ему сигналы.

1.4. Краткое введение в командную строку

Командная строка - это самый прямой способ для отправки команд своей машине. Если вы будете использовать командную строку GNU/Linux, то вы вскоре обнаружите, что она значительно мощнее и обладает более широкими возможностями, чем любой другой интерпретатор команд. Эта мощь доступна благодаря тому, что вы имеете доступ не только ко всем приложениям X, но также и к тысячам утилит в консольном режиме (в противоположность графическому режиму), которые не имеют графических аналогов, с их многочисленными опциями и возможными комбинациями, достичь которых в виде кнопок или меню было бы труднее.

Надо отметить, что большинству людей для того, чтобы начать действовать, требуется некоторая помощь. Если вы еще не работаете в консольном режиме, и используете графический интерфейс, то первым делом вам надо запустить эмулятор терминала. Зайдите в главное в меню GNOME, KDE или любого другого оконного менеджера, который вы используете, и вы найдете несколько эмуляторов в меню Система+Терминалы. Выберите эмулятор на свой вкус, например Konsole или XTerm. В зависимости от вашего пользовательского интерфейса на панели задач также может существовать значок, четко его определяющий (Рис. 1-2).

Рисунок 1-2. Значок терминала на панели KDE

Когда вы запускаете этот эмулятор терминала, на самом деле вы используете shell. Это название программы, с которой вы работаете. Вы обнаружите перед собой ***приглашение***:

```
[queen@localhost queen]$
```

Здесь подразумевается, что ваше имя пользователя - queen, а ваша машина называется `localhost` (это в случае, если ваша машина не является частью существующей сети). Пространство после приглашения предназначено для ввода ваших команд. Обратите внимание, что когда вы `root`, знак `$` в приглашении меняется на # (это верно только в конфигурации по умолчанию, так как в GNU/Linux вы можете настроить все эти элементы под себя). Для того, чтобы стать `root`'ом, наберите `su` после запуска shell.

```
[queen@localhost queen]$ su
# Введите пароль root'а; (на экране он не появится)
Password:
# exit (или Ctrl+D) вернет вас назад в вашу обычную учетную запись пользователя
[root@localhost queen]# exit
[queen@localhost queen]$
```

Когда вы ***запускаете*** shell в первый раз, вы обычно попадаете в свой домашний каталог `home/`. Чтобы вывести на экран имя каталога, в котором вы в данный момент находитесь, наберите команду `pwd` (которая означает *вывести рабочий каталог (Print Working Directory)*):

```
$ pwd
/home/queen
```

Далее мы рассмотрим несколько основных и весьма полезных команд.

1.4.1. cd: Смена каталога

Команда `cd` такая же, как и в DOS'е, но с дополнительными возможностями. Она выполняет как раз то, что заявлено в ее аббревиатуре - сменяет рабочий каталог. Вы можете использовать `.` и `..`, которые означают текущий и родительский каталоги соответственно. Запуск `cd` без параметров вернет вас назад в ваш домашний каталог. Запуск `cd -` вернет вас назад в последний посещенный вами каталог. И, наконец, вы можете указать домашний каталог пользователя peter, набрав `cd ~peter` (`~` сама по себе означает ваш собственный каталог `home/`). Обратите внимание, что как обычный пользователь, вы, как правило, не можете попасть в каталоги `home/` других пользователей (если только они не разрешили это, или если это не настройка системы по умолчанию), если вы не `root`, поэтому давайте станем `root`'ом и попрактикуемся:

```
$ su
Password:
# pwd
/root
# cd /usr/share/doc/HOWTO
# pwd
/usr/share/doc/HOWTO
# cd ../FAQ-Linux
# pwd
/usr/share/doc/FAQ-Linux
# cd ../../../lib
# pwd
/usr/lib
# cd ~peter
# pwd
/home/peter
# cd
# pwd
/root
```

Теперь вернемся назад в состояние обычного пользователя, набрав `exit` (или нажав **Ctrl-D**).

1.4.2. Некоторые переменные окружения и команда echo

Все процессы имеют свои ***переменные окружения***, а shell позволяет вам увидеть их непосредственно при помощи команды `echo`. Некоторые интересные переменные:

1. `HOME`: эта переменная окружения содержит строку, в которой отображен путь к вашему домашнему каталогу.
2. `PATH`: содержит список всех каталогов, в которых shell должен будет искать исполняемые файлы, когда вы набираете команду. Обратите внимание, что в отличие от DOS, shell по умолчанию **не** будет искать команды в текущем каталоге!
3. `USERNAME`: эта переменная содержит ваше имя логина.
4. `UID`: эта переменная содержит ваш ID пользователя.
5. `PS1`: определяет, как будет выглядеть ваше приглашение, и зачастую является комбинацией специальных последовательностей. Для получения дополнительной информации вы можете прочитать bash(1) (***страницу руководства***), набрав в терминале `man bash`.

Чтобы shell вывел значение переменной, вы должны указать перед ее именем знак `$`. Вот пример с командой `echo`:

```
$ echo Hello
Hello
$ echo $HOME
/home/queen
$ echo $USERNAME
queen
$ echo Hello $USERNAME
Hello queen
$ cd /usr
$ pwd
/usr
$ cd $HOME
$ pwd
/home/queen
```

Как видите, shell подставляет значение переменной перед выполнением команды. В противном случае наш пример `cd $HOME` не заработал бы. По сути командный процессор сначала заменил `$HOME` на ее значение (**/home/queen**), так что строка стала `cd /home/queen`, чего мы и добивались. То же самое происходит и с примером `echo $USERNAME`.

Если одна из ваших переменных окружения не существует, вы можете временно их создать, набрав `export` **ИМЯ_ПЕРЕМ_ОКР=значение**. После этого вы можете проверить, были ли они созданы:

```
$ export USERNAME=queen $ echo $USERNAME queen
```

1.4.3. cat: Вывод на экран содержимого одного или более файлов

Нечего добавить, эта команда делает только это: она выводит содержимое одного или более файлов на стандартный вывод, обычно на экран:

```
$ cat /etc/fstab
/dev/hda5 / ext2 defaults 1 1
/dev/hda6 /home ext2 defaults 1 2
/dev/hda7 swap swap defaults 0 0
/dev/hda8 /usr ext2 defaults 1 2
/dev/fd0 /mnt/floppy auto sync,user,noauto,nosuid,nodev 0 0
none /proc proc defaults 0 0
none /dev/pts devpts mode=0620 0 0
/dev/cdrom /mnt/cdrom auto user,noauto,nosuid,exec,nodev,ro 0 0
$ cd /etc
$ cat modules.conf shells
alias parport_lowlevel parport_pc
pre-install plip modprobe parport_pc ; echo 7 > /proc/parport/0/irq
#pre-install pcmcia_core /etc/rc.d/init.d/pcmcia start
#alias char-major-14 sound
alias sound esssolo1
keep
/bin/zsh
/bin/bash
/bin/sh
/bin/tcsh
/bin/csh
/bin/ash
/bin/bsh
/usr/bin/zsh
```

1.4.4. less: Пейджер

Название произошло от игры слов, связанной с первым пейджером из когда-либо использовавшихся в UNIX® с именем `more` (more - больше; less - меньше). ***Пейджер*** (page - страница) - это программа, которая позволяет пользователю просматривать большие файлы по частями страница за страницей (точнее, экран за экраном). Причина, по которой мы рассматриваем `less`, а не `more`, заключается в том, что `less` более интуитивна. Вам следует использовать `less` для просмотра больших файлов, которые не умещаются на одном экране. Например:

```
less /etc/termcap
```

Для перемещения по этому файлу используйте клавиши вверх и вниз. Для выхода нажмите **Q**. Возможности `less` значительно шире: нажмите **H** для вызова справки с различными доступными опциями.

1.4.5. ls: Вывод списка файлов

Команда `ls` (*LiSt*) эквивалентна команде `dir` в DOS, но сделать она может больше, гораздо больше. В действительности это связано с тем, что файлы также могут делать гораздо больше. Синтаксис команды `ls`:

```
ls [опции] [файл|каталог] [файл|каталог...]
```

Если в командной строке не указан файл или каталог, `ls` выведет список файлов в текущем каталоге. Опций довольно много, поэтому мы опишем только некоторые из них:

- `-a`: вывод списка всех файлов, включая ***скрытые файлы***. Напомним, что в UNIX® скрытые файлы - это те, чьи имена начинаются с `.`; опция `-A` выводит список “почти” всех файлов, т.е. всех файлов, которые были бы показаны опцией `-a` за исключением “.” и “..”
- `-R`: рекурсивный вывод списка, т.е. все файлы и подкаталоги в каталогах, указанных в командной строке.
- `-h`: вывод для каждого файла его размера в удобном для чтения формате. Это означает, что вы увидите размеры файлов с использованием суффиксов типа "K", "M" и "G", например, "234K" и "132M". Пожалуйста, обратите внимание, что размеры вычисляются по основанию 2, а не по основанию 10. Это означает, что 1 K на самом деле равен 1024 байтам, а не 1000 байт.
- `-l`: вывод дополнительной информации о файлах: их права доступа, владельцы и группы, размеры файлов и время последнего доступа.
- `-i`: вывод перед каждым файлом номера inode (уникальный номер файла в файловой системе, см. Гл. 9).
- `-d`: обработка каталогов, указанных в командной строке так, как если бы они были обычными файлами, вместо вывода списка их файлов.

Вот несколько примеров:

- `ls -R`: рекурсивно выводит список содержимого текущего каталога.
- `ls -is images/ ..`: выводит список с номером inode и размером в килобайтах для каждого из файлов в каталоге `images/`, а также в родительском каталоге по отношению к текущему.
- `ls -l images/*.png`: выводит список всех файлов в каталоге `images/`, чьи имена заканчиваются на `.png`, включая файл `.png`, если такой существует.

1.4.6. Полезные комбинации клавиш

Существует большое количество сокращенных клавиатурных команд. Их основное преимущество состоит в том, что они экономят для вас время при наборе на клавиатуре. В этом разделе подразумевается, что вы пользуетесь стандартным shell’ом, по умолчанию поставляемым вместе с Mandrakelinux - bash, но эти клавиши также могут работать и в других командных процессорах.

Во-первых, клавиши со стрелками. bash хранит историю предыдущих команд, которые вы можете увидеть при помощи клавиш вверх и вниз. Вы можете прокрутить назад столько строк, сколько определено в переменной окружения `HISTSIZE`. Кроме того, от сеанса к сеансу история остается неизменной, поэтому вы сохраните все команды, введенные вами в предыдущем сеансе работы.

Клавиши со стрелками влево и вправо перемещают курсор влево и вправо в текущей строке, позволяя вам редактировать свои команды. Но при редактировании вы можете не только просто перемещаться на один символ за раз: **Ctrl-A** и **Ctrl-E**, например, переместят вас в начало и конец текущей строки. Клавиши **Backspace** и **Del** работают так, как и должны. **Backspace** и **Ctrl-H** эквивалентны. **Del** и **Ctrl-D** также могут быть использованы поочередно. **Ctrl-K** удалит все от текущей позиции курсора до конца строки, а **Ctrl-W** удалит слово перед курсором (так же как и **Alt-Backspace**).

Нажатие **Ctrl-D** в пустой строке позволит вам закрыть текущий сеанс, что намного проще, чем необходимость вводить `exit`. **Ctrl-C** прервет выполняющуюся в данный момент команду, если только вы не в процессе редактирования своей командной строки, в этом случае редактирование будет отменено и вы вернетесь назад к исходному приглашению. **Ctrl-L** очищает экран. **Ctrl-Z** временно останавливает

выполнение задачи, т.е. она приостанавливается. Эта комбинация очень полезна, если вы забыли ввести символ "&" после набора команды. Например:

```
$ xpdf MyDocument.pdf
```

С этого момента вы больше не сможете воспользоваться своим командным процессором, т.к. приоритетной задачей стал процесс xpdf. Чтобы сделать процесс фоновым и восстановить работу своей консоли, просто нажмите **Ctrl-Z** и введите `bg`.

И в заключение, имеются комбинации **Ctrl-S** и **Ctrl-Q**, которые используются для приостановки и возобновления вывода на экран. Они нечасто используются, но вы можете по ошибке ввести **Ctrl-S** (в конце концов, **S** и **D** на клавиатуре находятся близко друг от друга). Поэтому, если вы попадете в ситуацию, когда вы что-то набираете на клавиатуре, но в Терминал'е никакие символы не появляются, попробуйте нажать **Ctrl-Q**. Обратите внимание, что все символы, введенные вами между нежелательной комбинацией **Ctrl-S** и **Ctrl-Q** будут единовременно выведены на экран.

Глава 2. Диски и разделы

В этой главе представлена информация для тех, кто просто хочет больше узнать о технических деталях относительно их системы. В ней будет дано полное описание схемы разметки дисков PC. Следовательно, наиболее полезной она будет в том случае, если вы планируете вручную настроить разделы своего жесткого диска. Т.к. инсталлятор может автоматически разметить ваш диск, понимание всего этого не является настолько важным, если вы планируете выполнить стандартную установку.

2.1. Структура жесткого диска

Диск физически разбит на секторы. Последовательность секторов может сформировать раздел. Грубо говоря, вы можете создать столько разделов, сколько вам нужно, но не более 67 (3 основных раздела и 1 дополнительный, содержащий до 64 логических разделов): каждый раздел рассматривается как отдельный жесткий диск.

2.1.1. Секторы

Упрощенно жесткий диск можно рассматривать как простую последовательность секторов, представляющих собой наименьшие блоки данных жесткого диска. Обычно размер сектора составляет 512 байт. Секторы на жестком диске из “n” секторов нумеруются от “0” до “n-1”.

2.1.2. Разделы

Использование нескольких разделов позволяет вам создать много виртуальных жестких дисков на реальном физическом диске. Это дает много преимуществ:

- Разные операционные системы используют разные структуры диска (называемые ***файловыми системами***): как в случае с Windows® и GNU/Linux. Наличие нескольких разделов на жестком диске позволяет вам установить разные операционные системы на один физический жесткий диск.
- Из соображений производительности операционная система может использовать различные диски с разными файловыми системами на них, потому что они могут использоваться для совершенно разных задач. Одним из примеров является GNU/Linux, для которого требуется второй раздел, называемый swap’ом. В В дальнейшем он используется менеджером виртуальной памяти в качестве виртуальной памяти.
- Даже если на всех ваших разделах используются одна и та же файловая система, весьма полезным может оказаться разнести отдельные части вашей ОС на разные разделы. Простейшим примером такой конфигурации будет разнесение ваших файлов на два раздела: один для вашей личной информации, а другой для ваших программ. Это позволит вам обновить свою ОС, полностью удалив раздел с программами, сохранив при этом нетронутым раздел с данными.
- Вследствие того, что физические ошибки на жестком диске обычно появляются на соседних секторах, а не разбросаны по всему диску, размещение ваших файлов на различных разделах может ограничить потерю информации в случае физического повреждения жесткого диска.

Обычно тип раздела определяет содержащуюся на нем файловую систему. Каждая из операционных систем может распознать некоторые типы разделов, но не может распознать других. Для получения дополнительной информации обратитесь, пожалуйста, к главам Гл. 8 и Гл. 9.

2.1.3. Определение структуры вашего диска

2.1.3.1. Простейший способ

Этот сценарий будет подразумевать наличие только двух разделов: один для swap-пространства, другой - для файлов[1].

На практике было установлено, что размер раздела для свопинга должен быть равен двум объемам вашей оперативной памяти (RAM, Random Access Memory). Т.е., если у вас 128 МБ ОЗУ, то размер свопа должен составлять 256 МБ. Однако при наличии большого объема памяти (>512 МБ) это правило не является обязательным, и допускается меньший размер. Пожалуйста, примите во внимание, что размер раздела для свопинга может быть ограничен в зависимости от используемой платформы. Например, он ограничен до 2Гб для x86, PowerPC и MC680x0; до 512МБ для MIPS; до 128Гб для Alpha и до 3ТБ для Ultrasparc. Запомните также, что, чем больше раздел для свопинга, тем больше требуется ресурсов ОС (в частности памяти RAM) для его обслуживания.

2.1.3.2. Другая общая схема

Отделение данных от программ. Для большей эффективности обычно создают третий раздел, называемый "корнем" ("root") и обозначаемый как /. В нем будут содержаться программы, необходимые для запуска вашей системы и выполнения базового обслуживания.

Следовательно, мы можем определить четыре раздела:

Своп

Раздел для свопинга (swap), чей размер, равен примерно двум объемам физической оперативной памяти.

Корень: /

Самый важный раздел. И не только потому, что он содержит критически важную информацию и программы для системы, он также является и точкой монтирования для других разделов (см. главу Гл. 8).

Требования к размеру корневого раздела не слишком велики - 400МБ будет вполне достаточно. Однако, если вы планируете устанавливать коммерческие приложения, которые зачастую размещаются в каталоге /opt, вам понадобится соответственно увеличить и размер корневого раздела. В качестве альтернативы вы можете создать отдельный раздел для каталога /opt.

Статические данные: /usr

Большинство пакетов устанавливают основную часть своих исполняемых файлов и файлов данных в каталог /usr. Преимущество создания отдельного раздела заключается в том, что это позволяет вам легко открыть к нему общий доступ для других машин в сети.

Рекомендуемый размер зависит от пакетов, которые вы хотите установить, и может варьироваться от 100МБ при облегченной установке до нескольких Гб при полной установке. Обычно достаточно выделить 2-3 Гб (в зависимости от размера вашего диска).

1. файловая система, используемая в Mandrakelinux по умолчанию, называется ext3

Домашние каталоги: `/home`

> Этот каталог содержит личные каталоги всех пользователей вашей системы. Размер раздела зависит от количества пользователей и их потребностей.

Как вариант, можно **не** создавать отдельного раздела для файлов `/usr`: `/usr` может быть просто каталогом корневого раздела (`/`). Однако при этом вам соответствующим образом потребуется увеличить размер и своего корневого раздела.

И в заключение, вы также можете создать только разделы swap и `root` (`/`) в случае, если вы не уверены в том, какую работу вы будете выполнять на своем компьютере. В этом случае в вашем корневом разделе будут расположены каталоги `/home`, `/usr` и `/var`.

2.1.3.3. Экзотические конфигурации

Когда ваша машина настраивается для использования в определенных целях, таких как веб-сервер или файервол, требования радикально отличаются от тех, что подходят для стандартной настольной системы. Например, для сервера FTP наверняка потребуется отдельный большой раздел для каталога `/var/ftp`, а размер каталога `/usr` может быть и меньше. В таких случаях вам следует хорошо продумать свои требования перед тем, как начинать процесс установки.

Если вам нужно изменить размеры своих разделов или использовать другую схему разметки диска, обратите внимание на то, что размеры большинства разделов можно изменить без переустановки системы и потери каких-либо данных. Пожалуйста, обратитесь к разделу *Управление дисковыми разделами* книги *Стартовое руководство* .

При наличии некоторого опыта вы даже сможете перенести переполненный раздел на новый жесткий диск.

2.2. Соглашения при именовании дисков и разделов

В GNU/Linux используется логический метод при именовании разделов. Во-первых, при нумерации разделов игнорируются типы файловых систем любого из разделов, которые у вас могут быть. Во-вторых, разделы именуются согласно диску, на котром они находятся. Вот как именуются диски:

- Первичный ведущий (primary master) и первичный ведомый (primary slave) устройства IDE (будь то жесткие диски, приводы CD-ROM или что-то еще) называются соответственно `/dev/hda` и `/dev/hdb`.
- На вторичном интерфейсе ведущее устройство называется `/dev/hdc`, а ведомое - `/dev/hdd`.
- Если в вашем компьютере имеются другие интерфейсы IDE (интерфейс IDE, например, присутствует на некоторых картах Soundblaster), диски будут называться `/dev/hde`, `/dev/hdf` и т.д. Вы также можете иметь дополнительные IDE-интерфейсы, если у вас есть RAID-контроллеры.
- SCSI-диски называются `/dev/sda`, `/dev/sdb` и т.д., в порядке их размещения в цепи SCSI (в зависимости от увеличения ID). Приводы SCSI CD-ROM называются `/dev/scd0`, `/dev/scd1` всегда в порядке их размещения в цепи SCSI.

Если у вас присутствуют SATA IDE-диски, применяется схема именования SCSI.

Разделы именуются по диску, на котором они найдены, следующим образом (в нашем примере мы использовали разделы на первичном ведущем диске IDE, но то же самое касается и дисков других типов):

- Основные (или расширенные) разделы именуются от `/dev/hda1` до `/dev/hda4`.

- Логические разделы, если они есть, именуются `/dev/hda5`, `/dev/hda6` и т.д. в порядке их появления в таблице логических разделов.

Таким образом GNU/Linux будет именовать разделы следущим образом:

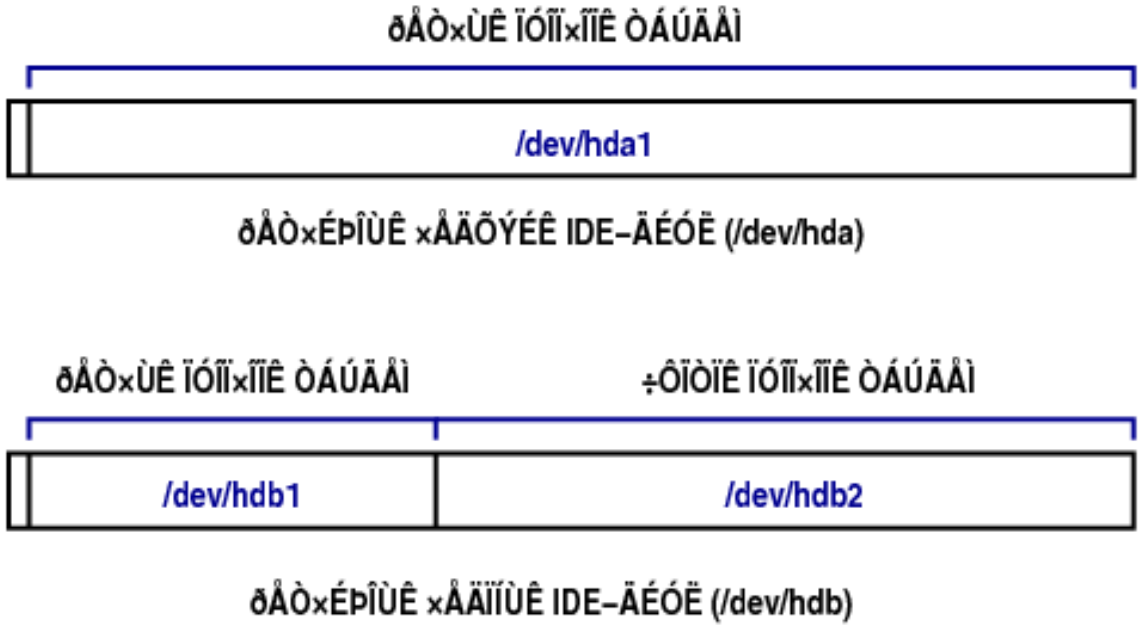


Рисунок 2-1. Первый пример именования разделов в GNU/Linux

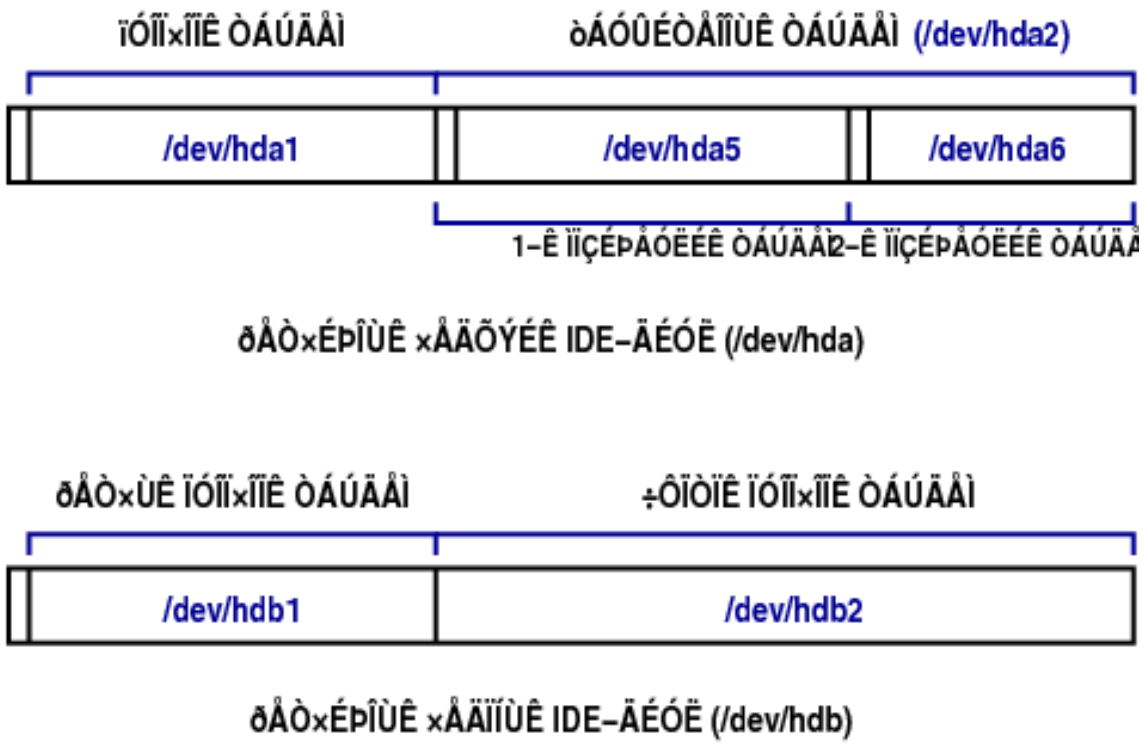


Рисунок 2-2. Второй пример именования разделов в GNU/Linux

Имея в своем распоряжении эти знания, вы сможете именовать различные разделы и жесткие диски при работе с ними. Вы также увидите, что GNU/Linux именует разделы, даже если изначально не знает, как управлять ими (при этом игнорируется тот факт, что они не являются родными разделами GNU/Linux).

В Mandrakelinux теперь используется пакет udev (для получения дополнительной информации обратитесь к udev FAQ (`http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ`)). Он обеспечивает полную совместимость с описанной выше схемой и со стандартами Linux Standards Base Project (`http://www.linuxbase.org/`). Каждое устройство динамически добавляется в систему по мере его необходимости или появления.

Глава 3. Введение в командную строку

В главе Гл. 1 вы увидели как запускать shell. В данной главе мы покажем вам как работать с ней.

Главная ценность shell это множество существующих утилит: их тысячи и каждая выполняет свою отдельную задачу. Мы рассмотрим только их небольшое число. Одно из величайших преимуществ UNIX® это умение комбинировать эти утилиты, что мы увидим позже.

3.1. Утилиты обработки файлов

В данном контексте под обработкой файлов подразумевается копирование, перемещение и удаление файлов. Позже мы рассмотрим способы изменения атрибутов файлов (владелец, права доступа).

3.1.1. mkdir, touch: Создание пустых каталогов и файлов

Команда `mkdir` (*MaKe DIRectory* - создать каталог) используется для создания каталогов. Ее синтаксис довольно прост:

```
mkdir [опции] <каталог> [каталог ...]
```

Только на одну опцию имеет смысл обратить внимание: опция `-p`. Она выполняет два действия:

1. создает родительские каталоги, если они не существовали до этого. Без этой опции `mkdir` просто вызовет сбой, жалуясь на отсутствие заявленных каталогов;
2. молча завершит работу если каталог, который вы хотели создать, уже существует. Для сравнения, если вы не указали опцию `-p`, `mkdir` возвратит сообщение об ошибке, жалуясь на то, что каталог уже существует.

Вот несколько примеров:

- `mkdir foo`: создает каталог `foo` в текущем каталоге;
- `mkdir -p images/misc docs`: создает каталог `misc` в каталоге `images`. Сначала создается последний каталог, если такой не существует (`-p`); также создается каталог с именем `docs` в текущем каталоге.

Изначально команда `touch` предназначалась не для создания файлов, а для обновления времени последнего доступа к файлу и времени его изменения[1]. Однако `touch` создаст перечисленные пустые файлы, если они не существуют. Ее синтаксис:

```
touch [опции] файл [файл...]
```

Таким образом, выполнение команды:

```
touch file1 images/file2
```

создаст в текущем каталоге пустой файл с именем `file1` и пустой файл `file2` в каталоге `images`, если ранее эти файлы не существовали.

3.1.2. rm: Удаление файлов или каталогов

Команда `rm` (*ReMove* - удалить) является аналогом команд DOS `del` и `deltree`, и содержит дополнительные опции. Ее синтаксис выглядит следующим образом:

```
rm [опции] <файл|каталог> [файл|каталог...]
```

Опции включают:

1. В UNIX® для всех файлов существуют три разных временных метки: дата последнего доступа к файлу (`atime`), т.е. дата, когда в последний раз файл открывался для чтения или записи; дата изменения атрибутов inode (`ctime`); и, наконец, дата последнего изменения **содержимого** файла (`mtime`).

- `-r` или `-R`: рекурсивное удаление. Эта опция является **обязательной** для удаления каталога, пустого или нет. Однако для удаления пустых каталогов вы также можете воспользоваться командой `rmdir`.
- `-i`: запрос подтверждения перед каждым удалением. Обратите внимание, что по умолчанию в Mandrakelinux, `rm` по соображениям безопасности - это ***алиас*** команды `rm -i` (похожие алиасы существуют для `cp` и `mv`). Однако ваше мнение относительно полезности этих алиасов может измениться. Если вы хотите удалить их, вы можете создать пустой файл `~/.alias`, который предотвратит использование общесистемных алиасов. Или же вы можете отредактировать свой файл `~/.bashrc` для отключения некоторых общесистемных алиасов путем добавления этой строки: `unalias rm cp mv`
- `-f`, как противоположность `-i`, принудительно удалит файлы или каталоги, даже если у пользователя нет доступа для записи в файлы[2].

Несколько примеров:

- `rm -i images/*.jpg file1`: удаляет все файлы с именами, заканчивающимися на `.jpg`, в каталоге `images` и удаляет `file1` в текущем каталоге, запрашивая подтверждение на удаление каждого файла. Отвечайте **y** для подтверждения удаления и **n** для отмены.
- `rm -Rf images/misc/ file*`: удаляет без запроса подтверждения весь каталог `misc/` в каталоге `images/` вместе со всеми файлами в текущем каталоге, чьи имена начинаются с `file`.

При использовании `rm` файлы удаляются **безвозвратно**. Способа для их восстановления не существует! (Ну, вообще-то для этого всё-таки есть несколько способов, но это нетривиальная задача и обычно используется предупреждение удаления файлов.) Не раздумывайте при использовании опции `-i`, чтобы убедиться в том, что вы не удалите по ошибке что-нибудь важное.

3.1.3. mv: Перемещение или удаление файлов

Синтаксис команды `mv` (*MoVe* - переместить) следующий:

```
mv [опции] <файл|каталог> [файл|каталог ...] <назначение>
```

Обратите внимание, что когда вы перемещаете несколько файлов, назначением должен быть каталог. Чтобы переименовать файл, просто переместите его в файл с новым именем.

Некоторые опции:

- `-f`: форсирование операции — если перезаписывается существующий файл, предупреждения не выводятся.
- `-i`: противоположное действие. У пользователя спрашивается подтверждение перед перезаписью существующего файла.
- `-v`: ***подробный*** режим, сообщает обо всех изменениях и действиях.

Несколько примеров:

- `mv -i /tmp/pics/*.png .`: перемещает все файлы из каталога `/tmp/pics/`, чьи имена заканчиваются на `.png`, в текущий каталог (`.`), но запрашивает подтверждение перед перезаписью в нем любых файлов.
- `mv foo bar`: переименовывает файл `foo` в `bar`. Если каталог `bar` уже существовал, результатом выполнения этой команды будет перемещение файла `foo` или всего каталога (самого каталога плюс всех файлов и каталогов в нем, рекурсивно) в каталог `bar`.
- `mv -vf file* images/ trash/`: перемещает без запроса подтверждения все файлы из текущего каталога с именами, начинающимися с `file`, вместе со всем каталогом `images/` в каталог `trash/`, и показывает порядок выполнения каждой операции.

2. Для пользователя достаточно иметь доступ для записи в **каталог**, чтобы он мог удалять файлы в нем, даже если он не является владельцем файлов.

3.1.4. cp: Копирование файлов и каталогов

`cp` (*CoPy* - копировать) является аналогом команд DOS `copy` и `xcopy` и содержит дополнительные опции. Ее синтаксис выглядит следующим образом:

```
cp [опции] <файл|каталог> [файл|каталог ...] <назначение>
```

`cp` имеет много опций. Вот самые общие из них:

- `-R`: рекурсивное копирование; **обязательна** для копирования каталога, даже если он пуст.
- `-i`: запрос подтверждения перед перезаписью любых файлов, которые могут быть перезаписаны.
- `-f`: противоположность `-i`, заменяет любые существующие файлы без запроса подтверждения.
- `-v`: подробный режим, сообщает обо всех действиях, выполняемых `cp`.

Несколько примеров:

- `cp -i /timages/* images/`: копирует все файлы из каталога **`/timages/`** в каталог **`images/`**, находящийся в текущем каталоге. Запрашивается подтверждение, если должен быть перезаписан файл.
- `cp -vR docs/ /shared/mp3s/* mystuff/`: копирует весь каталог **`docs`**, плюс все файлы из каталога **`/shared/mp3s`** в каталог **`mystuff`**.
- `cp foo bar`: делает копию файла **`foo`** в файл с именем **`bar`** в текущем каталоге.

3.2. Управление атрибутами файлов

Ряд показанных здесь команд используется для изменения владельца или группы файла или его прав. Мы рассматривали различные типы прав доступа в Гл. 1.

3.2.1. chown, chgrp: Изменение владельца или группы для одного или нескольких файлов

Синтаксис команды `chown` (*CHange OWNer* - изменить владельца) выглядит следующим образом:

```
chown [опции] <пользователь[:группа]> <файл|каталог> [файл|каталог...]
```

Опции включают:

- `-R`: рекурсивно. Для изменения владельца для всех файлов и подкаталогов в указанном каталоге.
- `-v`: подробный режим. Показывает все действия, выполняемые `chown`; сообщает, у каких файлов были изменены владельцы в результате выполнения команды, а какие файлы остались без изменений.
- `-c`: подобна опции `-v`, но сообщает только о тех файлах, что были изменены.

Несколько примеров:

- `chown nobody /shared/book.tex`: изменяет владельца файла **`/shared/book.tex`** на `nobody`.
- `chown -Rc queen:music *.mid concerts/`: изменяет владельца всех файлов в текущем каталоге, чьи имена заканчиваются на `.mid`, и всех файлов и подкаталогов в каталоге **`concerts/`** на пользователя `queen` и группу `music`, сообщая только о тех файлах, которые были затронуты командой.

Команда `chgrp` (*CHange GRouP* - изменить группу) позволяет вам изменять принадлежность к группе файла (или файлов); ее синтаксис очень похож на синтаксис команды `chown`:

```
chgrp [опции] <группа> <файл|каталог> [файл|каталог...]
```

Опции для этой команды такие же, как и у `chown`, и она используется очень похожим способом. Так, команда:

```
chgrp disk /dev/hd*
```

изменяет принадлежность всех файлов в каталоге **/dev/** с именами, начинающимися с **hd**, на группу **disk**.

3.2.2. chmod: Изменение прав файлов и каталогов

Команда chmod (*CHange MODe* - изменить режим) имеет весьма специфический синтаксис. В общем случае это выглядит так:

```
chmod [опции] <изменение режима> <файл|каталог> [файл|каталог...]
```

но различие состоит в формах, которыми можно изменять режим. Он может быть определен двумя способами:

1. через восьмеричные числа. Права владельца в этом случае соответствуют числам в форме <x>00, где <x> соответствует присвоенным правам: 4 на чтение, 2 на запись и 1 на выполнение. Так же права группы берутся по форме <x>0 и права для "других" в форме <x>. Затем, все что вам нужно сделать, это сложить вместе присвоенные права, чтобы получить правильный режим. Так, права rwxr-xr-- соответствуют 400+200+100 (права владельца, rwx) +40+10 (права группы, r-x) +4 (права для других, r--) = 754; таким образом права выражены в абсолютных значениях. Это значит, что предыдущие права безоговорочно изменены;
2. через выражения. Здесь права выражены последовательностью выражений, разделенных запятыми. Следовательно, выражение будет иметь вид: [category]<+|-|=><**права**>.

   Категорий может быть одна или несколько:

   - u (*User* - пользователь), права владельца;
   - g (*Group* - группа), права владельца группы;
   - o (*Others* - остальные), права для "остальных".

   Если категория не определена, изменения будут приняты для всех категорий. + устанавливает права, – забирает права и = устанавливает права на то, что указано в командной строке. Вообщем, права бывают следующие:

   - r (*Read* - чтение);
   - w (*Write* - запись);
   - x (*eXecute* - выполнение).

Главные опции полностью идентичны опциям команд chown и chgrp:

- -R: изменяет права рекурсивно.
- -v: подробный режим. Показывает выполняемые действия для каждого файла.
- -c: подобна опции -v, но сообщает только о тех файлах, которые были подвержены действию команды.

Примеры:

- chmod -R o-w /shared/docs: рекурсивно снимает права на запись для **остальных** всем файлам и подкаталогам в каталоге **/shared/docs/**.
- chmod -R og-w,o-x private/: рекурсивно снимает права на запись для **группы** и **остальных** во всем каталоге **private/** и снимает права на выполнение для **остальных**.
- chmod -c 644 misc/file*: изменяет права для всех файлов в каталоге **misc/**, чьи имена начинаются с **file** на rw-r--r-- (то есть чтение для всех, а запись только для владельца), и сообщает только о тех файлах, которые были подвержены действию команды.

3.3. Шаблоны подстановки в командном процессоре

Вы, вероятно, уже использовали символы ***подстановки***, не зная, что это такое. Если вы указываете файл в Windows® или выполняете поиск файла, вы используете * для установки соответствия случайной строке. Например, `*.txt` соответствует всем файлам с именами, заканчивающимися на `.txt`. Мы также часто использовали это в последнем разделе. Но кроме * существует еще много других подстановок.

Когда вы вводите команду типа `ls *.txt` и нажимаете на **Enter**, задача по нахождению файлов, соответствующих шаблону `*.txt`, выполняется не командой `ls`, а самим shell'ом. Требуется некоторое пояснение того, как командная строка интерпретируется shell'ом. Когда вы вводите:

```
$ ls *.txt readme.txt  recipes.txt
```

командная строка разбивается сначала на слова (`ls` и `*.txt` в этом примере). Когда командный процессор видит в слове *, он будет интерпретировать все слово как шаблон универсализации и будет заменять его именами всех соответствующих файлов. Поэтому команда, перед тем как командный процессор ее исполнит, принимает вид `ls readme.txt recipe.txt`, что и дает ожидаемый результат. Другие символы также заставляют командный процессор реагировать подобным образом:

- `?`: соответствует одному и только одному символу, независимо от того, чем является этот символ;
- `[...]`: соответствует любому символу, найденному в скобках. Символы могут быть заданы или в виде диапазона символов (т.е. `1-9`), или в виде ***дискретных значений***, или даже в двух видах одновременно. Пример: `[a-zBE5-7]` будет соответствовать всем символам между `a` и `z`, а также символам `B`, `E`, `5`, `6` или `7`;
- `[!...]`: соответствует любому символу **не** найденному в скобках. `[!a-z]`, например, будет соответствовать любому символу, который не является буквой в нижнем регистре[3];
- `{c1,c2}`: соответствует `c1` или `c2`, где `c1` и `c2` также являются шаблонами подстановки. Это означает, что вы, например, можете написать `{[0-9]*,[acr]}`.

Далее представлено несколько шаблонов и их значения:

- `/etc/*conf`: все файлы в каталоге `/etc` с именами, заканчивающимися на `conf`. Этому могут соответствовать файлы `/etc/inetd.conf`, `/etc/conf.linuxconf`, **а также** `/etc/conf` , если такой файл существует. Помните, что * также может соответствовать пустой строке.
- `image/{cars,space[0-9]}/*.jpg`: все файлы с именами, заканчивающимися на `.jpg`, в каталогах `image/cars`, `image/space0`, (...), `image/space9`, если такие каталоги существуют.
- `/usr/share/doc/*/README`: все файлы с именами `README` во всех подкаталогах, следующих непосредственно после `/usr/share/doc`. Одним из совпадений, например, будет `/usr/share/doc/mandrake/README`, но не `/usr/share/doc/myprog/doc/README`.
- `*[!a-z]`: все файлы в текущем каталоге с именами, которые **не** оканчиваются буквой в нижнем регистре.

3.4. Перенаправления и каналы

3.4.1. Немного подробнее о процессах

Чтобы понять принцип действия перенаправлений и каналов, мы должны объяснить понятие процесса, которое пока еще не было представлено. Большинство процессов UNIX® (сюда также включаются графические приложения, но исключается большинство демонов) использует как минимум три файловых дескриптора: стандартный ввод, стандартный вывод и стандартный поток ошибок. Их соответствующие номера - 0, 1 и 2. В общем случае эти три дескриптора ассоциируются с терминалом, из которого был запущен процесс, с клавиатурой в качестве устройства ввода. Цель перенаправлений и каналов - переадресация этих дескрипторов. Примеры в этом разделе помогут вам лучше понять этот принцип.

3. Будьте осторожны! Хотя это справедливо для большинства языков, это может быть не так в вашей собственной настройке языка (**локали**). Это зависит от **порядка сортировки**. В некоторых языковых настройках `[a-z]` будет соответствовать a, A, b, B, (...), z. Не говоря уже о том, что некоторые языки имеют подчеркнутые символы..

3.4.2. Перенаправления

Допустим вам нужно получить список файлов, имена которых заканчиваются на `.png` [4] в каталоге `images`. Этот список очень длинный, поэтому у вас может возникнуть желание сохранить его в файл для того, чтобы просмотреть его позже. Вы можете ввести следующую команду:

```
$ ls images/*.png 1>file_list
```

Это означает, что стандартный вывод этой команды (`1`) перенаправляется (`>`) в файл с именем `file_list`. Оператор `>` - это оператор перенаправления вывода. Если файл для перенаправления не существует, он будет создан, но если он существует, его предыдущее содержимое будет перезаписано. Однако дескриптором по умолчанию, который перенаправляется этим оператором, является стандартный вывод, и поэтому нет необходимости явным образом указывать его в командной строке. Следовательно, вы можете записать команду более упрощенно:

```
$ ls images/*.png >file_list
```

и результат при этом будет точно таким же. Затем вы можете просмотреть файл при помощи программы просмотра текстовых файлов, например, `less`.

Теперь представьте, что вам нужно узнать, сколько существует этих файлов. Вместо того, чтобы считать их вручную, вы можете воспользоваться утилитой под названием `wc` (*Word Count* - подсчет слов) с опцией `-l`, которая выводит на стандартный вывод число строк в файле. Вот один из вариантов решения:

```
wc -l 0<file_list
```

который выдаст вам искомый результат. Оператор `<` - это оператор перенаправления ввода, а дескриптором по умолчанию, который перенаправляется этим оператором, является стандартный ввод, т.е. `0`, и вы можете ввести просто строку:

```
wc -l <file_list
```

Теперь представьте, что вам нужно удалить все "расширения" файлов и поместить результат в другой файл. Одной из утилит для выполнения этого является `sed` (*Stream EDitor* - потоковый редактор). Вы просто перенаправляете стандартный ввод `sed` в файл `file_list`, а его вывод перенаправляете в результирующий файл, т.е. `the_list`:

```
sed -e 's/\.png$//g' <file_list >the_list
```

и ваш список создан, готовый для просмотра в удобное для вас время любой из программ просмотра.

Также полезным может оказаться перенаправлять стандартный поток ошибок. Например, вам нужно узнать, к каким каталогам в `/shared` у вас нет доступа: одним из решений будет рекурсивное получение списка файлов в этом каталоге и перенаправление ошибок в файл, не показывая при этом стандартного вывода:

```
ls -R /shared >/dev/null 2>errors
```

это означает, что стандартный вывод будет перенаправлен (`>`) в специальный файл `/dev/null`, в котором удаляется все, что вы в него записываете (т.е. стандартный вывод не отображается), а стандартный поток ошибок (`2`) перенаправляется (`>`) в файл `errors`.

3.4.3. Каналы

Каналы (pipes) - это, в некотором роде, комбинирование перенаправлений ввода и вывода. Принцип действия подобен физическому каналу, отсюда и такое название: один процесс отправляет данные в один конец канала, а другой процесс считывает данные на другом конце. Оператор канала - `|`. Давайте вернемся назад к примеру со списком файлов. Допустим, вам нужно сразу узнать, сколько там соответствующих

4. Вы могли подумать, что глупо говорить "файлы, оканчивающиеся на `.png`", а не просто "PNG-изображения". Однако, следует напомнить, что в UNIX® понятие расширения принято условно: расширения вовсе не обязаны определять типы файлов. Файл, оканчивающийся на `.png`, с тем же успехом может быть изображением JPEG, файлом приложения, текстовым файлом или файлом любого другого типа. Кстати, то же самое справедливо и для Windows®!

файлов, но без сохранения списка во временном файле. Тогда вам надо воспользоваться следующей командой:

```
ls images/*.png | wc -l
```

которая означает, что стандартный вывод команды `ls` (т.е. список файлов) перенаправляется на стандартный ввод команды `wc`. При этом вы получите искомый результат.

Также вы можете сразу составить список файлов “без расширений”, воспользовавшись следующей командой:

```
ls images/*.png | sed -e 's/\.png$//g' >the_list
```

или же, если вы хотите увидеть список сразу, не сохраняя его в файл:

```
ls images/*.png | sed -e 's/\.png$//g' | less
```

Действие каналов и перенаправлений не ограничивается только текстом, который может читать человек. Например, следующая команда, выполненная в Терминал’е:

```
xwd -root | convert - ~/my_desktop.png
```

отправит снимок вашего рабочего стола в файл `my_desktop.png` [5] в вашем личном каталоге.

3.5. Завершение командной строки

Завершение - это очень удобная функция, и все современные shell'ы (включая bash) обладают ее. Ее роль заключается в облегчении работы пользователя настолько, насколько это возможно. Наилучший способ демонстрации завершения - это иллюстрация на примере.

3.5.1. Пример

Допустим, что в вашем личном каталоге имеется `D09;_A_>G5=L_4;8==K<_8_A;>6=K<_4;O_22>40_8<5=5<`, и вам нужно его просмотреть. Допустим, что у вас в том же самом каталоге есть файл с именем `D09;_A_B5:AB><`. Вы находитесь в своем личном каталоге, поэтому введите следующую последовательность:

```
$ less фа<TAB>
```

(т.е., введите `less фа`, а затем нажмите клавишу **Tab**). Затем shell расширит командную строку до следующего:

```
$ less файл_с_
```

а также выведет список возможных вариантов (это его конфигурация по умолчанию, которая может быть настроена). Затем введите следующую последовательность символов:

```
less файл_с_о<TAB>
```

и shell расширит командную строку до нужного вам результата:

```
less файл_с_очень_длинным_и_сложным_для_ввода_именем
```

Затем все, что вам нужно сделать - это нажать клавишу **Enter** для подтверждения и приняться за чтение файла.

5. Да, это и в самом деле будет изображение PNG (однако должен быть установлен пакет ImageMagick).

3.5.2. Другие способы завершения

Клавиша **Tab** - это не единственный способ для выполнения завершения, хотя он и является самым общим. Как правило, завершаемым словом будет название команды для первого слова в командной строке (результатом `nsl<TAB>` будет `nslookup`), а имя файла - для всех остальных слов, если только перед словом не идет "магический" символ типа `~`, `@` или `$`. В данном случае shell попытается завершить имя пользователя, имя машины или имя переменной окружения соответственно[6]. Также существует магический символ для завершения имени файла (`/`) и команда для повторного вызова команды из истории (`!`).

Другие два способа для выполнения завершения - это последовательности **Esc-<x>** и**Ctrl-X-<x>**, где **<x>** - это один из упомянутых выше магических символов. **Esc-<x>** попробует предложить единственное завершение. В случае неудачи слово будет завершено наибольшей возможной подстрокой из списка вариантов. ***Звуковой сигнал*** (beep) означает, что выбор не является единственным, или просто нет подходящего варианта. Последовательность **Ctrl-X-<x>** выводит список возможных вариантов без попытки какого-либо завершения. Нажатие на клавишу **Tab** - это то же самое, что и последовательное нажатие **Esc-<x>** и **Ctrl-X-<x>**, где магический символ зависит от контекста.

Таким образом, единственным способом увидеть все определенные переменные окружения будет ввести в пустой строке последовательность **Ctrl-X-$**. Другой пример: если вам нужно просмотреть страницу man для команды `nslookup`, просто введите `man nsl`, а затем **Esc-!**, и командный процессор автоматически завершит команду до `man nslookup`.

3.6. Запуск и обработка фоновых процессов: управление заданиями

Вы, наверное, заметили, что, после того, как вы ввели команду в Терминал'е, вам обычно нужно дождаться завершения ее работы, прежде чем shell вернет вам управление. Это значит, что вы запустили команду в ***приоритетном режиме***. Однако, бывают случаи, когда это нежелательно.

Допустим, например, что вы решили рекурсивно скопировать один большой каталог в другой. Вы также решили игнорировать ошибки, поэтому вы перенаправили поток ошибок в `/dev/null`:

```
cp -R images/ /shared/ 2>/dev/null
```

Выполнение такой команды может занять несколько минут пока она не выполнится полностью. У вас есть два варианта решения: первый - жестокий, подразумевающий остановку (убивание) команды, а затем повторное ее выполнение, но уже в более подходящее время. Для этого нажмите **Ctrl-C**: при этом процесс будет завершен, а вы вернетесь назад к строке приглашения. Но подождите, пока что не делайте этого! Читайте дальше.

Допустим, вы хотите, чтобы команда выполнялась, а вы занимались чем-нибудь другим. Решением будет запуск процесса в ***фоновом режиме***. Для этого нажмите **Ctrl-Z**, чтобы приостановить процесс:

```
$ cp -R images/ /shared/
2>/dev/null
# Нажмите сейчас C-z
[1]+  Stopped                 cp -R images/ /shared/ 2>/dev/null
$
```

и вы снова в строке приглашения. Процесс теперь находится в режиме ожидания, ожидая вашей команды для его повторного запуска (как это отмечено ключевым словом `Stopped`). Это как раз и есть то, что вам нужно, но уже в фоновом процессе. Введите `bg` (*BackGround* - фоновый) для получения желаемого результата:

```
$ bg
      [1]+ cp -R images/ /shared/ 2>/dev/null &
      $
```

При этом процесс продолжит свою работу, но уже как фоновая задача, как это отмечено знаком `&` (амперсанд) в конце строки. Затем вы вернетесь назад в строку приглашения и сможете продолжить

6. Помните: UNIX® различает верхний и нижний регистры. Переменная окружения `HOME` и переменная `home` - это не одно и то же.

работу. Процесс, который выполняется как фоновая задача, или в фоновом режиме, называется фоновым ***заданием***.

Конечно, вы можете сразу запускать процессы как фоновые задачи, добавляя знак **&** в конце команды. Например, вы можете запустить команду копирования каталога в фоновом режиме, набрав:

```
cp -R images/ /shared/ 2>/dev/null &
```

Если хотите, вы также можете восстановить этот процесс в приоритетный режим и дождаться его завершения, набрав `fg` (*ForeGround* - приоритетный). Чтобы перевести его назад в фоновый режим, введите следующую последовательность **Ctrl**-**Z**, `bg`.

Таким способом вы можете запустить несколько заданий: каждой команде при этом будет присвоен номер задания. Команда shell'а `jobs` выводит список всех заданий, связанных с текущим shell'ом. Перед заданием ставится знак `+`, отмечающий последний процесс, запущенный в фоновом режиме. Для восстановления конкретного задания в приоритетный режим вы можете ввести команду `fg <n>`, где `<n>` - номер задания, например, `fg 5`.

Обратите внимание, что таким способом вы также можете приостанавливать или запускать ***полноэкранные*** приложения, такие как `less` или текстовый редактор Vi, и восстанавливать их в приоритетном режиме, когда вам это понадобится.

3.7. Заключительное слово

Как видите, shell обладает очень широкими возможностями и эффективное его использование является делом практики. В этой относительной длинной главе мы упомянули лишь о нескольких доступных командах: Mandrakelinux имеет тысячи утилит, и даже самые опытные пользователи используют не все из них.

Существуют утилиты на любой вкус и для любых задач: у вас есть утилиты для обработки изображений (наподобие упомянутой выше `convert`, а также GIMP ***пакетного*** режима и утилиты обработки всех ***pixmap***), звука (кодировщики Ogg Vorbis, проигрыватели звуковых CD), для записи CD, клиенты e-mail, клиенты FTP и даже веб-браузеры (типа `lynx` или `links`), не говоря уже обо всех утилитах администрирования.

Даже если существуют графические приложения с идентичными функциями, они зачастую являются графическим интерфейсами, созданными на основе этих самых утилит. В дополнение, утилиты командной строки имеют преимущество в своей способности работать в неинтерактивном режиме: вы можете поставить записываться CD и выйти из системы с уверенностью, что запись будет сделана (смотрите страницу man для nohup(1) или страницу man для screen(1)).

Глава 4. Редактирование текста: Emacs и VI

Как было сказано во введении, редактирование текста [1] - это фундаментальная особенность при использовании систем UNIX®. Два редактора, которые мы здесь собираемся кратко рассмотреть, несколько сложноваты для первоначального использования, но после того, как вы разберетесь с основами, каждый из них может стать для вас мощным инструментом. В частности, это связано с тем, что в них доступно множество режимов редактирования, предоставляющих специфические возможности для большого числа файлов разнообразных типов (perl, C++, XML и др.).

4.1. Emacs

Emacs - это, вероятно, самый мощный текстовый редактор из всех существующих. Он может делать абсолютно все, и неограниченно расширяется, благодаря своему встроенному языку программирования на базе lisp. С помощью Emacs вы можете бродить по вебу, читать почту, принимать участие в группах новостей Usenet, готовить кофе и т.п. Это не значит, что из этой главы вы узнаете, как все это делается, однако вы получите хорошие начальные знания о том, как запустить Emacs, отредактировать один или несколько файлов, сохранить их и выйти из Emacs.

Если после прочтения этого, вы захотите более подробно изучить Emacs, вы можете заглянуть сюда: Tutorial Introduction to GNU Emacs (`http://www.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html`).

4.1.1. Краткое представление

Запускается Emacs следующим образом:

```
emacs [файл1] [файл2...]
```

Emacs откроет каждый файл, указанный в качестве аргумента, в отдельном буфере. Если в командной строке были указаны два файла, окно будет автоматически разделено на два, в первом из которых будет показан последний указанный файл, а во втором - список доступных буферов. Если вы запустите Emacs без указания каких-либо файлов в командной строке, вы окажетесь в буфере под названием `*scratch*`. Если вы работаете в X, будут доступны меню при помощи мыши, а если вы в текстовом режиме, вы также можете получить доступ к меню посредством клавиши **F10**, но в этой главе мы сконцентрируемся на работе с клавиатурой без всяких меню.

4.1.2. Начало работы

А теперь пора заняться практикой. Например, давайте начнем с открытия двух файлов: `file1` и `file2`. Если эти файлы не существуют, они будут созданы после того, как вы что-нибудь запишите в них:

```
$ emacs file1 file2
```

После выполнения этой команды, будет показано следующее окно:

1. Под “редактированием текста” понимается изменение содержимого файла, состоящего только из букв, цифр и знаков пунктуации. Такими файлами могут быть электронные письма, исходные коды, документы и даже конфигурационные файлы.

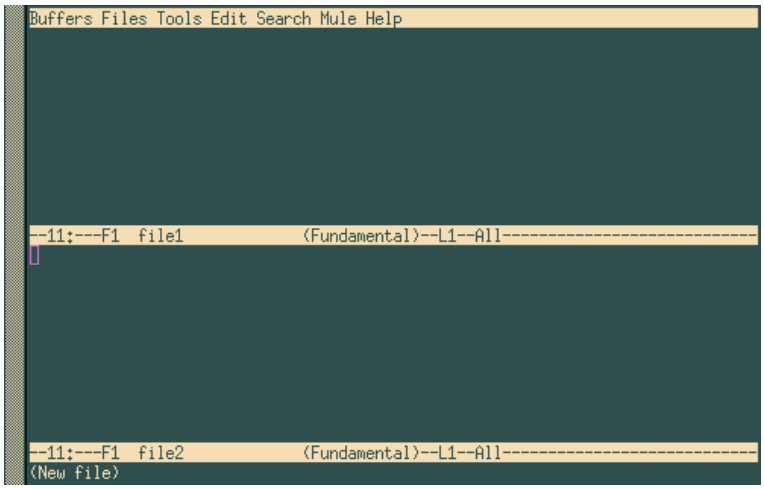


Рисунок 4-1. Редактирование двух файлов одновременно

Как вы можете видеть, были созданы два буфера. Также в нижней части экрана (где вы видите `(New file)`) находится третий буфер. Это мини-буфер. Вы не можете получить к этому буферу непосредственный доступ. Вы должны быть приглашены Emacs'ом во время интерактивных операций. Для смены текущего буфера, введите **Ctrl-X-O**. Вы можете набирать текст как в "обычном"редакторе, удаляя символы при помощи клавиш **Del** или **Backspace**.

Для перемещения курсора вы можете использовать клавиши со стрелками, или вы можете воспользоваться следующими комбинациями клавиш: **Ctrl-A** для перехода в начало строки, **Ctrl-E** для перехода в конец строки, **Alt-<** или **Ctrl-Home** для перехода в начало буфера и **Alt->** или **Ctrl-End** для перехода в конец буфера. Существует много других комбинаций, даже комбинации для каждой из клавиш со стрелками [2].

Как только вы готовы сохранить свои изменения на диск, наберите **Ctrl-X Ctrl-S**, или, если вам нужно сохранить содержимое буфера в другой файл, наберите **Ctrl-X Ctrl-W**. Emacs спросит у вас имя файла, в который должно быть записано содержимое буфера. При этом вы можете воспользоваться ***завершением***.

4.1.3. Обработка буферов

При необходимости вы можете оставить на экране только один буфер. Для этого есть два способа:

- Если вы находитесь в буфере, который надо скрыть, наберите **Ctrl-X 0**.
- Если вы находитесь в буфере, который надо оставить на экране, наберите **Ctrl-X 1**.

Для восстановления буфера обратно на экран имеется два способа:

- наберите **Ctrl-X B** и введите имя нужного вам буфера, или
- наберите **Ctrl-X Ctrl-B**. При этом будет открыт новый буфер с именем `*Buffer List*` (список буферов). Вы можете перемещаться по этому буферу при помощи последовательности **Ctrl-X O**, затем выберите нужный вам буфер и нажмите клавишу **`Enter`**, или введите в мини-буфере имя нужного буфера. После того, как вы сделаете выбор, буфер `*Buffer List*` вернется в фоновый режим.

Если вы закончили работу с файлом и хотите избавиться от связанного с ним буфера, наберите **Ctrl-X K**. После этого Emacs спросит у вас, какой из буферов он должен закрыть. По умолчанию это будет буфер, в котором вы в данный момент находитесь. Если вы хотите избавиться от другого буфера, введите его имя или нажмите **`TAB`**: Emacs откроет еще один буфер с именем `*Completions*` (завершения), предлагая список возможных вариантов. Подтвердите выбор клавишей **`Enter`**.

Вы также в любое время можете восстановить на экран два видимых буфера. Для этого наберите **Ctrl-X 2**. По умолчанию, новый созданный буфер будет копией текущего буфера (который, например, позволяет вам редактировать большой файл в разных местах "одновременно"). Для перемещения между буферами используйте описанные выше команды.

2. Emacs был разработан для работы на самых разнообразных машинах, некоторые из которых оснащены клавиатурами без клавиш со стрелками. Это еще более справедливо для Vi.

Вы в любое время можете открывать новые файлы при помощи **Ctrl-X Ctrl-F**. Emacs спросит у вас имя файла, и вы вновь можете воспользоваться автозавершением, если найдете это более удобным.

4.1.4. Копирование, вырезание, вставка, поиск

Допустим, что вы находитесь в следующей ситуации Рис. 4-2.

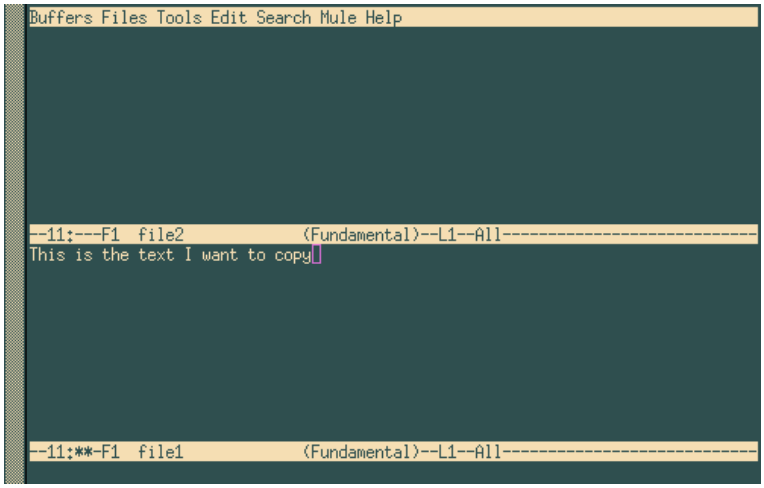


Рисунок 4-2. Emacs, перед копированием текстового блока

Сначала вам нужно выделить блок, который вы хотите скопировать. В этом примере мы хотим скопировать целое предложение. Первым шагом будет установка курсора в начало копируемой области. При условии, что курсор находится в той же позиции, что и на Рис. 4-2, последовательность команд будет следующей: **Ctrl-Space** (**`Control`** + клавиша пробел). Emacs выведет в мини-буфере сообщение `Mark set` (Метка установлена). Затем перейдите в начало строки, нажав **Ctrl-A**. Область, выделенная для копирования или вырезания - это вся область между отметкой и текущим положением курсора, следовательно, данном случае это будет целая строка текста. Доступны две последовательности команд: **Alt-W** (для копирования) или **Ctrl-W** (для вырезания). Если вы выполняете копирование, Emacs сразу вернет курсор на позицию отметки, чтобы вы могли видеть выделенную область.

И в завершение, перейдите в буфер, в который вы хотите скопировать текст, и нажмите **Ctrl-Y**. При этом вы получите следующий результат:

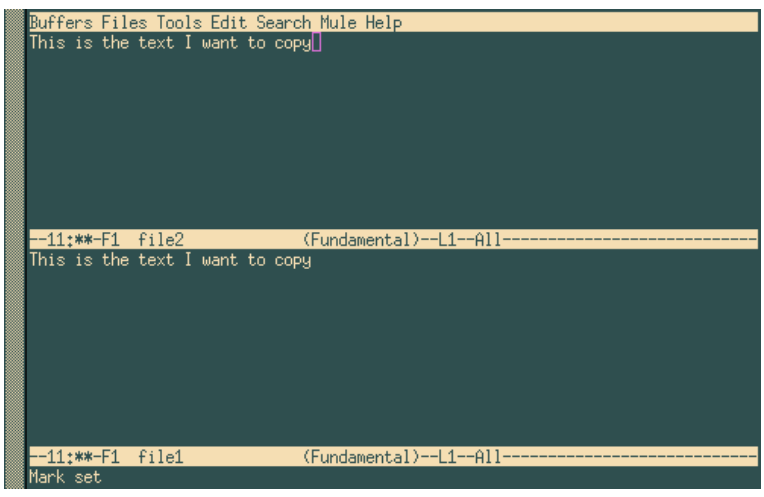


Рисунок 4-3. Копирование текста в emacs

В действительности то, что вы сейчас сделали, называется копированием текста в ***kill ring***. Этот kill ring содержит все области, скопированные или вырезанные с момента запуска Emacs. **Любая** только что скопированная или вырезанная область помещается в начало kill ring. Последовательность **Ctrl-Y** только

“вставляет” область в начало. Если вы хотите получить доступ к любой другой области, нажмите **Ctrl-Y**, затем нажимайте **Alt-Y** до тех пор, пока не получите нужный текст.

Для поиска текста перейдите в нужный буфер и нажмите **Ctrl-S**. Emacs спросит, какую строку необходимо найти. Для продолжения поиска в текущем буфере этой строки просто жмите опять **Ctrl-S**. Когда Emacs достигнет конца буфера и не найдет больше совпадений, вы можете нажать **Ctrl-S** еще раз, чтобы повторно начать поиск от начала буфера. Нажатие на **Enter** заканчивает поиск.

Для поиска и замены нажмите **Alt-%**. Emacs спросит, какую строку надо найти, на что её заменить, и будет запрашивать подтверждение для каждого найденного совпадения.

Для отмены действия нажмите **Ctrl-X U** или **Ctrl-Shift--**, что выполнит откат назад на предыдущую операцию. Вы можете отменить столько операций, сколько вам нужно.

4.1.5. Выход из emacs

Комбинация клавиш для этого - **Ctrl-X Ctrl-C**. Если вы не сохранили сделанные изменения, Emacs спросит у вас, хотите ли вы сохранить буферы или нет.

4.2. Vi: предок

Vi был первым из существующих полноэкранных редакторов. Это одна из основных программ, которая подвергается нападкам со стороны противников UNIX®, и которая в это же время является одним из лучших аргументов его защитников: хотя Vi довольно сложен в изучении, он становится чрезвычайно мощным инструментом, когда человек набирается опыта в его использовании. При помощи нескольких нажатий клавиш пользователь Vi может "свернуть горы", и, кроме Emacs, немногие текстовые редакторы могут похвастаться тем же.

На самом деле версия, поставляемая с Mandrakelinux - это Vim (от *VI iMproved*, улучшенный VI), но в этой главе мы будем называть его Vi.

Если вы хотите более подробно изучить Vi, вы можете взглянуть на эти документы: Hands-On Introduction to the Vi Editor (`http://www.library.yale.edu/wsg/docs/vi_hands_on/`) или Vim home page (`http://www.vim.org/`).

4.2.1. Режим вставки, командный режим, режим ex...

Чтобы начать изучение Vi, мы используем тот же набор команд, что и для Emacs. Поэтому давайте вернемся к нашим двум файлам и наберем:

```
$ vi файл1 файл2
```

На этом этапе вы обнаружите перед собой окно наподобие этого:

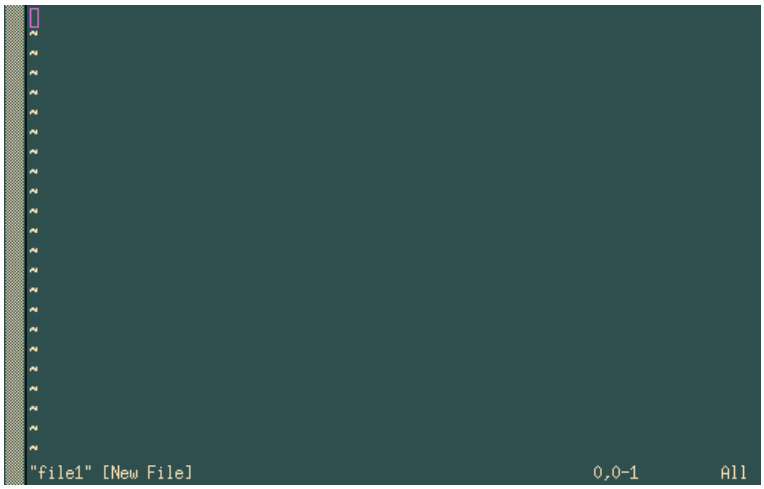


Рисунок 4-4. Исходная позиция в VIM

Сейчас вы находитесь в т.н. ***командном режиме*** с первым открытым файлом на экране. В этом режиме вы не можете вставить текст в файл. Для этого вы должны переключиться в ***режим вставки***.

Вот несколько клавиатурных команд для вставки текста:

- **a** и **i**: для вставки текста после курсора и перед ним (**A** и **I** вставляют текст в конце или начале текущей строки);
- **o** и **O**: для вставки текста под текущей строкой и над ней.

В режиме вставки вы увидите строку `--INSERT--` в нижней части экрана (так вы определяете, в каком режиме вы находитесь). Это единственный режим, который позволит вам вставить текст. Для возврата в командный режим, нажмите клавишу **Esc**.

В режиме вставки вы можете использовать клавиши **Backspace** и **DEL** для удаления текста по мере необходимости. Клавиши со стрелками позволят вам перемещаться по тексту в командном режиме и режиме вставки. В командном режиме имеются также и другие комбинации клавиш, которые мы рассмотрим позже.

Переключение в режим ex осуществляется путём нажатия в командном режиме клавиши **:**. В нижней левой части экрана появится **:** с курсором. Все, что вы введете до нажатия на клавишу **Enter**, Vi будет интерпретировать как команду ex. Если вы удалите введенную вами команду и сам символ **:**, вы будете возвращены в командный режим, а курсор вернется на исходную позицию в тексте.

Для сохранения изменений в файле, введите в командном режиме **:w**. Если вы хотите сохранить содержимое буфера в другой файл, введите **:w <имя_файла>**.

4.2.2. Обработка буферов

Для перемещения в одном и том же буфере между файлами, чьи имена были указаны в командной строке, введите **:next** для перехода к следующему файлу и **:prev** для перехода к предыдущему файлу. Вы также можете воспользоваться командой **:e <имя_файла>**, которая позволяет вам либо перейти к нужному файлу, если он уже открыт, либо открыть другой файл. Также вы можете использовать завершение имен файлов.

Также как и в Emacs, у вас на экране может быть несколько буферов. Для этого воспользуйтесь командой **:split**.

Чтобы сменить буфер, введите **Ctrl-w j** для перехода в нижний буфер, или **Ctrl-w k** для перехода в верхний буфер. Вы также можете использовать клавиши со стрелками вверх и вниз вместо **j** или **k**. Команда **:close** скрывает буфер, а команда **:q** его закрывает.

Вам следует знать, что если вы попытаетесь скрыть или закрыть буфер без сохранения сделанных изменений, команда не будет выполнена, а Vi выведет этого сообщение:

```
No write since last change (use! to override) (После последнего изменения не была выполнена запись (используйте ! для перекрытия))
```

В этом случае сделайте, как было сказано выше, и наберите **:q!** или **:close!**.

4.2.3. Редактирование текста и команды перемещения

В Vi кроме клавиш режима редактирования **Backspace** и **DEL** имеется много других команд для удаления, копирования, вставки и замены текста в командном режиме. Все приведенные ниже команды по сути делятся на две части: выполняемое действие и его эффект. Действиями могут быть:

- **c**: заменить (*Change*). Редактор удаляет запрошенный текст и после выполнения этой команды возвращается в режим вставки.
- **d**: удалить (*Delete*).
- **y**: скопировать (*Yank*). Мы рассмотрим его в следующем разделе.
- **.**: повторить последнее действие.

Эффект определяет группу символов, для которой применяется команда.

- **h**, **j**, **k**, **l**: один символ слева, внизу, вверху и справа соответственно.[3]
- **e**, **b**, **w**: конец, начало текущего слова и начало следующего слова соответственно.
- **^**, **0**, **$**: первый непустой символ, начало и конец текущей строки соответственно.
- **f<x>**: следующий найденный символ **<x>**. Например, **fe** переместит курсор к следующему найденному символу **e**.
- **/<строка>**, **?<строка>**: следующая и предыдущая найденная строка или регулярное выражение **<строка>**. Например, **/lafox** перемещает курсор к следующему слову **lafox**.
- {, }: начало и конец текущего параграфа соответственно.
- **G**, **H**: конец файла, начало экрана.

Каждому из этих "эффективных" символов или каждой из команд перемещения может предшествовать номер повторения. Для команды **G** ("Go") он означает номер строки в файле. Основываясь на этой информации, вы можете создавать любые варианты комбинаций.

Вот несколько примеров:

- **6b**: переход на 6 слов назад;
- **c8fk**: удаление всего текста до восьмого найденного символа **k**, а затем переход в режим вставки;
- **91G**: переход на 91-ю строку файла;
- **d3$**: удаление до конца текущей строки плюс следующие две строки.

Хотя многие из этих команд не очень наглядны, наилучшим способом для их запоминания будет практика. Однако, как видите, выражение "свернуть горы несколькими клавишами" не так уж и преувеличено.

4.2.4. Вырезание, копирование, вставка

В Vi имеется команда, которую мы уже видели при копировании текста - это команда **y**. Для вырезания текста используйте команду **d**. Для хранения текста выделяется 27 ячеек памяти или буферов: анонимная память и 26 ячеек с именами в виде букв алфавита в нижнем регистре.

Для использования анонимной ячейки памяти введите команду "как есть". Так, команда **y12w** скопирует в анонимную память 12 слов после курсора[4]. Используйте **d12w**, если вы хотите вырезать эту область.

Чтобы воспользоваться одной из 26 именованных ячеек памяти, введите перед командой последовательность **"<x>**, где **<x>** определяет имя ячейки памяти. Следовательно, чтобы скопировать те же самые 12 слов в ячейку памяти **k**, вы должны будете набрать **"ky12w**, или **"kd12w**, чтобы вырезать их.

---

3. Сокращенная клавиатурная команда для **dl** (удаление одного символа справа) - **x**; сокращенная клавиатурная команда для **dh** - **X**; **dd** удаляет текущую строку.
4. Но только, если курсор находится в начале первого слова!

Чтобы вставить содержимое анонимной памяти, используйте команды **p** или **P** (от слова *Paste*) для вставки текста после или до курсора соответственно. Чтобы вставить содержимое именованной ячейки памяти, таким же образом используйте **"<x>p** или **"<x>P** (например, **"dp** вставит после курсора содержимое ячейки памяти **d**).

Давайте рассмотрим пример:

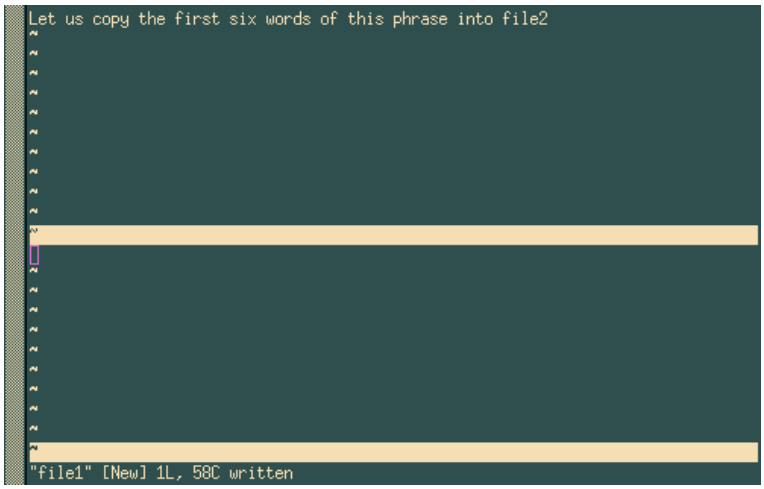


Рисунок 4-5. VIM, перед копированием текстового блока

Чтобы выполнить это действие, мы:

- повторно скопируем первые 6 слов предложения в ячейку памяти `r` (для примера): **"ry6w**[5];
- перейдем в буфер `file2`, который находится снизу: **Ctrl-w j**;
- вставим перед курсором содержимое ячейки памяти `r`: **"rp**.

Мы получили ожидаемый результат, как показано на Рис. 4-6.

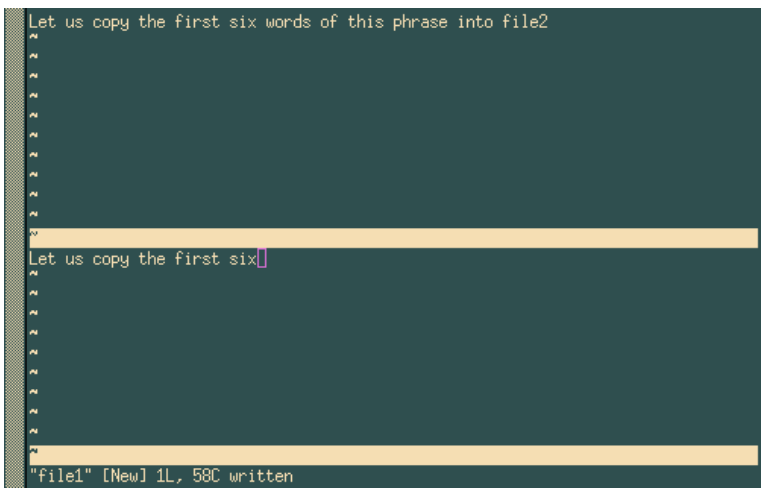


Рисунок 4-6. VIM, после копирования текстового блока

Поиск текста очень прост: просто введите в командном режиме **/**, искомую строку и нажмите клавишу **Enter**. Например, **/kyiv** будет искать строку `kyiv` от текущей позиции курсора. Нажатие на **n** переместит вас к следующему найденному совпадению, а если вы достигните конца файла, снова поиск начнется с начала файла. Для поиска в обратном направлении вместо **/** используйте **?**.

5. В буквальном смысле **y6w** означает: “Вырезать 6 слов”.

4.2.5. Выход из Vi

Команда для выхода - **`:q`** (на самом деле, как мы уже видели, она закрывает активный буфер, но если это единственный открытый буфер, вы завершите работу Vi). Имеется сокращенная клавиатурная команда, т.к. в большинстве случаев вы редактируете только один файл. Поэтому для выхода вы воспользуетесь:

- **`:wq`** или **`:x`** для сохранения изменений и выхода (более быстрым решением будет **Z Z**), или
- **`:q!`** для выхода без сохранения.

Вам следует обратить внимание на то, что если у вас открыто несколько буферов, то **`:wq`** запишет и закроет только активный буфер.

4.3. Заключительное слово...

Конечно, мы рассказали здесь гораздо больше, чем было необходимо (ведь первоначальной целью, как-никак, было редактирование текстового файла), но будем надеяться, что мы смогли показать вам некоторые возможности каждого из этих текстовых редакторов. Об этих редакторах можно еще многое рассказать, о чем свидетельствует большое число книг, посвященных каждой из этих программ.

Дайте себе немного времени, чтобы усвоить всю эту информацию, отдав предпочтение одному из этих редакторов, или изучите ровно столько, сколько посчитаете для себя нужным. Но по крайней мере теперь вы знаете, что если вы захотите пойти дальше, то всё в ваших руках.

Глава 5. Утилиты командной строки

Цель этой главы - представить небольшое число утилит командной строки, которые могут оказаться полезными для повседневного использования.

Одним из основных достоинств GNU/Linux является использование простых утилит для выполнения сложных задач. Вам уже было показано, как использовать связку команд и как вычищать выходные данные для лучшего восприятия (см. Разд. 3.4). Теперь настало время изучить еще несколько полезных утилит, которые дадут вам повышенный контроль и повышенную продуктивность.

Эта глава подразумевает под собой набор упражнений, чтобы вы могли полностью усвоить изложенные в ней функции и их использование. Поэтому каждая из команд будет продемонстрирована на примере. Не бойтесь останавливаться и консультироваться со страницами руководств по любой из этих команд. В нижней части каждой из них вы найдете раздел "SEE ALSO" ("СМ. ТАКЖЕ"), указывающие на другие интересующие вас команды. Теперь у вас будет новая область для исследования своей системы GNU/Linux!

5.1. Операции с файлами и фильтрация

Основная часть работы командной строки ведется с файлами. В этом разделе мы рассмотрим, как просматривать и фильтровать содержимое файлов, извлекать из файлов нужную информацию при помощи одной команды и сортировать содержимое файла.

5.1.1. cat, tail, head, tee: Команды для вывода содержимого файлов

Эти команды имеют почти один и тот же синтаксис: `имя_команды[опции] [файл(ы)]`, и могут быть использованы в каналах. Все они используются для вывода части файла согласно определенным критериям.

Утилита `cat` объединяет файлы и выводит результат на стандартный вывод, которым обычно является экран вашего компьютера. Это одна из наиболее часто используемых команд. Например, вы можете использовать:

```
# cat /var/log/mail/info
```

для вывода содержимого файла журнала почтового демона на стандартный вывод [1]. Команда `cat` имеет очень полезную опцию (`-n`), которая позволяет вам выводить номера строк.

Некоторые файлы, типа журналов демонов (если они запущены) обычно имеют довольно большой размер [2] и полный их вывод на экран будет не очень полезным. Вообще говоря, вам нужны только несколько строк из файла. Для этого вы можете воспользоваться командой `tail`. Следующая команда выведет (по умолчанию) последние 10 строк из файла `/var/log/mail/info`:

```
# tail /var/log/mail/info
```

Файлы типа журналов обычно динамически изменяются, т.к. демоны постоянно добавляют в них отчеты о совершенных действиях или событиях. Для наблюдения за изменениями в лог-файле в режиме реального времени вы можете воспользоваться преимуществами опции `-f`:

```
# tail -f /var/log/mail/info
```

В этом случае все изменения в файле `/var/log/mail/info` будут немедленно выводиться на экран. Использование команды `tail` с опцией `-f` весьма полезно, когда вам нужно знать, как работает ваша система. Например, наблюдая за файлом журнала `/var/log/messages`, вы сможете всегда получать обновленную информацию о системных сообщениях и различных демонах.

Если вы используете `tail` для нескольких файлов, она будет выводить в отдельной строке имена этих файлов перед выводом их содержимого. Работа с опцией `-f` при этом также возможна, что является ценным дополнением для наблюдения за взаимодействием различных частей системы.

---

1. Некоторые примеры в этом разделе основаны на реальной работе с файлами журналов некоторых серверов (служб, демонов). Убедитесь, что у вас запущен syslogd (позволяет журналировать действия демонов) и соответствующий демон (в нашем случае Postfix), и что вы в работаете под `root`'ом. Естественно вы всегда можете применять наши примеры к другим файлам.
2. Например, файл `/var/log/mail/info` содержит информацию обо всех отправленных письмах, сообщениях о выборке почты пользователями по протоколу POP и т.п.

Вы можете использовать опцию `-n` для вывода последних n строк файла. Например, для вывода последних 2-х строк наберите:

```
# tail -n2 /var/log/mail/info
```

Как и для других команд, вы можете одновременно использовать разные опции. Например, при одновременном использовании опций `-n2` и `-f` вы начнете с двух последних строк файла и будете продолжать наблюдать появление новых строк по мере добавления их в файл журнала.

Команда `head` похожа на `tail`, но она выводит первые строки файла. Следующая команда выведет (по умолчанию) первые 10 строк файла `/var/log/mail/info`:

```
# head /var/log/mail/info
```

Как и в случае с `tail` вы можете указать опцию `-n` для указания числа выводимых строк. Например, для вывода первых 2-х наберите:

```
# head -n2 /var/log/mail/info
```

Также вы можете использовать эти две команды совместно. Например, если вы хотите увидеть только строки 9 и 10, вы можете воспользоваться командой, в которой `head` выберет первые 10 строк из файла и передаст их по каналу в команду `tail`.

```
# head /var/log/mail/info | tail -n2
```

При этом последняя команда возьмет последние 2 строки и выведет их на экран. Таким же способом вы можете выбрать 20-ю от конца файла строку:

```
# tail -n20 /var/log/mail/info |head -n1
```

В этом примере мы говорим команде `tail` взять последние 20 строк и передать их по каналу в `head`. Затем команда `head` выводит на экран первую строку из полученных данных.

Допустим, что нам нужно вывести на экран результат последнего примера и сохранить его в файл `results.txt`. Нам может помочь утилита `tee`. Ее синтаксис:

```
tee [опции] [файл]
```

Теперь мы можем изменить предыдущую команду следующим образом:

```
# tail -n20 /var/log/mail/info |head -n1|tee results.txt
```

Давайте рассмотрим еще один пример. Нам нужно выбрать последние 20 строк, сохранить их в файл `results.txt`, а на экран вывести только первую из них. Тогда мы должны ввести следующее:

```
# tail -n20 /var/log/mail/info |tee results.txt |head -n1
```

У команды `tee` есть полезная опция (`-a`), которая позволяет вам дописать данные в конец существующего файла.

В следующем разделе мы увидим, как можно использовать команду grep в качестве фильтра для отделения сообщений Postfix от сообщений других служб.

5.1.2. grep: Поиск строк в файлах

Ни имя команды, ни ее аббревиатура ("General Regular Expression Parser" - синтаксический анализатор общих регулярных выражений) не являются интуитивными, однако ее действие и ее использование довольно просты: grep ищет в одном или нескольких файлах шаблон, заданный в качестве аргумента. Ее синтаксис:

```
grep [опции] <шаблон> [один или несколько файлов]
```

Если указано несколько файлов, в отображаемом результате их имена будут выводиться перед каждой найденной строкой. Вы можете использовать опцию `-h` для предотвращения вывода этих имен или вы можете использовать опцию `-l` для вывода только имен файлов с найденными совпадениями. Шаблон - это

регулярное выражение, хотя в большинстве случаев он состоит из одного единственного слова. Наиболее часто используемые опции:

- `-i`: поиск без учета регистра (т.е. игнорирование разницы между верхним и нижним регистром);
- `-v`: обратный поиск. Вывод строк, которые **не** соответствуют шаблону;
- `-n`: вывод номера строки для каждой из найденных строк;
- `-w`: сообщает grep'у, что шаблон должен совпадать со всем словом.

Итак, давайте теперь вернемся к анализу лог-файла почтового демона. Нам необходимо найти все строки в файле `/var/log/mail/info`, содержащие шаблон "postfix". Для этого мы вводим такую команду:

```
# grep postfix /var/log/mail/info
```

Команда grep может быть использована в канале. Так, мы можем получить такой же результат, что и в предыдущем примере, при помощи следующего действия:

```
# cat /var/log/mail/info | grep postfix
```

Но, пожалуйста, обратите внимание, что здесь нет необходимости использовать команду `cat`. С другой стороны комбинирование grep и `tail` является весьма интересным для сбора полезной информации в работающей системе.

Если нам нужно найти все строки, НЕ содержащие шаблона `postfix`, нам необходимо воспользоваться опцией `-v`:

```
# grep -v postfix /var/log/mail/info
```

Допустим, что нам нужно найти все сообщения об успешно отправленных письмах. В этом случае мы должны отфильтровать все строки, добавленные почтовым демоном в файл журнала (содержащие шаблон `postfix`), и они должны содержать сообщение об успешной отправке (`status=sent`)[3]:

```
# grep postfix /var/log/mail/info |grep status=sent
```

В этом случае команда grep использована дважды. Это разрешается, но выглядит не совсем красиво. Тот же результат может быть получен при помощи утилиты `fgrep`. По сути `fgrep` - это более простой способ для вызова `grep -F`. Сначала нам нужно создать файл, содержащий шаблоны, записанные в отдельной строке каждый. Такой файл может быть создан следующим образом (мы используем `patterns.txt` в качестве имени файла):

```
# echo -e 'status=sent\npostfix' >./patterns.txt
```

Проверьте результат командой `cat`. `\n` - это специальный шаблон, означающий "новую строку".

Затем мы вызываем следующую команду, в которой мы используем файл `patterns.txt` и утилиту `fgrep` вместо "двойного вызова" команды grep:

```
# fgrep -f ./patterns.txt /var/log/mail/info
```

Файл `./patterns.txt` может содержать сколько угодно шаблонов. Например, для выборки сообщений о письмах, успешно отправленных на адрес `peter@mandrakesoft.com`, достаточно будет добавить этот электронный адрес в наш файл `./patterns.txt`, выполнив следующую команду:

```
# echo 'peter@mandrakesoft.com' >>./patterns.txt
```

Понятное дело, что вы можете комбинировать команду grep с `tail` и `head`. Если нам нужно найти сообщения о предпоследнем электронном письме, отправленном на адрес `peter@mandrakesoft.competer@mandrak` мы используем:

```
# fgrep -f ./patterns.txt /var/log/mail/info | tail -n2 | head -n1
```

Здесь мы применили описанный выше фильтр и отправили результат через канал в команды `tail` и `head`. Они выбрали из данных предпоследнее значение.

3. Мы знаем, что вы скажете, что можно фильтровать просто по шаблону состояния, но тогда мы не сможем показать вам новую команду. Поэтому, пожалуйста, позвольте нам так поступить.

5.1.3. egrep: Регулярные выражения и фильтрование

С помощью grep мы ограничены шаблонами и фиксированными данными. Как нам найти все электронные письма, отправленные каждому работнику "ABC Company"? Перечисление всех их электронных адресов будет не такой уж и простой задачей, т.к. мы можем пропустить кого-либо, или нам придется вручную копаться в файле журнала.

Как и в случае с `fgrep`, grep имеет сокращенный вызов для команды `grep -E`: `egrep`. `egrep` использует регулярные выражения вместо шаблонов, предоставляя нам более мощный интерфейс для "grep'анья" текста.

Вдобавок к тому, что мы упоминали в разделе Разд. 3.3 при рассмотрении шаблонов подстановки, вот еще несколько дополнительных регулярных выражений:

- `[:alnum:]`, `[:alpha:]` и `[:digit:]` могут быть использованы вместо определения классов самих символов и представляют, соответственно: все буквы плюс все цифры, все буквы и все цифры (верхний и нижний регистры). У них есть дополнительное преимущество - они включают в себя международные символы и учитывают региональные настройки системы.
- `[:print:]` представляет все символы, которые могут быть выведены на экран.
- `[:lower:]` и `[:upper:]` представляют все буквы верхнего и нижнего регистров соответственно.

Существует много других доступных классов и вы можете просмотреть их в egrep(1). Перечисленные выше классы являются наиболее часто используемыми.

После регулярных выражений могут следовать один или несколько различных повторяющихся операторов:

?

Предшествующий элемент является необязательным и, как правило, соответствует одному вхождению.

*

Предшествующий элемент будет соответствовать 0 или более вхождениям.

+

Предшествующий элемент будет соответствовать одному или более вхождениям.

{n}

Предшествующий элемент соответствует ровно `n` вхождениям.

{n,}

Предшествующий элемент соответствует `n` или более вхождениям.

{n,m}

Предшествующий элемент соответствует как минимум `n` вхождениям, но не более `m` раз.

Если вы заключите регулярное выражение в квадратные скобки, позже вы сможете восстановить его. Допустим, что вы указали выражение `[:alpha:]+`. Оно может представлять слово. Если вы хотите определить дважды повторяющиеся слова, вы можете поместить это выражение в скобки и повторно использовать его с помощью `\1`, если это первая группа. У вас может быть до 9 таких "записей".

```
$ echo -e "abc def\nabc abc def\nabc1 abc1\nabcdef\nabcdabcd\nabcdef abcef" > testfile
$ egrep "([[:alpha:]]+) \1" testfile
abc abc def
$
```

Символы `[` и `]` являются частью имени группы, поэтому мы должны включить их, чтобы использовать этот класс символов. Первый знак `[` сообщает, что мы будем использовать группу символов, вторая скобка является частью имени группы, а затем следуют соответствующие закрывающие скобки `]`.

Единственной возвращаемой строкой будет строка, соответствующая исключительно двум группам букв, разделённых пробелом. Никакая другая группа не является вхождением регулярного выражения.

Также вы можете использовать символ `|`, определяющий вхождение для выражения слева от знака `|` или для выражения справа от этого знака. Этот оператор объединяет эти выражения. Используя созданный ранее файл **testfile**, вы можете попробовать поискать выражения, содержащие только дублирующиеся слова или содержащие дублирующиеся слова с числами:

```
$ egrep "([[:alpha:]]+) \1|([[:alpha:][:digit:]]+) \2" testfile
abc abc def
abc1 abc1
$
```

Обратите внимание, что для второй группы, использующей скобки, мы должны использовать `\2`, в противном случае выражение не будет соответствовать тому, что нам нужно. Более эффективным выражением в данном отдельно взятом случае будет:

```
$ egrep "([[:alnum:]]+) \1" testfile
abc abc def
abc1 abc1
$
```

И, в заключение, для использования определенных символов вы должны их “заэкранировать”, вставив перед ними обратную косую черту. Вот эти символы: `?`, `+`, `{`, `|`, `(`, and `)`. Для использования их в выражениях вы должны писать: `\?`, `\+`, `\{`, `\|`, `\(` и `\)`.

Эта маленькая хитрость может помочь избежать повторения вводимых слов в “вашем вашем” тексте.

Регулярные выражения во всех утилитах должны соблюдать эти (или очень похожие) правила. Потратив некоторое время на их понимание, вы во многом поможете себе при работе с другими утилитами, такими как `sed`. `sed` - это утилита, которая среди всего прочего может обрабатывать текст путем его изменения с использованием регулярных выражений в качестве правил.

5.1.4. wc: Подсчет элементов в файлах

Команда `wc` (*Word Count* - подсчет слов) используется для подсчета числа строк и слов в файлах. Она также полезна для подсчета байтов, символов и длины самой длинной строки. Ее синтаксис:

```
wc [опции] [файл(ы)]
```

Список полезных опций:

- `-l`: вывод количества строк;
- `-w`: вывод количества слов;
- `-m`: вывод общего количества символов;
- `-c`: вывод количества байт;
- `-L`: вывод длины самой длинной строки в тексте.

По умолчанию команда `wc` выводит количество строк, слов и символов. Вот несколько примеров использования:

Если нам нужно определить число пользователей в нашей системе, мы можем ввести:

```
$ wc -l /etc/passwd
```

Если нам нужно узнать число CPU в нашей системе, мы пишем:

```
$ grep "model name" /proc/cpuinfo |wc -l
```

В предыдущем разделе мы получили список сообщений об успешно отправленных письмах на адреса, перечисленные в нашем файле `./patterns.txt`. Если нам нужно узнать количество этих сообщений, мы можем перенаправить наш отфильтрованный результат через канал в команду `wc`:

```
# fgrep -f ./patterns.txt /var/log/mail/info | wc -l
```

5.1.5. sort: Сортировка содержимого файла

Ниже представлен синтаксис этой мощной утилиты для сортировки[4]:

```
sort [опции] [файл(ы)]
```

Давайте отсортируем часть файла /etc/passwd. Как видите, этот файл не отсортирован:

```
$ cat /etc/passwd
```

Если нам нужно отсортировать его по полю login, мы набираем:

```
$ sort /etc/passwd
```

По умолчанию команда sort сортирует информацию по первому полю в порядке возрастания (в нашем случае по полю login). Если нам нужно отсортировать данные в порядке убывания, используйте опцию -r:

```
$ sort -r /etc/passwd
```

Для каждого пользователя имеется свой собственный UID, записанный в файле /etc/passwd. Следующая команда сортирует файл в порядке возрастания по полю UID:

```
$ sort /etc/passwd -t":" -k3 -n
```

Здесь мы используем следующие опции sort:

- -t":": сообщает sort'у, что разделителем полей является символ ":";
- -k3: означает, что сортировка должна быть выполнена по третьему столбцу;
- -n: сообщает что выполняется сортировка числовых данных, а не буквенных.

То же самое может быть выполнено в обратном порядке:

```
$ sort /etc/passwd -t":" -k3 -n -r
```

Обратите внимание, что sort обладает двумя важными опциями:

- -u: строгая сортировка: исключаются повторяющиеся поля сортировки;
- -f: игнорирование регистра (строчные символы обрабатываются так же, как и прописные).

И, наконец, если мы хотим найти пользователя с максимальным UID, мы можем воспользоваться следующей командой:

```
$ sort /etc/passwd -t":" -k3 -n |tail -n1
```

где мы сортируем файл /etc/passwd в порядке возрастания по столбцу UID и перенаправляем результат по каналу в команду tail. Далее выводится первое значение отсортированного списка.

5.2. find: Поиск файлов по определенным критериям

find - это одна из старейших утилит UNIX®. Она предназначена для рекурсивного сканирования одного или нескольких каталогов и поиска в них файлов, соответствующих определенному набору критериев. При всей своей полезности ее синтаксис не слишком понятен, и для ее использования требуется некоторая практика. Общий синтаксис:

```
find [опции] [каталоги] [критерий1] ... [критерийN] [действие]
```

Если вы не укажете ни одного каталога, find будет выполнять поиск в текущем каталоге. Если вы не укажете критерии, это будет эквивалентно "истине", т.е. будут найдены все файлы. Опции, критерии и действия настолько многочисленны, что здесь мы упомянем только некоторые из них. Вот некоторые опции:

4. Здесь мы только вкратце рассмотрим sort. Об ее возможностях можно написать целую книгу.

- `-xdev`: не искать в каталогах, находящихся в других файловых системах.
- `-mindepth <n>`: спускаться при поиске файлов как минимум на `n` уровней ниже указанного каталога.
- `-maxdepth <n>`: искать файлы не ниже `n` уровней относительно указанного каталога.
- `-follow`: следовать по символическим ссылкам, если они ссылаются на каталоги. По умолчанию `find` не переходит по символическим ссылкам.
- `-daystart`: при использовании проверок, связанных со временем (см. ниже), вместо значения по умолчанию (24 часа назад от текущего времени) за точку отсчета принимается начало текущего дня.

Критериями могут быть одна или несколько ***атомарных*** проверок. Вот некоторые полезные проверки:

- `-type <тип_файла>`: поиск файла указанного типа. `<Типом_файла>` может быть: `f` (обычный файл), `d` (каталог), `l` (символическая ссылка), `s` (сокет), `b` (файл блочного типа), `c` (файл символьного типа) или `p` (именованный канал).
- `-name <шаблон>`: поиск файлов, чьи имена соответствуют указанному шаблону. В этой опции под шаблоном подразумевается ***подстановка имен файлов*** (см. раздел Разд. 3.3).
- `-iname <шаблон>`: эквивалент `-name`, но без учета регистра.
- `-atime <n>`, `-amin <n>`: поиск файлов, обращение к которым был выполнено `n` дней назад (`-atime`) или `n` минут назад (`-amin`). Вы также можете указать опцию `+<n>` или `-<n>`, в этом случае будет выполнен поиск файлов, обращение к которым было выполнено больше или меньше, чем `n` дней/минут назад.
- `-anewer <файл>`: поиск файлов, обращение к которым было выполнено позже, чем к `файлу`.
- `-ctime <n>`, `-cmin <n>`, `-cnewer <файл>`: эквивалент `-atime`, `-amin` и `-anewer`, но применимо к дате последнего изменения содержимого файла.
- `-regex <шаблон>`: эквивалент `-name`, но под `шаблоном` подразумевается ***регулярное выражение***.
- `-iregex <шаблон>`: эквивалент `-regex`, но без учета регистра.

Существует много других проверок. Для получения дополнительной информации обратитесь к странице руководства find(1). Проверки можно комбинировать одним из следующих способов:

- `<c1> -a <c2>`: истина, если истинны оба выражения `c1` и `c2`; опция `-a` является неявной, поэтому, если вам нужно проверить все выражения `c1`, `c2` и `c3`, вы можете ввести `<c1> <c2> <c3>`.
- `<c1> -o <c2>`: истина, если истинно любое из выражений `c1` или `c2`. Обратите внимание, что опция `-o` имеет более низкий ***приоритет***, чем `-a`, поэтому, если вам нужно найти файлы, удовлетворяющие критерию `c1` илис2 и удовлетворяющие критерию `c3`, вам понадобится использовать скобки и ввести `( <c1> -o <c2> ) -a <c3>`. Вы должны ***заэкранировать*** (дезактивировать) круглые скобки, иначе они будут интерпретированы shell'ом!
- `-not <c1>`: инвертирует проверку `c1`, поэтому `-not <c1>` будет истиной, если `<c1>` - ложь.

И, в заключение, вы можете указать действие для каждого найденного файла. Вот наиболее часто используемые:

- `-print`: просто выводит имена файлов на стандартный вывод. Это действие по умолчанию.
- `-ls`: для каждого найденного файла выводит на стандартный вывод эквивалент команды `ls -ilds`.
- `-exec <командная_строка>`: для каждого найденного файла выполняет `<командную_строку>`. `<Командная_строка>` должна заканчиваться символом `;`, который вы должны заэкранировать, чтобы shell его не интерпретировал. Положение в файле отмечается при помощи `{}`. Смотрите примеры по использованию.
- `-ok <команда>`: эквивалент `-exec`, но спрашивает подтверждение перед каждой командой.

Наилучшим способом разобраться со всеми опциями и параметрами будет рассмотрение нескольких примеров. Нам нужно найти все каталоги в `/usr/share`. Для этого введите:

```
find /usr/share -type d
```

Предположим, что у вас есть HTTP-сервер, все ваши HTML-файлы находятся в каталоге `/var/www/html`, в котором вы в данный момент находитесь. Вам нужно найти все файлы, содержимое которых не изменялось в течение месяца. Поскольку эти страницы писали разные авторы, некоторые файлы имеют расширение

`html`, а некоторые - `htm`. Вам нужно поместить ссылки на эти файлы в каталог `/var/www/obsolete`. Для этого нужно сделать следующее[5]:

```
find \( -name "*.htm" -o -name "*.html" \) -a -ctime -30 \
-exec ln {} /var/www/obsolete \;
```

Этот пример несколько сложноват и требует небольшого пояснения. Критерий поиска следующий:

```
\( -name "*.htm" -o -name "*.html" \) -a -ctime -30
```

он делает то, что нам нужно - находит все файлы, имена которых заканчиваются на `.htm` или `.html` “`\( -name "*.htm" -o -name "*.html" \)`”, **и** (`-a`) те файлы, которые не были изменены на протяжении последних 30 дней или, грубо говоря, месяца (`-ctime -30`). Обратите внимание на скобки: здесь они необходимы потому, что опция `-a` имеет более высокий приоритет. Если бы они отсутствовали, были бы найдены все файлы, заканчивающиеся на `.htm`, плюс все файлы, заканчивающиеся на `.html`, которые не были изменены в течение месяца, а это не то, что нам нужно. Также обратите внимание, что круглые скобки заэкранированы для shell'а: если бы мы ввели `( .. )` вместо `\( .. \)`, командный процессор интерпретировал бы их и попытался выполнить `-name "*.htm" -o -name "*.html"` в sub-shell'е... Другое решение - заключить круглые скобки в двойные или одинарные кавычки, но здесь предпочтительней использовать обратную косую черту, т.к. нам нужно изолировать только один символ.

И, наконец, вот команда, которая будет выполнена для каждого файла:

```
-exec ln {} /var/www/obsolete \;
```

Здесь вы также должны заэкранировать знак `;`. В противном случае командный процессор интерпретирует его как разделитель команд. Если вы забудете сделать это, `find` пожалуется, что у `-exec` отсутствует аргумент.

Последний пример: у вас есть огромный каталог (`/shared/images`), содержащий изображения всех видов. Вы регулярно используете команду touch для обновления в этом каталоге временной метки у файла с именем `stamp`, чтобы иметь привязку ко времени. Вам нужно найти все изображения ***JPEG*** более новые, чем файл `stamp`, но поскольку вы получали изображения из различных источников, эти файлы имеют расширения `jpg`, `jpeg`, `JPG` или `JPEG`. Вы также хотите избежать поиска в каталоге `old`. И вам нужно, чтобы этот список файлов был отправлен к вам по почте, а ваше имя пользователя - **`peter`**:

```
find /shared/images -cnewer     \
     /shared/images/stamp       \
     -a -iregex ".*\.jpe?g"     \
     -a -not -regex ".*/old/.*" \
       | mail peter -s "Новые изображения"
```

Конечно, эта команда не слишком полезна, если вы каждый раз должны набирать ее, и вы бы предпочли, чтобы она выполнялось регулярно. Простым способом периодического запуска команды является использование демона cron, как показано в следующем разделе.

5.3. Запуск команд по расписанию

5.3.1. crontab: Уведомления или редактирование вашего файла crontab

`crontab` позволяет вам периодически выполнять команды через определенные промежутки времени с тем преимуществом, что вам нет необходимости входить в систему. `crontab` отправит вам письмо с отчетом о выполнении вашей команды. Вы можете указывать интервалы в минутах, часах, днях и даже месяцах. В зависимости от указанных опций, `crontab` будет работать по-разному:

- `-l`: вывод вашего текущего файла `crontab`;
- `-e`: редактирование вашего файла `crontab`;
- `-r`: удаление вашего текущего файла `crontab`;

5. Обратите внимание, что в этом примере требуется, чтобы каталоги `/var/www` и `/var/www/obsolete` находились в одной файловой системе!

- `-u <пользователь>`: применение одной из перечисленных выше опций для `<пользователя>`. Это может сделать только `root`.

Давайте начнем с редактирования `crontab`'а. Если вы введете команду `crontab -e`, перед вами окажется ваш любимый текстовый редактор, если у вас установлена переменная окружения `EDITOR` или `VISUAL`, в противном случае будет использован Vi. Строка в файле `crontab` состоит из шести полей. Первые пять полей - это интервалы времени в минутах, часах, днях месяца, месяцах и днях недели соответственно. Шестое поле - это выполняемая команда. Строки, начинающиеся с # - это комментарии, они будут проигнорированы демоном `crond` (программой, отвечающей за выполнение заданий из `crontab`). Этот формат несколько отличается для системного `crontab`'а (файл `/etc/crontab`). В нем в качестве шестого поля выступает имя пользователя, которое будет использовано для запуска программы из седьмого поля. Этот файл должен быть использован только для задач администрирования и для запуска заданий пользователей, которые существуют только для обеспечения безопасности системы (такие как пользователь демона антивируса или пользователь для запуска сервера баз данных). Вот пример файла `crontab`:

Для того, чтобы напечатать его удобочитаемым шрифтом, мы должны были разорвать длинные строки. Поэтому некоторые участки кода должны быть набраны одной строкой. Когда строка заканчивается знаком \, это означает, что строка имеет продолжение. Это соглашение действительно в файлах `Makefile`, в shell'е, а также в других контекстах.

```
# Если вы не хотите получать почту, просто
#   закомментируйте следующую строку
#MAILTO="ваш_электронный_адрес"
#
# Уведомление о новых изображениях каждые 2 дня в 14:00,
#   из примера выше - после чего, выполнение "retouch"
#   для файла "stamp". Знак "%" означает
#   новую строку, это позволяет вам указывать несколько команд
#   в одной строке.
0 14 */2 * *  find /shared/images              \
  -cnewer /shared/images/stamp                 \
  -a -iregex ".*\.jpe?g"            \
  -a -not -regex                               \
    ".*/old/.*"%touch /shared/images/stamp
#
# Воспроизведение мелодии на каждое Рождество :)
0 0 25 12 * mpg123 $HOME/sounds/merryxmas.mp3
#
# Каждый вторник в 17:00 вывод списка покупок...
0 17 * * 2 lpr $HOME/shopping-list.txt
```

Существует несколько способов указания интервалов, кроме тех, что показаны в этом примере. Вы можете указать набор ***дискретных значений***, разделенных запятыми (`1,14,23`) или диапазон (`1-15`), или даже комбинировать их (`1-10,12-20`), а можно и с некоторым шагом (`1-12,20-27/2`). Теперь вашей задачей будет найти полезные команды и поместить их в этот файл!

5.3.2. at: Выполнение команды по расписанию, но только один раз

Возможно, вы хотели бы выполнить какую-нибудь команду в определенный день, но только один раз. Например, вам нужно напомнить себе о сегодняшней встрече в 18:00. Вы работаете в X, у вас установлен пакет `X11R6-contrib` и вы хотели бы получить напоминание, скажем, в 17:30 о том, что пора выходить. Тогда `at` - это то, что вам нужно:

```
$ at 17:30
  # Теперь перед вами приглашение "at"
at> xmessage "Пора идти! Встреча в 18:00"
  # Нажмите CTRL-d для выхода
at> <EOT>
job 1 at 2005-02-23 17:30
$
```

Указывать время можно разными способами:

- `now +<интервал>`: означает сейчас + интервал (Не обязателен. Отсутствие интервала означает немедленное выполнение). Синтаксис для интервала: `<n>`(minutes | hours | days | weeks | months). Здесь minutes - минуты, hours - часы, days - дни, weeks - недели и months - месяцы. Например, вы можете указать `now + 1 hour` (через один час), `now + 3 days` (через трое суток) и так далее.
- `<time> <day>`: полное указание даты. Опция `<time>` (время) является обязательной. Ее формат для `at` довольно свободный: вы, например, можете ввести `0100`, `04:20`, `2am`, `0530pm`, `1800` или одно из трех специальных значений: `noon` (полдень), `teatime` (время вечернего чая в 16:00) или `midnight` (полночь). Опция `<day>` (день) является необязательной. Вы также можете указать ее различными способами: например, `12/20/2004`, что соответствует двадцатому декабря 2004 года, или, по европейскому стандарту, `20.12.2004`. Вы можете не указывать год, но тогда допускается только европейская форма записи: `20.12`. Также вы можете записать месяц буквами: и `Dec 20`, и `20 Dec` будут верны.

Также `at` допускает использование разнообразных опций:

- `-l`: выводит список заданий, стоящих в очереди на выполнение. Первое поле представляет собой номер задания. Это эквивалент команды `atq`.
- `-d <n>`: удаляет из очереди задание под номером `<n>`. Вы можете получить номера заданий при помощи команды `atq`. Это эквивалент команды `atrm <n>`.

Как обычно, для получения дополнительной информации смотрите **страницу руководства** для at(1).

5.4. Архивирование и упаковка данных

5.4.1. tar: Архиватор для накопителей на магнитной ленте (Tape ARchiver)

`tar`, как и `find`, - это одна из старейших утилит UNIX®, поэтому ее синтаксис несколько специфичен. Вот он:

```
tar [опции] [файлы...]
```

Вот список некоторых опций. Обратите внимание, что все они имеют эквивалентную длинную запись, но вам понадобится обратиться к странице руководства tar(1), т.к. здесь мы их перечислять не будем.

Теперь в `tar` не используется начальное тире (`-`) перед короткими опциями, за исключением использования после длинной опции.

- `c`: используется для создания новых архивов.
- `x`: используется для извлечения файлов из существующего архива.
- `t`: выводит список файлов существующего архива.
- `v`: подробный режим. Выводит список файлов, добавленных или извлеченных из архива, или, в сочетании с опцией `t` (см. выше), выводит список файлов в длинном формате вместо короткого.
- `f <имя_файла>`: создает архив с именем `имя_файла`, извлекает из архива с именем `имя_файла` или выводит список файлов архива `имя_файла`. Если этот параметр отсутствует, файлом по умолчанию будет `/dev/rmt0`, который обычно является специальным файлом, связанным со ***стримером***. Если именем файла является `-` (тире), ввод или вывод данных (в зависимости от того, создается архив или выполняется извлечение из него) будет ассоциирован со стандартным вводом или стандартным выводом.
- `z`: сообщает `tar`'у, что создаваемый архив должен быть сжат при помощи `gzip`, или что архив, из которого выполняется извлечение, упакован `gzip`'ом.
- `j`: эквивалент `z`, но для упаковки используется программа `bzip2`.
- `p`: при извлечении файлов из архива сохраняет все файловые атрибуты, включая владельца, время последнего доступа и так далее. Очень полезна для дампов файловой системы.

- `r`: добавляет в существующий архив файлы, список которых указан в командной строке. Обратите внимание, что архив, к которому вы хотите добавить файлы, должен быть **не** упакованным!
- `A`: добавляет указанные в командной строке архивы в архив, определенный опцией `f`. По аналогии с опцией `r`, чтобы это сработало, архивы должны быть не упакованными.

Существует еще великое множество других опций, поэтому для получения полного списка вы можете обратиться к странице руководства tar(1). Взгляните, например, на опцию `d`.

Давайте рассмотрим пример. Допустим, вам нужно создать архив со всеми изображениями каталога `/shared/images`, упаковать его `bzip2`'ом, назвать `images.tar.bz2` и поместить в свой домашний каталог `/home`. Для этого наберите следующее:

```
#
 # Примечание: вы должны находится в каталоге,
 # файлы которого вы хотите заархивировать!
 #
$ cd /shared
$ tar cjf ~/images.tar.bz2 images/
```

Как видите, здесь мы использовали три опции: `c` сообщила `tar`'у, что нам нужно создать архив, `j` упаковала его при помощи `bzip2`, а `f ~/images.tar.bz2` создала архив в нашем домашнем каталоге с именем `images.tar.bz2`. Теперь у нас может возникнуть желание проверить целостность архива. Для этого мы можем вывести список его файлов:

```
#
 # Возвращаемся назад в наш домашний каталог
 #
$ cd
$ tar tjvf images.tar.bz2
```

Здесь мы сообщили `tar`'у вывести список (`t`) файлов архива `images.tar.bz2` (`f images.tar.bz2`), и предупредили его о том, что этот архив был упакован при помощи `bzip2` (`j`), а также о том, что нам интересно получить список в длинном формате (`v`). Теперь, предположим, что вы удалили каталог с картинками. К счастью у вас остался нетронутый архив, и теперь вы хотите распаковать его в исходное местоположение в `/shared`. Но т.к. вы не хотите нарушить работу команды `find` при поиске новых изображений, вам нужно сохранить все атрибуты файлов:

```
#
 # переход в каталог, в который вы хотите выполнить распаковку
 #
$ cd /shared
$ tar jxpf ~/images.tar.bz2
```

Вот и все!

Теперь давайте предположим, что вам нужно извлечь из архива только каталог `images/cars`. Тогда вы можете набрать следующее:

```
$ tar jxf ~/images.tar.bz2 images/cars
```

Если вы попытаетесь сделать резервную копию специальных файлов, программа `tar` поместит их в архив "как есть", не выполняя дампа их содержимого. Поэтому вы можете спокойно поместить в архив файл `/dev/mem`. Также `tar` корректно работает и со ссылками, так что не волнуйтесь насчёт этого. По поводу символических ссылок взгляните на опцию `h` в `странице руководства`.

5.4.2. bzip2 и gzip: Программы упаковки данных

Мы уже говорили об этих двух программах, когда рассматривали `tar`. В отличии от WinZip® для Windows®, архивирование и сжатие осуществляется двумя различными утилитами: `tar` для архивации и две программы для сжатия `bzip2` и `gzip`, рассмотрением которых мы сейчас и займемся. Вы также можете воспользоваться другими утилитами сжатия данных, такими как `zip`, `arj` или `rar`, которые также существуют для GNU/Linux (но весьма редко используются).

Для начала следует отметить, что `bzip2` был написан для замены `gzip`. Его степень сжатия обычно гораздо выше, но с другой стороны для него требуется больше памяти при работе. Несмотря на это, `gzip` все еще используется для совместимости со старыми системами.

Обе команды имеют похожий синтаксис:

```
gzip [опции] [файл(ы)]
```

Если не указано имя файла, и `gzip`, и `bzip2` будут ожидать данные со стандартного ввода и отправлять результат на стандартный вывод. Поэтому вы можете использовать обе программы в каналах. Они также имеют набор общих опций:

- `-1`, ..., `-9`: установка степени сжатия. Чем больше число, тем выше степень сжатия и медленнее процесс упаковки.
- `-d`: распаковка файлов. Это эквивалентно использованию утилит `gunzip` или `bunzip2`.
- `-c`: сброс на стандартный вывод результатов упаковки/распаковки файлов, указанных в виде параметров.

По умолчанию и `gzip`, и `bzip2` удаляют упакованные (или распакованные) файлы, если только вы не используете опцию `-c`. В `bzip2` вы можете избежать этого, воспользовавшись опцией `-k`. В `gzip` эквивалентной опции нет.

Теперь приведем несколько примеров. Допустим, вы хотите упаковать в текущем каталоге все файлы, названия которых заканчиваются на `.txt`, используя программу `bzip2` с максимальным коэффициентом сжатия. Вы можете сделать это так:

```
$ bzip2 -9 *.txt
```

Теперь вы хотите дать попользоваться кому-то своими картинками, но у этого человека нет `bzip2`, а есть только `gzip`. Вам нет необходимости распаковывать архив, а затем снова упаковывать его. Вы можете просто распаковать его на стандартный вывод, затем, воспользовавшись каналом, упаковать стандартный ввод и перенаправить результат в новый архив: Например, так:

```
bzip2 -dc images.tar.bz2 | gzip -9 >images.tar.gz
```

Вы могли вызвать `bzcat` вместо `bzip2 -dc`. Это аналог программы `gzip`, но называется она `zcat`, а не **gzcat**. Также в вашем распоряжении `bzless` для `bzip2`-файлов и `zless` для `gzip`, если вы хотите просматривать упакованные файлы непосредственно, без предварительной распаковки. В качестве упражнения найдите и попробуйте команды, необходимые для просмотра сжатых файлов без их распаковки и без использования утилит `bzless` или `zless`.

5.5. Больше, гораздо больше...

Существует так много команд, что книга, которая бы охватывала их все, была бы размером с солидную энциклопедию. Эта глава не охватила даже десятой части рассмотренной темы, однако вы многое можете сделать благодаря полученным здесь знаниям. При желании вы можете прочитать следующие страницы руководств: sort(1), sed(1) и zip(1L) (да, это то, о чем вы подумали: вы можете распаковывать или создавать `.zip`-архивы в GNU/Linux), convert(1) и др. Наилучшим способом изучения этих утилит является практика и эксперименты с ними. И вы, возможно, найдете много вариантов для их использования, порой даже самых неожиданных. Развлекайтесь!

Глава 6. Управление процессами

Что такое процесс, мы уже рассмотрели в Разд. 1.3. Теперь мы изучим, как получить список процессов и их характеристики, и как управлять ими.

6.1. Подробнее о процессах

За процессами можно вести наблюдение и можно сообщать им что нужно прерваться, приостановиться, продолжить работу и т.д. Чтобы понять примеры, которые мы собираемся здесь рассмотреть, будет полезным немного больше узнать о процессах.

6.1.1. Дерево процессов

По аналогии с файлами, все процессы, работающие в системе GNU/Linux, организованы в виде дерева. Корнем этого дерева является `init` - процесс системного уровня, запускаемый во время загрузки. Система присваивает номер каждому из процессов (PID, *Process ID, идентификатор процесса*), чтобы уникально их идентифицировать. Процессы также наследуют идентификаторы своих родительских процессов (PPID, *Parent Process ID, идентификатор родительского процесса*). `init` сам себе является отцом - его PID и PPID равны 1.

6.1.2. Сигналы

Каждый процесс в UNIX® может реагировать на отправленные ему сигналы. Существует 64 различных сигнала, которые идентифицируются по номерам (начиная с `1`) или по символьным именам (`SIGx`, где `x` - имя сигнала). 32 "старших" сигнала (от `33` до `64`) - это сигналы реального времени, их рассмотрение выходит за рамки этой главы. Для каждого из этих сигналов у процесса может быть определено свое собственное поведение, за исключением двух сигналов: сигнала номер 9 (`KILL`) и сигнала номер 19 (`STOP`).

Сигнал `9` безвозвратно уничтожает процесс, не оставляя ему времени на нормальное завершение работы. Этот сигнал вы отправляете процессу, который завис или вызывает другие проблемы. Полный список сигналов можно вызвать при помощи команды `kill -l`.

6.2. Информация о процессах: ps и pstree

Эти две команды выводят список процессов, запущенных на данный момент в системе, согласно установленным вами критериям. `pstree` выводит информацию в более понятном виде по сравнению с `ps -f`.

6.2.1. ps

Запуск `ps` без аргументов покажет только те процессы, что были запущены вами, и которые привязаны к используемому вами терминалу:

```
$ ps
    PID TTY          TIME CMD
  18614 pts/3    00:00:00 bash
  20173 pts/3    00:00:00 ps
```

Как и многие утилиты UNIX®, `ps` обладает рядом полезных опций, наиболее общими из которых являются:

- **a**: выводит процессы, запущенные всеми пользователями;
- **x**: выводит процессы без управляющего терминала или с управляющим терминалом, но отличающимся от используемого вами;

- **u**: выводит для каждого из процессов имя запустившего его пользователя и время запуска.

Существует еще множество других опций. За дополнительной информацией обращайтесь к странице руководства ps(1).

Вывод `ps` разделен на несколько полей: чаще всего вас будет интересовать поле `PID`, содержащее идентификатор процесса. Поле `CMD` содержит имя выполняемой команды. Чаще всего команда `ps` вызывается так:

```
$ ps ax | less
```

При этом вы получите список всех запущенных на данный момент процессов, что даст вам возможность определить один или несколько проблемных процессов и уничтожить их.

6.2.2. pstree

Команда `pstree` выводит процессы в форме дерева. Основным преимуществом является то, что вы сразу можете увидеть родительские процессы: если вам нужно уничтожить целую серию процессов, а они все происходят от одного родителя, вы можете просто убить этот родительский процесс. Вам придется воспользоваться опцией `-p` для вывода PID всех процессов и опцией `-u` для вывода имени пользователя, запустившего процесс. Т.к. дерево зачастую довольно большое, вам потребуется запустить `pstree` следующим образом:

```
$ pstree -up | more
```

При этом вы получите обзор всей структуры дерева процессов.

6.3. Отправка сигналов процессам: kill, killall и top

6.3.1. kill, killall

Эти две команды используются для отправки сигналов процессам. Для команды `kill` требуется номер процесса в качестве аргумента, а для `killall` требуется имя процесса.

Обе эти команды допускают опциональное использование аргумента с номером сигнала, отправляемого процессу. По умолчанию они обе отправляют соответствующим процессам сигнал 15 (`TERM`). Например, если вам нужно убить процесс с PID 785, используйте команду:

```
$ kill 785
```

Если вам нужно отправить ему сигнал 19 (`STOP`), введите:

```
$ kill -19 785
```

Допустим обратное, т.е. вам нужно убить процесс, для которого вы знаете имя команды. Вместо того, чтобы искать номер процесса при помощи команды `ps`, вы можете убить его по имени. Если имя процесса “mozilla”, вы можете воспользоваться командой:

```
$ killall -9 mozilla
```

В любом случае вы убьете только свои собственные процессы (только если вы не `root`), поэтому вам не стоит волноваться о процессах других пользователей, если работаете в многопользовательской системе, так как на них это не повлияет.

6.3.2. Объединение ps и kill: top

`top` - это программа, одновременно совмещающая функции `ps` и `kill`, а также используемая для наблюдения за процессами в режиме реального времени, предоставляя информацию об использовании процессора и памяти, времени работы и т.п., как показано на Рис. 6-1.

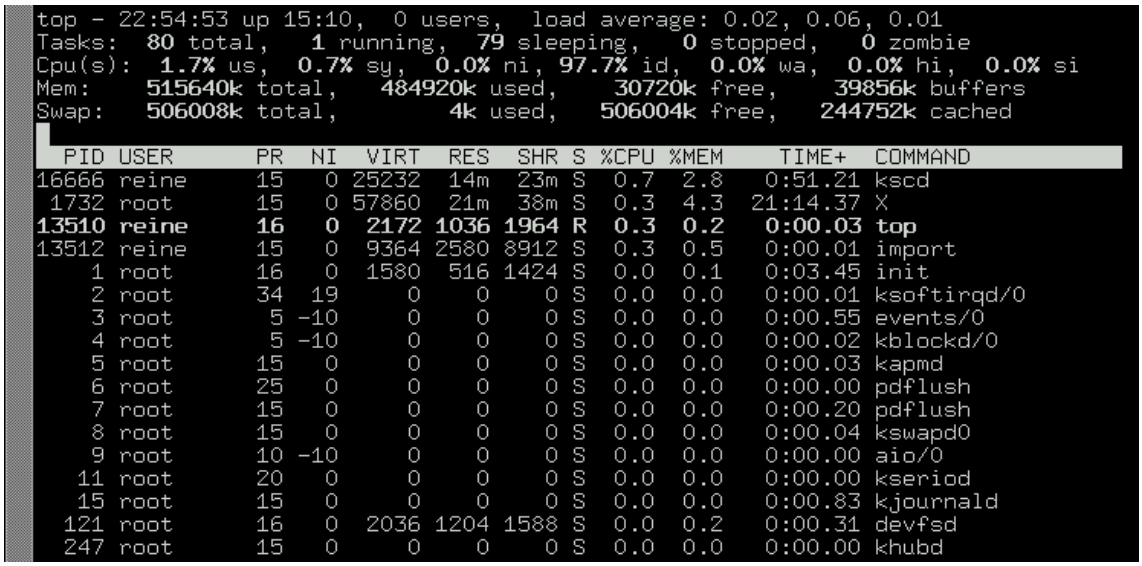


Рисунок 6-1. Наблюдение за процессами при помощи top

Утилита `top` полностью управляется с клавиатуры. Вы можете получить справку, нажав на клавишу **h**. Наиболее полезные команды:

- **k**: эта команда используется для отправки сигнала процессу. При этом `top` спросит у вас PID процесса, а затем номер или название отправляемого сигнала (по умолчанию используется `TERM` или `15`);
- **M**: эта команда используется для сортировки процессов по объему занятой ими памяти (поле `%MEM`);
- **P**: эта команда используется для сортировки процессов по занятому ими процессорному времени (поле `%CPU`;). Это метод сортировки по умолчанию;
- **u**: эта команда используется для вывода процессов заданного пользователя. `top` спросит у вас его имя. Вам необходимо ввести **имя** пользователя, а не его UID. Если вы не введете никакого имени, будут показаны все процессы;
- **i**: по умолчанию выводятся все процессы, даже спящие. Эта команда обеспечивает вывод информации только о работающих в данный момент процессах (процессы, у которых поле `STAT` имеет значение `R`, *Running*). Повторное использование этой команды вернет вас назад к списку всех процессов.
- **r**: эта команда используется для изменения приоритета выбранного процесса.

6.4. Установка приоритетов для процессов: nice, renice

Все процессы в системе работают с определенными приоритетами, также называемыми "значениями nice", которые могут изменяться от -20 (наивысший приоритет) до 19 (наименьший приоритет). Если приоритет не определен, каждый процесс будет запускаться с приоритетом по умолчанию - 0 ("базовым" приоритетом распределения машинного времени). Для процессов с более высоким приоритетом (меньшим значением nice, вплоть до -20) будет выделено больше системных ресурсов по сравнению с другими процессами с меньшим приоритетом (до 19), предоставляя им большее число циклов процессора. Пользователи, за исключением `root`'а, могут только понижать приоритет своих собственных процессов в диапазоне от 0 до 19. Суперпользователь (`root`) может установить для любого процесса любое значение приоритета.

6.4.1. renice

Если один или несколько процессов используют слишком много ресурсов системы, вы можете изменить их приоритеты вместо того, чтобы убивать их. Для этого используйте команду `renice`. Ее синтаксис:

```
renice приоритет [[-p] pid ...] [[-g] pgrp ...] [[-u] пользователь ...]
```

где `приоритет` - значение приоритета, `pid` - идентификатор процесса (используйте опцию `-p` для указания нескольких процессов), `pgrp` - идентификатор группы процесса (если их несколько, используйте `-g`) и `пользователь` - имя пользователя, владеющего процессом (`-u` для нескольких пользователей).

Допустим, что вы запустили процесс с PID 785, который выполняет длительные научные вычисления, а пока он работает, вы хотели бы немного поиграть, для чего вам нужно освободить немного системных ресурсов. Тогда вы можете набрать:

```
$ renice +15 785
```

В этом случае выполнение вашего процесса потенциально может занять больше времени, однако он не будет забирать процессорное время у других процессов.

Если вы системный администратор и видите, что какой-то пользователь запустил слишком много процессов, и они используют слишком много системных ресурсов, вы можете изменить приоритет процессов этого пользователя при помощи одной команды:

```
# renice +20 -u peter
```

После этого все процессы пользователя peter получат наименьший приоритет и не будут затруднять работу процессов других пользователей.

6.4.2. nice

Теперь, когда вы знаете о том, что можно изменять приоритеты процессов, вам может понадобиться запустить программу с определенным приоритетом. Для этого используйте команду `nice`.

В этом случае вам необходимо указать свою команду в качестве опции для `nice`. Опция `-n` используется для установки значения приоритета. По умолчанию `nice` устанавливает приоритет 10.

Например, вам нужно создать ISO-образ установочного CD-ROM'а с Mandrakelinux:

```
$ dd if=/dev/cdrom of=~/mdk1.iso
```

В некоторых системах со стандартным IDE CD-ROM процесс копирования больших объемов информации может захватить слишком много ресурсов системы. Чтобы предотвратить блокирование других процессов из-за копирования, вы можете запустить процесс с пониженным приоритетом при помощи этой команды:

```
$ nice -n 19 dd if=/dev/cdrom of=~/mdk1.iso
```

Глава 7. Организация дерева файлов

В настоящее время система UNIX® стала большой, очень большой. В особенности это касается GNU/Linux: количество доступного программного обеспечения сделало бы систему неуправляемой, если бы не было никаких руководящих принципов для организации структуры файлов в виде дерева.

Общепринятым стандартом является FHS (*Filesystem Hierarchy Standard* - стандарт иерархии файловой системы), для которого в январе 2004 была выпущена версия 2.3. Документ, описывающий стандарт, доступен в Интернете в различных форматах на веб-сайте Pathname (`http://www.pathname.com/fhs/`). Эта глава содержит только краткую сводку, но ее будет достаточно, чтобы показать вам в каком каталоге скорее всего находится определенный файл, или куда следует поместить тот или иной файл.

7.1. Разделяемые/неразделяемые, статические/переменные данные

Данные в системе UNIX® могут быть классифицированы согласно следующим критериям: разделяемые данные могут быть общими для нескольких компьютеров в сети, в то время как неразделяемые не могут. Статические данные не должны изменяться при обычном использовании, а переменные данные могут изменяться. По мере исследования структуры дерева мы будем классифицировать различные каталоги согласно этим категориям.

Эта классификация является только рекомендацией. Вам вовсе необязательно следовать ей, но принятие этих рекомендаций здорово поможет вам в управлении своей системой. Также примите во внимание, что разделение данных на статические/переменные применимо только в общем использовании системы, но не в ее конфигурации. Если вы устанавливаете программу, вам, очевидно, придется изменять каталоги типа `/usr`, которые "обычно" являются статическими.

7.2. Корневой каталог: /

Корневой каталог содержит всю иерархию системы. Он не может быть классифицирован, т.к. его подкаталоги могут быть (а могут и не быть) статическими или разделяемыми. Вот список главных каталогов и подкаталогов с их классификациями:

- `/bin`: важнейшие бинарные файлы. Он содержит базовые команды, которые могут использоваться всеми пользователями, и которые являются необходимыми для работы системы: `ls`, `cp`, `login` и др. Статический, неразделяемый.
- `/boot`: содержит файлы, необходимые для начального загрузчика GNU/Linux (GRUB или LILO для **Intel**, yaboot для PPC и т.п.). В нем может находиться (а может и нет) ядро, но если ядро в этом каталоге отсутствует, тогда оно должно быть в корневом каталоге. Статический, неразделяемый.
- `/dev`: файлы системных устройств (`dev` от англ. *DEVices*). Некоторые файлы, находящиеся в `/dev`, являются обязательными, например, `/dev/null`, `/dev/zero` и `/dev/tty`. Статический, неразделяемый.
- `/etc`: содержит все конфигурационные файлы данного компьютера. Этот каталог не может содержать бинарные файлы. Статический, неразделяемый.
- `/home`: содержит все личные каталоги пользователей системы. Этот каталог может быть разделяемым (в некоторых больших сетях к нему открывается общий доступ через NFS). Конфигурационные файлы ваших любимых приложений (типа почтовых клиентов и браузеров) располагаются в этом каталоге и начинаются с точки ("."). Например, конфигурационные файлы Mozilla находятся в каталоге `.mozilla`. Переменный, разделяемый.
- `/lib`: содержит библиотеки, жизненно необходимые для системы; в нем также хранятся модули ядра в подкаталоге `/lib/modules/`. Он содержит все библиотеки, необходимые для работы бинарных файлов из каталогов `/bin` и `/sbin`. Также в этом каталоге должны находиться: необязательный компоновщик на этапе выполнения или загрузчик `ld*`, а также динамически подключаемая библиотека C `libc.so`. Статический, неразделяемый.

- `/mnt`: содержит точки монтирования для временно монтируемых файловых систем, таких как `/mnt/cdrom`, `/mnt/floppy` и т.п. Каталог `/mnt` также используется для монтирования временных каталогов (карта USB, например, будет примонтирована в `/mnt/removable`). Переменный, неразделяемый.
- `/opt`: содержит не слишком важные для работы системы пакеты. Он зарезервирован для дополнительных пакетов; пакеты типа Adobe Acrobat Reader часто устанавливаются в `/opt`. FHS рекомендует, чтобы статические файлы (бинарники, библиотеки, страницы руководств и т.п.), устанавливаемые в каталог `/opt`, помещались в его подкаталоги `/opt/package_name`, а их конфигурационные файлы - в `/etc/opt`.
- `/root`: домашний каталог `root`'а. Переменный, неразделяемый.
- `/sbin`: содержит важные системные бинарные файлы, необходимые для запуска системы. Большинство этих файлов могут запускаться только `root`'ом. Обычный пользователь тоже может запустить их, но результат их работы может остаться нулевым. Статический, неразделяемый.
- `/tmp`: каталог предназначен для хранения временных файлов, которые могут быть создаваться отдельными программами. Переменный, неразделяемый.
- `/usr`: более подробно описан в Разд. 7.3. Статический, разделяемый.
- `/var`: место для размещения данных, которые могут изменяться программами в режиме реального времени (например, почтовые серверы, программы наблюдения, серверы печати и др.). Переменный. Отдельные его подкаталоги могут быть разделяемыми или неразделяемыми.

7.3. /usr: просто Большой каталог

Каталог `/usr` является главным каталогом для хранения приложений. Все бинарные файлы в этом каталоге не требуются для загрузки или обслуживания системы, поэтому иерархия `/usr` может, а зачастую так и есть, размещаться на отдельной файловой системе. Вследствие его (обычно) большого размера, `/usr` имеет свою собственную иерархию подкаталогов. Мы затронем только несколько из них:

- `/usr/X11R6`: полная иерархия X Window System . Все бинарные файлы и библиотеки, необходимые для работы X (включая X-серверы) должны находиться здесь. Каталог `/usr/X11R6/lib/X11` содержит все аспекты конфигурации X, которые являются общими для разных компьютеров. Индивидуальные конфигурации для каждого компьютера должны помещаться в `/etc/X11`;
- `/usr/bin`: содержит значительное большинство системных бинарных файлов. **Любая** бинарная программа, которая не является необходимой для обслуживания системы и не предназначена для системного администрирования, должна находиться в этом каталоге. Единственным исключением являются программы, которые вы самостоятельно компилируете и устанавливаете: они должны помещаться в `/usr/local`;
- `/usr/lib`: содержит все библиотеки, необходимые для запуска программ, находящихся в `/usr/bin` и `/usr/sbin`. Имеется также символическая ссылка `/usr/lib/X11`, указывающая на `/usr/X11R6/lib` - каталог, содержащий библиотеки X Window System (но только, если установлен X); [1].
- `/usr/local`: это место, куда вы должны устанавливать любые приложения, компилируемые вами из исходных кодов. Программа установки должна будет создать необходимую иерархию.
- `/usr/share`: содержит все аппаратно-независимые данные в режиме только для чтения, необходимые для приложений из `/usr`. Среди всего прочего вы найдете в нем информацию о часовых поясах и региональных стандартах (локали) (`zoneinfo` и `locale`).

Также следует упомянуть каталоги `/usr/share/doc` и `/usr/share/man`, которые соответственно содержат документацию к приложениям и системные страницы руководств.

1. Пожалуйста, обратите внимание на то, что в качестве системы X Window по умолчанию в Mandrakelinux вместо X Window System теперь используется Xorg.

7.4. /var: Изменяемые при использовании данные

Каталог `/var` содержит все рабочие данные для работающих в системе программ. В отличие от рабочих данных каталога `/tmp`, эти данные должны остаться нетронутыми в случае перезагрузки. В нем имеется много подкаталогов, вот некоторые из наиболее полезных:

- `/var/log`: содержит файлы системных журналов, которые вы можете читать для выявления неисправностей в своей системе (особенно эти два: `/var/log/messages` и `/var/log/kernel/errors`).
- `/var/run`: используется для слежения за всеми процессами, используемыми системой с момента ее загрузки, позволяя вам выполнять над ними действия в случае изменения ***уровня выполнения*** системы (см. Гл. 11).
- `/var/spool`: содержит рабочие файлы системы, ожидающие определенных действий или обработки. Например, `/var/spool/cups` содержит рабочие файлы сервера печати, а `/var/spool/mail` хранит рабочие файлы почтового сервера (например, всю входящую и исходящую почту вашей системы).

7.5. /etc: Конфигурационные файлы

`/etc` - это один из самых жизненно важных каталогов систем UNIX®, потому что он содержит все конфигурационные файлы системы, индивидуальные для каждого хоста. **Никогда** не удаляйте его для освобождения дискового пространства! Более того, если вы желаете разнести структуру вашего дерева на несколько разделов, запомните, что `/etc` не должен быть помещен на отдельный раздел: он необходим для инициализации системы и при загрузке должен находиться на загрузочном разделе.

Вот некоторые важные файлы:

- `passwd` и `shadow`: это два текстовых файла, в которых хранятся все пользователи системы и их зашифрованные пароли соответственно. Вы увидите файл `shadow` только в случае, когда используются теневые пароли, что по соображениям безопасности является опцией установки по умолчанию.
- `inittab`: это конфигурационный файл для команды `init`, которая играет основную роль в загрузке системы.
- `services`: этот файл содержит список существующих сетевых служб.
- `profile`: это общесистемный конфигурационный файл shell'а. Его настройки могут быть переопределены конфигурационными файлами shell'ов. Например, `.bashrc` для shell'а bash.
- `crontab`: конфигурационный файл для `cron` - программы, отвечающей за периодическое выполнение программ.

Для программ, которым требуется большое число конфигурационных файлов, существуют отдельные подкаталоги. Это относится, например, к X Window System, которая хранит все свои конфигурационные файлы в каталоге `/etc/X11`.

Глава 8. Файловые системы и точки монтирования

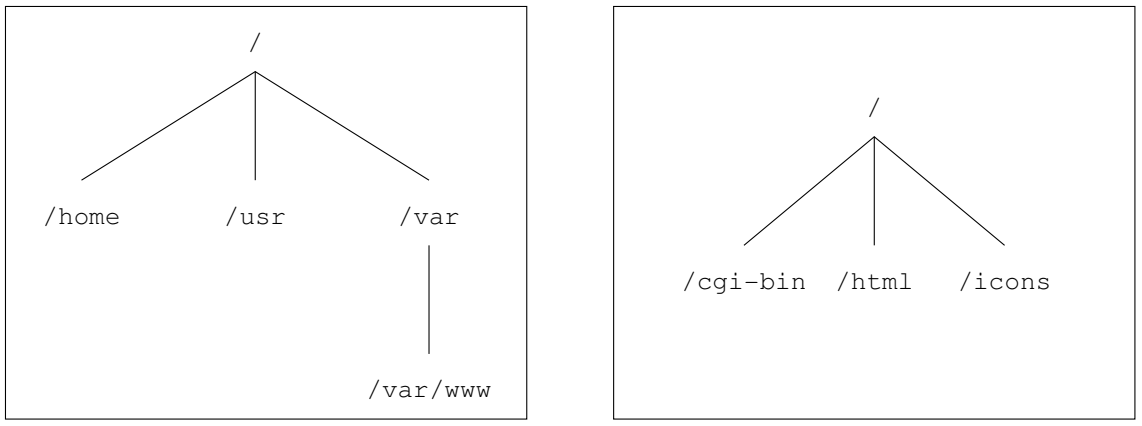
Наилучший способ разобраться с этими понятиями - рассмотреть практическое применение. Предположим вы только что приобрели новый жесткий диск и на нем нет никаких разделов. Ваш раздел Mandrakelinux полон до отказа и вместо того, чтобы начинать все сначала, вы решаете перенести целый раздел древовидной структуры [1] на новый диск. Так как новый диск обладает большей емкостью, вы решаете перенести самый большой каталог на него: `/usr`. Но сначала немного теории.

8.1. Принципы

Каждый жесткий диск разбивается на несколько разделов, каждый из которых содержит файловую систему. В то время, как Windows® назначает букву для каждой из этих файловых систем (хотя на самом деле только для тех, которые она распознает), GNU/Linux имеет уникальную древовидную структуру файлов, и каждая из файловых систем ***монтируется*** в одно местоположение этой древовидной структуры.

Также, как для Windows® нужен "Диск C:", GNU/Linux должен иметь возможность примонтировать корень своего дерева файлов (`/`) в раздел, содержащий ***корневую файловую систему***. Как только корень примонтирован, вы можете монтировать другие файловые системы древовидной структуры в различные ***точки монтирования*** этого дерева. Любой каталог в корневой структуре может выполнять роль точки монтирования, и вы можете несколько раз монтировать одну и ту же файловую систему в различные точки монтирования.

Это дает большую гибкость в настройке. Например, если вы настаиваете веб-сервер, обычным делом будет выделить целый раздел под каталог, содержащий данные веб-сервера. Каталог, который обычно содержит эти данные и выполняет роль точки монтирования раздела - это `/var/www`. Также должен быть рассмотрен вариант создания большого раздела `/home`, если вы планируете загружать большие объемы программного обеспечения. Вы можете посмотреть, как выглядит система до и после монтирования файловой системы, на Рис. 8-1 и Рис. 8-2.



ëÏÒÎÅ×ÁÑ ÆÁÊÌÏ×ÁÑ ÓÉÓÔÅÍÁ
(ÕÖÅ ÐÒÉÍÏÎÔÉÒÏ×ÁÎÁ)

æÁÊÌÏ×ÁÑ ÓÉÓÔÅÍÁ, ÓÏÄÅÒÖÁÝÁÑ
ÆÁÊÌÙ ËÁÔÁÌÏÇÁ "/var/www"
(ÅÝ£ ÎÅ ÐÒÉÍÏÎÔÉÒÏ×ÁÎÁ)

Рисунок 8-1. Файловая система еще не примонтирована

1. В нашем примере подразумевается, что все дерево находится на одном разделе.

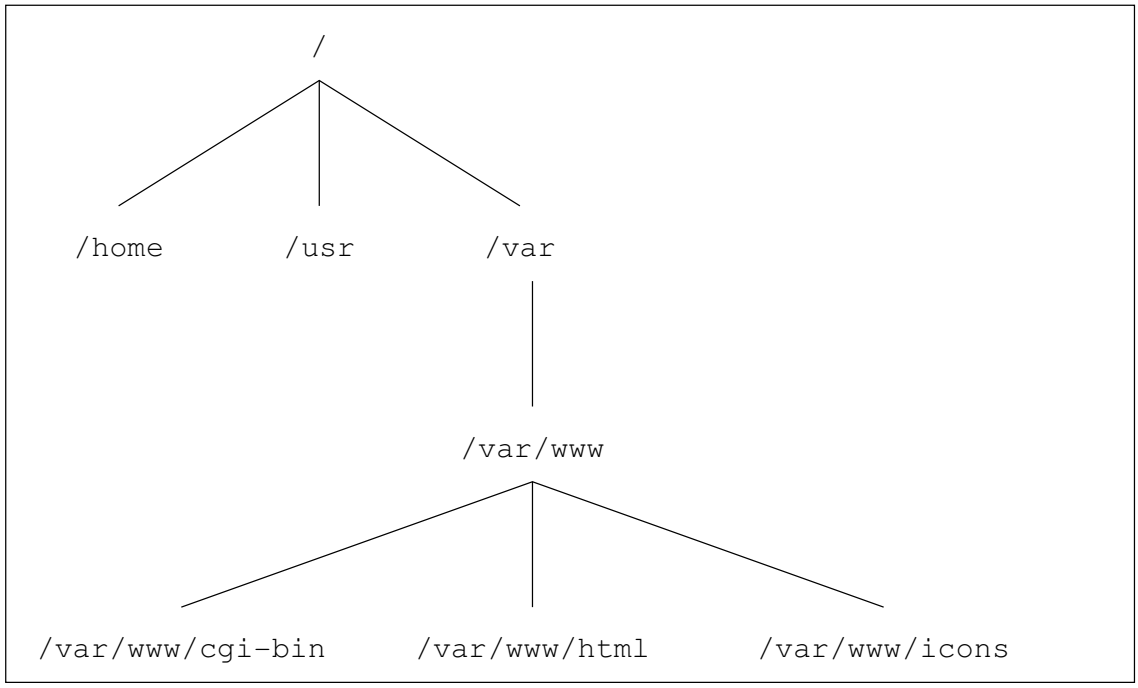


Рисунок 8-2. Файловая система теперь примонтирована

Можете себе представить, сколько это дает преимуществ: древовидная структура всегда будет оставаться одинаковой, находись она на одной или на нескольких файловых системах[2]. Эта гибкость позволяет вам перенести ключевую часть структуры дерева на другой раздел, когда появляется недостаток свободного пространства, что мы и собираемся сейчас сделать.

Имеются два обстоятельства, которые вы должны знать о точках монтирования:

1. Должен существовать каталог, выполняющий роль точки монтирования.
2. И **желательно**, чтобы этот каталог **был пустым**: если в каталоге, выбранном в качестве точки монтирования, уже есть файлы и подкаталоги, они будут просто "скрыты" новой примонтированной файловой системой. Файлы удалены не будут, но они будут недоступны до тех пор, пока вы не освободите точку монтирования.

Вообще-то доступ к "скрытым" данным можно получить при помощи заново примонтированной файловой системы. Вам просто нужно примонтировать скрытый каталог с опцией `--bind`. Например, если вы только что примонтировали какую-то файловую систему в скрытый каталог **`/hidden/directory/`** и хотите получить доступ к оригинальному его содержимому в другом каталоге **`/new/directory`**, вы должны будете выполнить:

```
mount --bind /hidden/directory/ /new/directory
```

8.2. Разметка жесткого диска, форматирование разделов

При чтении этого раздела вы должны учесть два обстоятельства: жесткий диск разбит на два раздела, и в каждом из них размещается по файловой системе. Ваш новый жесткий диск еще не имеет ни того, ни другого, поэтому начнем с разметки. Чтобы продолжать дальше, вы должны иметь права `root`'а.

Для начала вы должны узнать "имя" жесткого диска (т.е. какой файл ему соответствует). Допустим, что новый жесткий диск установлен как подчиненный на первичном интерфейсе IDE. В этом случае он будет

2. GNU/Linux может работать с большим количеством одновремено примонтированных файловых систем. На момент издания этого руководства текущее ядро Mandrakelinux (`2.6.10-3mdk`) могло работать с 256 примонтированными файловыми системами

известен под именем `/dev/hdb`.[3]. Пожалуйста, обратитесь к разделу *Управление дисковыми разделами* в книге *Стартовое руководство*, который поможет вам разметить диск. DiskDrake также может создавать файловые системы, поэтому, как только вы закончите размечать диск и создавать файловые системы, мы сможем продолжить дальше.

8.3. Команды mount и umount

Теперь, когда создана файловая система, вы можете примонтировать раздел. Изначально он будет пустым, т.к. система не имела доступа к файловой системе, чтобы добавить в нее файлы. Команда для монтирования файловых систем - `mount` , а ее синтаксис следующий:

```
mount [опции] <-t тип> [-o опции монтирования] <устройство> <точка монтирования>
```

В нашем случае, мы хотим временно примонтировать наш раздел в `/mnt/new` (или в любую другую выбранную вами точку монтирования; но помните, что точка монтирования должна существовать). Команда для монтирования нашего вновь созданного раздела:

```
$ mount -t ext3 /dev/hdb1 /mnt/new
```

Опция `-t` используется для указания типа файловой системы, которая предположительно находится на разделе. Вот файловые системы, с которыми вы будете встречаться наиболее часто: ext2FS (файловая система GNU/Linux) или ext3FS (усовершенствованная версия ext2FS с возможностями журналирования), VFAT (почти для всех разделов DOS/Windows®: FAT 12, 16 или 32), NTFS (для более новых версий Windows®) и ISO9660 (файловая система CD-ROM'ов). Если вы не указали никакого типа, `mount` попробует самостоятельно определить, какая файловая система находится в разделе, путем считывания суперблока.

Опция `-o` используется для указания одной или нескольких опций монтирования. Опции, применимые к определенным файловым системам, будут зависеть от используемой файловой системы. За более подробной информацией обращайтесь к странице руководства mount(8).

Теперь, когда вы примонтировали свой новый раздел, настало время скопировать в него весь каталог `/usr`:

```
$ (cd /usr && tar cf - .) | (cd /mnt/new && tar xpvf -)
```

После того, как файлы скопированы, мы можем отмонтировать наш раздел. Для этого воспользуйтесь командой `umount`. Ее синтаксис прост:

```
umount <точка монтирования|устройство>
```

Таким образом, чтобы отмонтировать наш новый раздел, мы можем ввести:

```
$ umount /mnt/new
```

или:

```
$ umount /dev/hdb1
```

Иногда может статься так, что устройство (обычно CD-ROM) оказывается занятым. В этом случае большинство пользователей решают эту проблему перезагрузкой своего компьютера. Однако, если команда `umount /dev/hdc` не срабатывает, вы можете попробовать "ленивый" `umount`. Ее синтаксис предельно прост:

```
umount -l <точка монтирования|устройство>
```

Этак команда по возможности отключает устройство и закрывает все открытые обработчики (handle) этого устройства. Обычно вы можете извлечь диск при помощи команды `eject <точка монтирования|устройство>`. Поэтому, если команда `eject` ничего не делает, а вы не хотите перезагружаться, используйте ленивое размонтирование.

3. Определение имени диска рассматривается в разделе Разд. 2.2.

Так как этот раздел должен "стать" нашим каталогом `/usr`, нам необходимо сообщить об этом системе. Для этого мы отредактируем файл `/etc/fstab`. Он позволяет автоматизировать монтирование определенных файловых систем, особенно при загрузке системы. В нем содержатся строки с описанием файловых систем, их точек монтирования и другими опциями. Вот пример такого файла:

```
/dev/hda2 / ext3 defaults 1 1
/dev/hdd /mnt/cdrom auto umask=0,iocharset=utf8,sync,nosuid,ro,nodev,users 0 0
/dev/fd0 /mnt/floppy auto umask=0,iocharset=utf8,sync 0 0
/dev/hda1 /mnt/windows ntfs umask=0,nls=utf8,ro 0 0
none /proc proc defaults 0 0
/dev/hda3 swap swap defaults 0 0
```

Каждая строка состоит из:

- устройства, содержащего файловую систему;
- точки монтирования;
- типа файловой системы;
- опций монтирования;
- ***флага*** для утилиты резервирования `dump`;
- порядка проверки файловой системы посредством `fsck` (*FileSystem CheckK*).

Всегда присутствует пункт для корневой файловой системы. Разделы swap являются специальными, поэтому их не видно в древовидной структуре, а поле точки монтирования для этих разделов всегда содержит ключевое слово `swap`. Что же касается файловой системы `/proc`, более подробно она будет описана в Гл. 10. Другой особой файловой системой является `/dev/pts`.

Также обратите внимание, что в вашей системе могут быть пункты, автоматически добавляемые и удаляемые из этого файла. Это выполняется командой `fstab-sync`, которая принимает специальные события от системы уровня аппаратных абстракций (HAL, Hardware Abstraction Layer), и обрабатывает файл `/etc/fstab`. Для получения дополнительной информации взгляните на страницу руководства fstab-sync(8).

Возвращаясь к изменениям нашей файловой системы, на данный момент мы перенесли всю иерархию `/usr` на `/dev/hdb1`, и теперь нам нужно, чтобы этот раздел был примонтирован как `/usr` во время загрузки системы. Для этого добавьте следующий пункт в файл `/etc/fstab`:

```
/dev/hdb1 /usr ext3 defaults 1 2
```

Теперь раздел будет монтироваться при каждой загрузке системы, а при необходимости будет проверяться на ошибки.

Если тип вашего раздела не ext3FS, вы должны будете изменить его на правильный тип. Общими опциями могут быть `ext2` и `reiserfs`. Также обратите внимание, что последнее поле содержит значение `2`. Это означает, что раздел будет проверен после всех пунктов со значением `1`, и после других файловых систем на том же жестком диске с таким же приоритетом, перечисленных до него в `/etc/fstab`. Только корневой раздел (`/`) должен иметь значение `1`.

Существуют две специальные опции: `noauto` и `users`. Опция `noauto` указывает на то, что файловая система не будет монтироваться при загрузке, а только в ручном режиме. Опция `users` указывает на то, что любой пользователь может монтировать или размонтировать эту файловую систему. Обычно эти две опции используются для приводов CD-ROM и дисководов. Существует еще много других опций, а для файла `/etc/fstab` есть своя страница `man` (fstab(5)), которую вы можете прочитать для получения дополнительной информации.

Одним из преимуществ использования `/etc/fstab` является упрощение синтаксиса команды `mount`. Чтобы примонтировать файловую систему, описанную в файле, вы можете указать просто точку монтирования или устройство. Чтобы примонтировать дискету, вы можете ввести:

```
$ mount /mnt/floppy
```

или:

```
$ mount /dev/fd0
```

Чтобы закончить с нашим примером переноса раздела, давайте повторим то, что мы уже сделали. Мы скопировали иерархию **/usr** и изменили **/etc/fstab**, чтобы новый раздел монтировался при загрузке системы. Но, минуточку, ведь старые файлы **/usr** все еще остаются на своем старом месте на диске, поэтому мы должны удалить их, чтобы освободить дисковое пространство (что и было нашей исходной целью).

- Для этого вам сначала необходимо переключиться в однопользовательский режим, выполнив в командной строке команду `telinit 1`. Она остановит все службы и сделает невозможным подключение к машине других пользователей.
- Далее мы удаляем все файлы из каталога **/usr**. Помните, что мы все еще ссылаемся на "старый" каталог, т.к. новый (большего размера) еще не примонтирован. `rm -Rf /usr/*`.
- И, наконец, мы монтируем новый каталог **/usr**: `mount /usr/`.

Вот и все. Теперь возвращаемся назад в многопользовательский режим (`telinit 3` для стандартного текстового режима или `telinit 5` для X Window System), и, если больше не осталось никакой административной работы, выходим из учетной записи `root`'а.

Глава 9. Файловая система Linux

Ваша система GNU/Linux находится на жестком диске с файловой системой. В этой главе мы обсудим различные аспекты файловых систем, доступных в GNU/Linux, а также возможности, которые они предлагают.

9.1. Сравнение нескольких файловых систем

В время установки вы можете выбрать различные файловые системы для своих разделов, таким образом они будут отформатированы с использованием различных алгоритмов.

Если вы не специалист, то выбор файловой системы не совсем понятен. Мы вкратце рассмотрим несколько современных файловых систем, доступных в Mandrakelinux.

9.1.1. Различные используемые файловые системы

9.1.1.1. Ext2

Second Extended Filesystem (сокращенно звучит как ext2FS или просто ext2) много лет была файловой системой GNU/Linux по умолчанию. Она заменила Extended File System (вот откуда в названии появилось "Second"). ext2 устраняет определенные проблемы и ограничения своего предка.

ext2 соблюдает обычные стандарты для файловых систем UNIX®-типа. С самого начала она была предназначена для дальнейшего развития, сохраняя при этом высокую отказоустойчивость и хорошую производительность.

Перед изменением размера раздела, он должен быть размонтирован.

9.1.1.2. Ext3

Как видно из названия, Third Extended File System является наследником ext2. Она совместима с последней, но была улучшена за счет добавления ***журналирования***

Одним из главных недостатков "традиционных" файловых систем типа ext2 является их низкая устойчивость к внезапным падениям системы (отключение электричества или сбой программного обеспечения). Вообще говоря, при дальнейшей перезагрузке системы такие виды событий приводят к очень долгой проверке структуры файловой системы и попыткам исправить ошибки, что иногда приводит к еще большим повреждениям. Это разрушение могло бы привести к частичной или полной потере хранимых данных.

Журналирование отвечает за решение этой проблемы. Для упрощения давайте будем говорить, что мы записываем действия (например, сохранение файла) **до** того, как они происходят на самом деле. Мы могли бы сравнить такой образ действий с тем, что ведет капитан корабля, который использует бортовой журнал для записи ежедневных событий. В результате мы получаем всегда согласованную файловую систему. А если возникают проблемы, проверка и окончательное восстановление выполняются очень быстро. Следовательно, время, потраченное на проверку файловой системы, пропорционально ее фактическому использованию и не связано с ее размером.

Таким образом, ext3 предлагает технологию журналирируемой файловой системы с сохранением структуры ext2, обеспечивая при этом отличную совместимость. Это значительно упрощает переход с ext2 на ext3 и обратно.

Как и для ext2, перед изменением размера такого раздела, он должен быть размонтирован.

9.1.1.3. ReiserFS

В отличие от ext3 `reiserfs` была написана с нуля. Это журналируемая файловая система типа ext3, но ее внутренняя структура радикально отличается, т.к. в ней используются идеи двоичного дерева, навеянные программным обеспечением для баз данных, а также применяется переменный размер блока, что делает ее оптимальной для работы с несколькими (тысячами или сотнями тысяч) небольших файлов. Она также хорошо ведет себя при работе с большими файлами, что позволяет использовать ее для разнообразных целей.

Размер такого раздела может быть изменен "на лету", без размонтирования файловой системы.

9.1.1.4. JFS

JFS - это журналируемая файловая система, разработанная и используемая в IBM. Изначально она была собственнической и закрытой, но потом корпорация IBM решила открыть к ней доступ движению за свободное программное обеспечение. Ее внутренняя структура близка к структуре `reiserfs`.

В GNU/Linux размер такого раздела не может быть изменен.

9.1.1.5. XFS

XFS - это журналируемая файловая система, разработанная в SGI, и также используемая в операционной системе Irix. Изначально она была собственнической и закрытой, но потом в SGI также решили открыть к ней доступ для движения за свободное программное обеспечение. Ее внутренняя структура имеет много разнообразных возможностей, таких как поддержка пропускной способности реального времени, экстенты (непрерывные области с прямым доступом, резервируемые для определенного набора данных) и кластерные файловые системы (но не в свободной версии).

В GNU/Linux размер такого раздела может быть изменен только в сторону увеличения. Вы не можете уменьшить его. Изменение размера может быть выполнено только для примонтированной файловой системы.

9.1.2. Различия между файловыми системами

	Ext2	Ext3	ReiserFS	JFS	XFS
Стабильность	Отличная	Очень хорошее	Хорошая	Среднее	Хорошая
Утилиты для восстановления удаленных файлов	Есть (комплекс)	Есть (комплекс)	Нет	Нет	Нет
Скорость перезагрузки после падения системы	Долго, даже очень долго	Быстро	Очень быстро	Очень быстро	Очень быстро

	Ext2	Ext3	ReiserFS	JFS	XFS
Состояние данных в случае падения системы	Вообще говоря, хорошее, но высок риск частичной или полной потери данных	Очень хорошее	Среднее[a]	Очень хорошее	Очень хорошее
Поддержка ACL	Да	Да	Нет	Нет	Да
Примечания: a. Имеется возможность улучшить результаты восстановления после падения путем журналирования **данных**, а не только **метаданных**, посредством добавления опции *data=journal* в /etc/fstab.					

Таблица 9-1. Характеристики файловой системы

Максимальный размер файла зависит от многих параметров (например, от размера блока для ext2/ext3), а также возможно дальнейшее развитие, в зависимости версии ядра и архитектуры. Согласно ограничениям файловой системы, текущий максимальный объем в настоящее время составляет порядка 2 терабайт или более (ТБ, 1ТБ=1024 ГБ) для ext2 или ext3 на стандартных 32-битных машинах. Для JFS он может составлять до 4 петабайт (ПБ, 1ПБ=1024 ТБ). К сожалению, эти значения ограничены также и максимальным размером блочного устройства[1].

В ядре 2.6.X этот предел блочного устройства может быть увеличен при использовании ядра, скомпилированного с включенной поддержкой Large Block Device (CONFIG_LBD=y). За дополнительной информацией обращайтесь к сайтам Adding Support for Arbitrary File Sizes to the Single UNIX Specification (`http://www.unix.org/version2/whatsnew/lfs.html`), Large File Support in Linux (`http://www.suse.com/~aj/linux_lfs.html`), и Large Block Devices (`http://www.gelato.unsw.edu.au/IA64wiki/LargeBlockDevices`). С помощью этой функции и поддерживающей ее файловой системы вы можете достичь емкости в 16 ТБ (на 32-битных машинах) без специальных "примочек" файловой системы, как это сделано в JFS для размера файловой системы. Хранимые в ней файлы ограничены меньшим размером.

9.1.3. А как насчет производительности?

Сравнивать производительность файловых систем всегда очень сложно. Все тесты имеют свои ограничения, и результаты должны толковаться очень осторожно. В настоящее время ext2 является очень зрелой файловой системой, но ее разработка ведется довольно медленно; ext3 и `reiserfs` с этой точки зрения созрели полностью. Новые возможности `reiserfs` включены в `reiserfs4`[2]. С другой стороны XFS имеет множество возможностей, и со временем все больше этих дополнительных возможностей работает лучше в Linux. JFS в этом отношении использует различные технологии и они интегрируются в Linux одна за одной. Это замедляет процесс, однако они также находятся в процессе завершения и с очень чистым базовым кодом. Сравнения, проведенные несколько месяцев или недель назад уже слишком устарели. Давайте не забывать, что сегодняшнее оборудование (особенно что касается емкости жестких дисков) значительно усиливает разницу между ними. XFS имеет преимущество в том, что на данный момент обладает повышенной производительностью при работе с большими файлами в потоковом режиме.

Каждая из систем обладает своими преимуществами и недостатками. В действительности все зависит от того, как вы используете свою машину. Для простой настольной машины вполне хватит ext2. Для сервера предпочтение следует отдать журналируемой файловой системе типа ext3. `reiserfs`, возможно из-за ее происхождения, больше подходит для сервера баз данных. JFS более предпочтительна в случаях, где на первом месте стоит производительность файловой системы. XFS интересна в том случае, если вам нужны ее расширенные возможности.

При "обычном" использовании, все четыре файловые системы дают приблизительно одинаковые результаты. `reiserfs` позволяет вам быстро получать доступ к маленьким файлам, но она довольно медленна при работе с большими файлами (сотни мегабайт). В большинстве случаев преимущества `reiserfs`,

1. Вас могло удивить, как достигнуть таких емкостей с жесткими дисками, объемы которых едва достигают 320-400ГБ. Воспользовавшись, например, одним контроллером RAID с 8*250ГБ дисками в аппаратном RAID 0-го уровня, вы можете достичь объема в 2ТБ. Скомбинировав хранилище из нескольких контроллеров RAID, и используя программный RAID для GNU/Linux или LVM (Logical Volume Manager - диспетчер логических томов), существует возможность выйти даже за предел (ограничение на размер блока) в 2 ТБ.
2. На момент написания этого руководства поддержка `reiserfs4` в ядре 2.6.X включена не была

полученные от способностей журналирования, превосходят ее недостатки. Обратите внимание, что по умолчанию `reiserfs` монтируется с опцией `notail`. Это означает, что отсутствует оптимизация для маленьких файлов.

9.2. Все является файлом

В книге *Стартовое руководство* вы узнали о понятиях владельца файла и прав доступа к файлу, но для того, чтобы действительно понять ***файловую систему*** UNIX® (а это также касается и файловых систем Linux), необходимо, чтобы мы заново определили понятие "Что такое файл."

Здесь "все" **действительно** означает все. Жесткий диск, раздел на жестком диске, параллельный порт, подключение к веб-сайту, карта Ethernet - все это файлы. Даже каталоги являются файлами. Linux различает много типов файлов в дополнение к стандартным файлам и каталогам. Обратите внимание, что здесь под типом файла мы не подразумеваем **содержимое** файла: в GNU/Linux, как и в любой другой системе UNIX®, файл, будь то изображение PNG, двоичный файл или что-либо еще - это просто поток байтов. Разделение файлов согласно их содержимому предоставлено приложениям.

9.2.1. Различные типы файлов

Когда вы выполняете команду `ls -l`, символ перед правами доступа определяет тип файла. Мы уже видели два типа файлов: обычные файлы (`-`) и каталоги (`d`). Когда вы бродите по дереву файлов и просматриваете содержимое каталогов, вы можете также встретить и другие типы файлов:

1. ***Файлы символьного режима***: эти файлы являются либо специальными системными файлами (типа `/dev/null`, который мы уже рассматривали), либо периферийными устройствами (последовательные или параллельные порты), характерной особенностью которых является то, что их содержимое (если оно есть) не ***буферизуется*** (т.е. оно не хранится в памяти). Такие файлы обозначаются буквой `c`.
2. ***Файлы блочного режима***: эти файлы являются периферийными устройствами, и, в отличие от символьных файлов, их содержимое **буферизуется**. Файлами этой категории, например, являются жесткие диски, разделы жесткого диска, дисководы, приводы CD-ROM и т.п. Примеры файлов блочного доступа: `/dev/hda`, `/dev/sda5`. Такие файлы обозначаются буквой `b`.
3. ***Символические ссылки***: эти файлы являются очень распространенными и широко используются в процедуре запуска системы Mandrakelinux (см. Гл. 11). Как следует из их имени, их цель - связывать файлы символическим способом. Это означает, что они являются файлами, содержащими путь к другому файлу. Они могут указывать на несуществующий файл. Очень часто их называют "***мягкими ссылками***", и обозначаются такие файлы буквой "l".
4. ***Именованные каналы***: если вас это удивило, то так оно и есть. Да, они очень похожи на каналы, используемые в командах shell'а, но с той разницей, что у этих каналов и в самом деле есть имена. Однако они очень редки и маловероятно, что вы встретите их во время своего путешествия по дереву файлов. Такие файлы обозначаются буквой `p`. Смотрите раздел Разд. 9.4.
5. ***Сокеты***: это тип файла для всех сетевых подключений, но только некоторые из них имеют названия. А самое главное, что существует несколько типов сокетов, а связь может быть установлена только через один из этих типов, но это уже выходит за рамки данной книги. Такие файлы обозначаются буквой `s`.

Вот примеры для каждого из файлов:

```
$ ls -l /dev/null /dev/sda  /etc/rc.d/rc3.d/S20random /proc/554/maps \
/tmp/ssh-queen/ssh-510-agent
crw-rw-rw-    1 root     root       1,   3 May  5  1998 /dev/null
brw-rw----    1 root     disk       8,   0 May  5  1998 /dev/sda
lrwxrwxrwx    1 root     root           16 Dec  9 19:12 /etc/rc.d/rc3.d/
  S20random -> ../init.d/random*
pr--r--r--    1 queen  queen         0 Dec 10 20:23 /proc/554/maps|
srwx------    1 queen  queen         0 Dec 10 20:08 /tmp/ssh-queen/
  ssh-510-agent=
$
```

9.2.2. Inode'ы

Inode'ы - это фундаментальная часть любой файловой системы UNIX® наряду с парадигмой "Все является файлом". Слово ***"inode"*** - это сокращение от *Information NODE* (информационный узел).

Inode'ы хранятся на диске в таблице inode. Они существуют для всех типов файлов, которые могут храниться в файловой системе, включая каталоги, именованные каналы, файлы символьного режима и так далее. Отсюда вытекает другая известная фраза: "Inode - это файл". При помощи inode'ов UNIX® идентифицирует файл уникальным способом.

Да, вы все верно прочитали: UNIX® **идентифицирует файл не по его имени**, а по номеру его inode[3]. Причина этого заключается в том, что один и тот же файл может иметь несколько имен или вообще не иметь имени. В UNIX® имя файла - это просто пункт в каталоге inode. Такой пункт называется ссылкой. Давайте рассмотрим ссылки более подробно.

9.3. Ссылки

Наилучшим способом понять, что такое ссылка, будет рассмотрение примера. Давайте создадим (обычный) файл:

```
$ pwd
/home/queen/example
$ ls
$ touch a
$ ls -il a
32555 -rw-rw-r--    1 queen     queen            0 Dec 10 08:12 a
```

Опция `-i` команды `ls` выводит номер inode в первом поле выходных данных. Как видите, до того как мы создали файл `a`, в каталоге не было никаких файлов. Однако нас интересует третье поле, которое представляет собой количество ссылок на файл (ну... на самом деле ссылок на inode).

Команду `touch a` можно разделить на два независимых действия:

- создание inode'а, которому операционная система присвоила номер 32555, и который является файлом обычного типа;
- и создание ссылки на этот inode с именем `a` в текущем каталоге `/home/queen/example`. Следовательно, файл `/home/queen/example/a` - это ссылка на inode с номером 32555, и в настоящий момент она является единственной: счетчик ссылок показывает `1`.

А теперь введем следующее:

```
$ ln a b
$ ls -il a b
32555 -rw-rw-r--    2 queen     queen            0 Dec 10 08:12 a
32555 -rw-rw-r--    2 queen     queen            0 Dec 10 08:12 b
$
```

Мы создали другую ссылку на тот же самый inode. Как видите, мы не создали файл с именем `b`. Вместо этого мы просто добавили другую ссылку на inode под номером 32555 в том же каталоге и присвоили этой новой ссылке имя `b`. Как видите в информации, выданной командой `ls -l`, счетчик ссылок для inode'а теперь равен 2, а не 1.

Теперь делаем следующее:

```
$ rm a
$ ls -il b
32555 -rw-rw-r--    1 queen     queen            0 Dec 10 08:12 b
$
```

3. **Важно**: обратите внимание, что номера inode уникальны **в пределах одной файловой системы**, т.е. inode с таким же номером может существовать в другой файловой системе. Это приводит к различению дисковых inode'ов и inode'ов "в памяти". В то время как два дисковых inode'а могут иметь одинаковые номера, если они находятся на двух различных файловых системах, inode'ы "в памяти" имеют номера, уникальные для всей системы. Например, одно из решений получения уникальности - хэш номера дискового inode'а против идентификатора блочного устройства.

Мы видим, что даже несмотря на то, что мы удалили "оригинальный файл", inode все равно существует. Но теперь единственная ссылка на этот inode - это файл с именем `/home/queen/example/b`.

Следовательно, файл в UNIX® не имеет имени; вместо этого он имеет одну или несколько ***ссылок*** в одном или нескольких каталогах.

Сами каталоги также хранятся в inode'ах. Количество ссылок на них совпадает с количеством их подкаталогов. Это является следствием того факта, что для всех каталогов существуют как минимум две ссылки: сам каталог (отображаемый как `.`) и его родительский каталог (отображаемый как `..`). Поэтому каталог с двумя подкаталогами будет иметь как минимум четыре ссылки: `.`, `..` и ссылки на каждый из подкаталогов.

Типичными примерами несвязанных файлов (т.е. не имеющих имен) являются сетевые подключения. Вы никогда не увидите файла, соответствующему вашему подключению к веб-сайту Mandrakelinux (`www.mandrakelinux.com`), в своем дереве файлов, в каком бы каталоге вы его не искали. Аналогично, когда вы используете ***канал*** в shell'е, inode, соответствующий этому каналу, существует, но ссылки на него нет. Другим примером использования inode'ов без имен являются временные файлы. Вы создаете временный файл, открываете его, а затем удаляете. Файл существует, пока он открыт, но больше его никто не может открыть (т.к. не существует имени для его открытия). Отсюда следует, что если приложение завершается аварийно, временный файл удаляется автоматически.

9.4. "Анонимные" каналы и именованные каналы

Давайте вернемся назад к примеру с каналами, поскольку он весьма интересен, а также является хорошей иллюстрацией для понимания ссылок. Когда вы в командной строке используете канал, shell создает для вас канал и работает так, что команда перед каналом выполняет в него запись, а команда после канала выполняет из него чтение. Все каналы, будь они анонимными (как те, что используются в shell'ах) или именованными (смотрите ниже), работают согласно принципу простой очереди FIFO (First In, First Out, "первым пришел - первым обслужен"). Мы уже видели примеры использования каналов в shell'е, но давайте взглянем еще на один пример для демонстрации этого принципа:

```
$ ls -d /proc/[0-9] | head -5
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
```

Одно обстоятельство, которое вы не заметите в этом примере (потому что это происходит слишком быстро), состоит в блокировке записей в каналы. Это означает, что когда команда `ls` выполняет запись в канал, он блокируется до тех пор, пока процесс выполняет чтение на другом конце. Чтобы увидеть этот эффект, вы можете создать именованные каналы, которые, в отличие от каналов, используемых shell'ами, имеют имена (т.е. они являются связанными, в то время как каналы shell'а - нет)[4]. Команда для создания именованного канала - `mkfifo`:

```
$ mkfifo a_pipe
$ ls -il
total 0
169 prw-rw-r--    1 queen     queen        0 Dec 10 14:12 a_pipe|
  #
  # Вы можете видеть, что счетчик ссылок равен 1,
  # а файл является каналом ('p').
  #
  # Вы также можете использовать здесь ln:
  #
$ ln a_pipe the_same_pipe
$ ls -il
total 0
169 prw-rw-r--    2 queen     queen        0 Dec 10 15:37 a_pipe|
169 prw-rw-r--    2 queen     queen        0 Dec 10 15:37 the_same_pipe|
$ ls -d /proc/[0-9] >a_pipe
  #
  # Процесс заблокирован, т.к. на другом конце нет считывающей программы.
  # Нажмите Control Z, чтобы приостановить процесс...
  #
```

4. Существуют и другие различия между этими двумя типами каналов, но это выходит за рамки данной книги.

```
[1]+  Stopped                 ls -d /proc/[0-9] >a_pipe
  #
  # ...Затем отправьте его в фоновый режим:
  #
$ bg
[1]+ ls -d /proc/[0-9] >a_pipe &
  #
  # теперь выполняем чтение из канала...
  #
$ head -5 <the_same_pipe
  #
  # ...процесс записи завершается....
  #
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
[1]+  Done                    ls -d /proc/[0-9] >a_pipe
$
```

Аналогичным образом чтение тоже блокируется. Если мы выполним приведенные выше команды в обратном порядке, мы увидим что команда `head` блокируется, ожидая, чтобы какой-либо процесс дал ей что-нибудь прочитать:

```
$ head -5 <a_pipe
#
# Программа заблокировалась, приостановите ее: C-z
#
[1]+  Stopped                 head -5 <a_pipe
#
# Отправляем ее в фоновый режим...
#
$ bg
[1]+ head -5 <a_pipe &
#
# ...И скармливаем ей что-нибудь :)
#
$ ls -d /proc/[0-9] >the_same_pipe
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
$
```

Вы также можете увидеть нежелательный эффект в предыдущем примере: команда `ls` завершилась до того, как вступила в действие команда `head`. В результате вы немедленно возвратились в приглашение консоли, а `head` выполнилась позже, и вы увидели ее вывод только после возвращения.

9.5. Специальные файлы: файлы символьного и блочного режима

Как уже отмечалось, такие файлы создаются либо системой, либо периферийными устройствами вашей машины. Мы также упоминали, что содержимое файлов блочного режима буферизуется, а файлы символьного режима не буферизуются. Чтобы продемонстрировать это, вставьте дискету в дисковод и дважды введите следующую команду:

```
$ dd if=/dev/fd0 of=/dev/null
```

Вы должны были увидеть следующее: при первом запуске команды было прочитано все содержимое дискеты. После второго запуска команды обращения к дисководу не было вообще. Это происходит потому, что содержимое дискеты было буферизовано при первом выполнении команды — а вы ничего не изменяли на дискете перед вторым запуском команды.

А теперь, если вы хотите распечатать большой файл таким способом (да, это будет работать):

```
$ cat /большой/пригодный/для/распечатки/файл/где-то >/dev/lp0
```

Выполнение команды займет столько времени, сколько ей потребуется, независимо от того, сколько раз вы ее запускали: один, два или пятьдесят. Это связано с тем, что `/dev/lp0` - это файл символьного режима и его содержимое не буферизуется.

Явление буферизации файлов блочного режима обладает хорошим побочным эффектом: буферизуется не только чтение, но и запись. Это позволяет записи на диск выполняться асинхронно: когда вы записываете на диск файл, операция самой записи не происходит немедленно. Она произойдет тогда, когда ядро Linux решит выполнить запись на устройство. Конечно, если вам нужно переопределить это для определенной файловой системы, взгляните на опции `sync` и `async` на странице руководства mount(8), а также на Разд. 9.7 для получения более подробной информации.

И в заключение, каждый специальный файл имеет ***старший*** и ***младший*** номера. В информации, выводимой командой `ls -l` они отображаются на месте размера файла, т.к. размер для таких файлов носит несущественный характер:

```
$ ls -l /dev/hdc /dev/lp0
brw-rw----  1 queen cdrom 22, 0 Feb 23 19:18 /dev/hdc
crw-rw----  1 lp    sys    6, 0 Feb 23 19:17 /dev/lp0
```

Здесь для файла `/dev/hdc` старший и младший номера - 22 и 0, а для файла `/dev/lp0` - 6 и 0. Обратите внимание, что эти номера уникальны для каждой категории файлов, это означает, что может существовать файл символьного режима со старшим номером 22 и младшим 0. Аналогичным образом может существовать файл блочного режима со старшим номером 6 и младшим 0. Причина существования этих номеров довольно проста: они позволяют ядру связывать соответствующие операции с этими файлами (т.е. с периферийными устройствами, на которые эти файлы ссылаются): вы не работаете с дисководом таким же образом, как, скажем, с жестким диском SCSI.

9.6. Символические ссылки. Ограничения “жестких” ссылок

Здесь мы вынуждены столкнуться с очень распространенным заблуждением (даже среди пользователей UNIX®), которое является в основном следствием того, что ссылки, как мы видели ранее (неверно называемые “жесткими” ссылками), ассоциируются только с обычными файлами (и мы видели, что это не так — так как даже символические ссылки являются “связанными”). Но для этого требуется, чтобы мы сначала разъяснили, что же представляют собой символические ссылки (часто называемые “мягкими” ссылками или даже еще чаще “симлинками”).

Символические ссылки - это файлы особого типа, единственным содержанием которых является произвольная строка, которая может указывать (а может и не указывать) на существующий файл. Когда вы обращаетесь к символической ссылке в командной строке или в программе, на самом деле вы обращаетесь к файлу, на который она указывает, если он существует. Например:

```
$ echo Hello >myfile
$ ln -s myfile mylink
$ ls -il
total 4
169 -rw-rw-r--   1 queen  queen  6 Dec 10 21:30 myfile
416 lrwxrwxrwx   1 queen  queen  6 Dec 10 21:30 mylink -> myfile
$ cat myfile
Hello
$ cat mylink
Hello
```

Как видите, тип файла `mylink` - `'l'` (от слова Link), т.е. он является символической *ссылкой*. Права доступа для символической ссылки значения не имеют: они всегда будут `rwxrwxrwx`. Вы также можете видеть, что она и файл `myfile` - это два **разных** файла, т.к. ее номер inode отличается. Но она ссылается на него символически, поэтому, когда вы вводите команду `cat mylink`, на самом деле вы выводите содержимое файла `myfile`. Чтобы продемонстрировать, что символическая ссылка содержит произвольную строку, мы можем сделать следующее:

```
$ ln -s "я не существующий файл" другойлинк
$ ls -il другойлинк
418 lrwxrwxrwx    1 queen     queen           20 Dec 10 21:43 другойлинк
-> я не существующий файл
$ cat другойлинк
cat: другойлинк: No such file or directory
$
```

Но символические ссылки существуют благодаря тому, что они преодолевают несколько ограничений, присущих ("жестким") ссылкам:

- Вы не можете создать ссылку на inode в каталоге, который находится в другой файловой системе. Причина проста: счетчик ссылки хранится в самом inode'е, а последний не может совместно использоваться в разных файловых системах. А симлинки позволяют сделать это.
- Вы не можете создать ссылки на каталоги, во избежание создания циклов в файловой системе. Но вы можете создать симлинк, указывающий на каталог и использовать его так, как если бы это на самом деле был каталог.

Поэтому символические ссылки очень полезны в различных ситуациях, и очень часто люди стремятся их использовать для связывания файлов даже тогда, когда можно использовать обычную ссылку. Одно из преимуществ обычного связывания состоит в том, что вы не потеряете файл если удалите "оригинальный".

И напоследок, если вы были внимательны, то могли заметить, что размер симлинка - это просто размер строки.

9.7. Атрибуты файлов

Подобно файловой системе FAT, имеющей атрибуты файлов (архивный, системный, скрытый, только для чтения), файловые системы GNU/Linux также имеют свои собственные атрибуты файлов, но они отличаются. Мы вкратце пройдемся по ним для полноты повествования, но используются они очень редко. Однако, если вы хотите получить действительно защищенную систему - читайте дальше.

Существуют две команды для управления атрибутами файла: `lsattr` и `chattr`. Вы, наверное, догадались, что команда `lsattr` выводит список ("LiSt") атрибутов, а команда `chattr` изменяет ("CHange") их. Эти атрибуты могут быть установлены только для каталогов и обычных файлов. Доступны следующие атрибуты:

1. `A` ("no **A**ccess time"): если для файла или каталога установлен этот атрибут, то, всякий раз при обращении к нему для чтения или записи, у него не будет обновляться время последнего доступа. Это может быть полезно, например, для файлов и каталогов, к которым очень часто обращаются для чтения, особенно из-за того, что это единственный параметр в inode, который изменяется при открытии файла только для чтения.
2. `a` ("**a**ppend only"): если для файла установлен этот атрибут, и этот файл открыт для записи, то единственной доступной операцией будет добавление данных к его предыдущему содержимому. Для каталога это означает, что вы сможете только добавить файлы, но не сможете переименовать или удалить ни одного из существующих файлов. Только `root` может установить или снять этот атрибут.
3. `d` ("no **d**ump"): `dump` - это стандартная утилита UNIX® для резервного копирования. Она делает дамп любой файловой системы, для которой счетчик дампов равен 1 в файле `/etc/fstab` (см. Гл. 8). Но если этот атрибут установлен для файла или каталога, то он, в отличие от других, будет пропущен при снятии дампа. Обратите внимание, что при установке его для каталогов, это также распространяется на все их подкаталоги и файлы.
4. `i` ("**i**mmutable"): файл или каталог с этим атрибутом вообще не может быть изменен: он не может быть переименован, на него не может быть создана ссылка[5] и он не может быть удален. Только `root` может установить или снять этот атрибут. Обратите внимание, что это также предотвращает изменение времени последнего доступа, поэтому вам нет необходимости устанавливать атрибут `A`, если установлен `i`.
5. `s` ("**s**ecure deletion"): когда удаляется файл или каталог с этим атрибутом, блоки, которые он занимал на диске перезаписываются нулями.
6. `S` ("**S**ynchronous mode"): если для файла или каталога установлен этот атрибут, все его изменения синхронизируются и немедленно записываются на диск.

К примеру, вы можете установить атрибут `i` на жизненно важные системные файлы, чтобы избежать неприятных сюрпризов. Также, рассмотрите вариант установки атрибута `A` на страницы руководств:

5. Убедитесь, что вы поняли, что означает "добавление ссылки" на файл и на каталог!

это позволит избежать многих дисковых операций, и, в частности, может немного продлить жизнь аккумуляторов портативных компьютеров.

Глава 10. Файловая система /proc

Файловая система /proc является особой для GNU/Linux. Это виртуальная файловая система, поэтому файлы, которые вы найдете в этом каталоге, на самом деле не занимают места на вашем жестком диске. Это очень удобный способ для получения информации о системе, в особенности из-за того, что большинство файлов в этом каталоге удобочитаемы для человека (ну, с некоторой помощью). В действительности многие программы собирают информацию из файлов в /proc, форматируют ее своим собственным способом, а результат затем выводят на экран. Существует несколько программ, которые поступают именно так при выводе информации о процессах (top, ps и их товарищи). /proc - это также хороший источник информации о вашем аппаратном обеспечении, и, по аналогии с программами, показывающими процессы, некоторые программы являются просто интерфейсами к информации, находящейся в /proc.

Также существует специальный подкаталог /proc/sys. Он позволяет вам отображать параметры ядра и изменять их в режиме реального времени.

10.1. Информация о процессах

Если вы выведете список содержимого каталога /proc, вы увидите много каталогов, именами которых являются номера. Эти каталоги содержат информацию о всех процессах в системе, запущенных в данный момент:

```
$ ls -d /proc/[0-9]*
/proc/1/    /proc/302/  /proc/451/  /proc/496/  /proc/556/  /proc/633/
/proc/127/  /proc/317/  /proc/452/  /proc/497/  /proc/557/  /proc/718/
/proc/2/    /proc/339/  /proc/453/  /proc/5/    /proc/558/  /proc/755/
/proc/250/  /proc/385/  /proc/454/  /proc/501/  /proc/559/  /proc/760/
/proc/260/  /proc/4/    /proc/455/  /proc/504/  /proc/565/  /proc/761/
/proc/275/  /proc/402/  /proc/463/  /proc/505/  /proc/569/  /proc/769/
/proc/290/  /proc/433/  /proc/487/  /proc/509/  /proc/594/  /proc/774/
/proc/3/    /proc/450/  /proc/491/  /proc/554/  /proc/595/
```

Обратите внимание, что как пользователь, вы (рассуждая логически) можете вывести информацию только о своих собственных процессах, но не других пользователей. Поэтому войдите в систему под root'ом и посмотрите, какая информация доступна для процесса 1, которым является процесс init и который отвечает за запуск всех остальных процессов:

```
$ su
Password:
$ cd /proc/1
$ ls -l
total 0
dr-xr-xr-x  2 root root 0 Feb 15 18:14 attr/
-r--------  1 root root 0 Feb 15 18:14 auxv
-r--r--r--  1 root root 0 Feb 15 18:14 cmdline
lrwxrwxrwx  1 root root 0 Feb 15 18:14 cwd -> //
-r--------  1 root root 0 Feb 15 18:14 environ
lrwxrwxrwx  1 root root 0 Feb 15 18:14 exe -> /sbin/init*
dr-x------  2 root root 0 Feb 15 18:14 fd/
-r--r--r--  1 root root 0 Feb 15 18:14 maps
-rw-------  1 root root 0 Feb 15 18:14 mem
-r--r--r--  1 root root 0 Feb 15 18:14 mounts
lrwxrwxrwx  1 root root 0 Feb 15 18:14 root -> //
-r--r--r--  1 root root 0 Feb 15 18:14 stat
-r--r--r--  1 root root 0 Feb 15 18:14 statm
-r--r--r--  1 root root 0 Feb 15 18:14 status
dr-xr-xr-x  3 root root 0 Feb 15 18:14 task/
-r--r--r--  1 root root 0 Feb 15 18:14 wchan
$
```

Каждый из каталогов содержит одинаковые пункты: Вот краткое описание некоторых из них:

1. cmdline: этот (псевдо-) файл содержит полную командную строку, использованную для вызова процесса. Он не отформатирован: между программой и ее аргументами нет пробелов, а в конце строки нет разделителя строки. Чтобы просмотреть его, вы можете использовать: perl -ple 's,\00, ,g' cmdline.
2. cwd: эта символическая ссылка указывает на текущий рабочий каталог процесса (следует из имени).

3. `environ`: этот файл содержит все переменные окружения, определенные для этого процесса, в виде `ПЕРЕМЕННАЯ=значение`. Как и в `cmdline`, вывод вообще не отформатирован: нет разделителей строк для отделения различных переменных, и в конце нет разделителя строки. Одно из решений для его просмотра: `perl -ple 's,\00,\n,g' environ`.
4. `exe`: эта символическая ссылка указывает на исполняемый файл, соответствующий запущенному процессу.
5. `fd`: этот подкаталог содержит список файловых дескрипторов, открытых в данный момент процессом. Смотрите ниже.
6. `maps`: когда вы выводите содержимое этого именованного канала (при помощи команды `cat`, например), вы можете увидеть части адресного пространства процесса, которые в текущий момент распределены для файла. Вот эти поля (слева направо): адресное пространство, связанное с этим распределением; права доступа, связанные с этим распределением; смещение от начала файла, где начинается распределение; старший и младший номера (в шестнадцатиричном виде) устройства, на котором находится распределенный файл; номер inode файла; и, наконец, имя самого файла. Если устройство обозначено как 0 и отсутствует номер inode или имя файла, тогда это анонимное распределение. Смотрите mmap(2).
7. `root`: эта символическая ссылка указывает на корневой каталог, используемый процессом. Обычно это будет `/`, однако рекомендуем вам просмотреть chroot(2).
8. `status`: этот файл содержит разнообразную информацию о процессе: имя исполняемого файла, его текущее состояние, его PID и PPID, его реальные и эффективные UID и GID, его использование памяти и другие данные. Обратите внимание, что файлы `stat` и `statm` теперь устарели. Информация, которая в них содержалась, теперь хранится в `status`.

Если мы выведем список содержимого каталога `fd` для случайно выбранного процесса, например, для процесса 127, мы получим следующее:

```
$ ls -l /proc/127/fd
total 0
lrwx------    1 root     root           64 Dec 16 22:04 0 -> /dev/console
l-wx------    1 root     root           64 Dec 16 22:04 1 -> pipe:[128]
l-wx------    1 root     root           64 Dec 16 22:04 2 -> pipe:[129]
l-wx------    1 root     root           64 Dec 16 22:04 21 -> pipe:[130]
lrwx------    1 root     root           64 Dec 16 22:04 3 -> /dev/apm_bios
lr-x------    1 root     root           64 Dec 16 22:04 7 -> pipe:[130]
lrwx------    1 root     root           64 Dec 16 22:04 9 ->
/dev/console
$
```

На самом деле это список дескрипторов файла, открытых процессом. Каждый открытый дескриптор представлен в виде символической ссылки, где имя - это номер дескриптора, который указывает на файл, открытый этим дескриптором[1]. Обратите внимание на права доступа к симлинкам: это - единственное место, где они имеют смысл, поскольку они представляют собой права, с которыми был открыт файл, соответствующий дескриптору.

10.2. Информация об аппаратном обеспечении

Кроме каталогов, связанных с различными процессами, в `/proc` также содержится огромный объем информации об аппаратном обеспечении вашей машины. Список файлов каталога `/proc` выглядит следующим образом:

```
$ ls -d [a-z]*
apm         devices      interrupts  loadavg  partitions  sysrq-trigger
asound/     diskstats    iomem       locks    pci         sysvipc/
bluetooth/  dma          ioports     mdstat   scsi/       tty/
buddyinfo   driver/      irq/        meminfo  self@       uptime
bus/        execdomains  kallsyms    misc     slabinfo    version
cmdline     fb           kcore       modules  splash      vmstat
cpufreq     filesystems  keys        mounts@  stat
cpuinfo     fs/          key-users   mtrr     swaps
```

1. Если вы помните, о чем говорилось в разделе Разд. 3.4, тогда вы должны знать, что означают дескрипторы `0`, `1` и `2`. Дескриптор `0` - это стандартный ввод, дескриптор `1` - стандартный вывод и дескриптор `2` - стандартный поток ошибок.

```
crypto      ide/         kmsg        net/     sys/
```

Например, если мы посмотрим на содержимое **/proc/interrupts**, мы можем увидеть, что он содержит список прерываний, используемых в данный момент системой, а также периферийные устройства, которые их используют. Аналогичным образом, **ioports** содержит список занятых в данный момент диапазонов адресов ввода-вывода, и, наконец, **dma** делает то же самое для каналов DMA. Поэтому, чтобы выловить конфликт, просмотрите содержимое этих трех файлов:

```
$ cat interrupts
           CPU0
  0:   73751906    IO-APIC-edge  timer
  2:          0          XT-PIC  cascade
  3:      44301    IO-APIC-edge  NVidia CK8
  9:     115618    IO-APIC-edge  ohci_hcd
 10:    7758240    IO-APIC-edge  ohci_hcd, eth0
 11:     218753    IO-APIC-edge  libata, ehci_hcd
 12:    1153980    IO-APIC-edge  i8042
 15:       3419    IO-APIC-edge  ide1
NMI:          0
LOC:   73749577
ERR:          0
MIS:          0

$ cat ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
0376-0376 : ide1
0378-037a : parport0
037b-037f : parport0
03c0-03df : vesafb
03f8-03ff : serial
0970-0977 : 0000:00:0b.0
  0970-0977 : sata_nv
09f0-09f7 : 0000:00:0b.0
  09f0-09f7 : sata_nv
0b70-0b73 : 0000:00:0b.0
  0b70-0b73 : sata_nv
0bf0-0bf3 : 0000:00:0b.0
  0bf0-0bf3 : sata_nv
0cf8-0cff : PCI conf1
d000-d00f : 0000:00:0b.0
  d000-d00f : sata_nv
d400-d47f : 0000:00:0b.0
  d400-d47f : sata_nv
d800-d8ff : 0000:00:06.0
  d800-d8ff : NVidia CK8
dc00-dc7f : 0000:00:06.0
  dc00-dc7f : NVidia CK8
e000-e007 : 0000:00:04.0
  e000-e007 : forcedeth
e400-e41f : 0000:00:01.1
f000-f00f : 0000:00:09.0
  f000-f007 : ide0
  f008-f00f : ide1

$ cat dma
 4: cascade
```

Или, еще проще, воспользуйтесь командой `lsdev`, которая собирает информацию из этих файлов и сортирует ее по периферийным устройствам, что, несомненно, более удобно.[2]:

```
lsdev
Device            DMA   IRQ  I/O Ports
```

2. `lsdev` входит в состав пакета **procinfo**.

```
------------------------------------------------
0000:00:01.1                e400-e41f
0000:00:04.0                e000-e007
0000:00:06.0                d800-d8ff dc00-dc7f
0000:00:09.0                f000-f00f
0000:00:0b.0                0970-0977 09f0-09f7 0b70-0b73 0bf0-0bf3 d000-d00f d400-d47f
cascade             4     2
CK8                       3
dma                         0080-008f
dma1                        0000-001f
dma2                        00c0-00df
ehci_hcd                 11
eth0                     10
forcedeth                     e000-e007
fpu                         00f0-00ff
i8042                    12
ide0                          f000-f007
ide1                     15 0170-0177 0376-0376   f008-f00f
keyboard                    0060-006f
NVidia                        d800-d8ff   dc00-dc7f
ohci_hcd                  9
parport0                    0378-037a 037b-037f
PCI                         0cf8-0cff
pic1                        0020-0021
pic2                        00a0-00a1
sata_nv                       0970-0977   09f0-09f7   0b70-0b73   0bf0-0bf3   d000-d00f   d400-d47f
serial                      03f8-03ff
timer                    0
timer0                      0040-0043
timer1                      0050-0053
vesafb                      03c0-03df
```

Вывод полного списка файлов был бы слишком долгим, но вот описание некоторых из них:

- `cpuinfo`: этот файл содержит, как видно из его имени, информацию о процессорах вашей машины.
- `modules`: этот файл содержит список модулей, используемых ядром в настоящий момент, вместе со счетчиком использования каждого из модулей. По сути эта информация используется командой `lsmod`, которая отображает ее в более удобной для чтения форме.
- `meminfo`: этот файл содержит информацию о загрузке памяти на момент вывода его содержимого. Команда `free` выведет ту же самую информацию, но уже в более удобном для чтения формате.
- `apm`: если у вас портативный компьютер, содержимое этого файла позволит вам увидеть состояние вашего аккумулятора. Вы сможете увидеть, подключен ли источник переменного тока, уровень зарядки аккумулятора и, если APM BIOS вашего ноутбука поддерживает это (к сожалению это не всегда так), оставшееся время "жизни" аккумулятора в минутах и т.п. Сам по себе файл не очень удобен для чтения, поэтому вы скорее всего захотите воспользоваться командой `apm`, которая выдаст ту же информацию в удобочитаемом для человека формате.

  Обратите внимание, что сейчас современные компьютеры предоставляют поддержку ACPI вместо APM. Смотрите ниже.
- `bus`: этот подкаталог содержит информацию обо всех периферийных устройствах, найденных на различных шинах вашего компьютера. Информация обычно не удобна для чтения, и большая ее часть переформатируется внешними утилитами: `lspcidrake`, `lspnp` и др.
- `acpi`: некоторые файлы, представленные в этом каталоге, особенно интересны для ноутбуков, которые позволяют вам выбирать различные варианты энергосбережения. Обратите внимание, что эти параметры проще изменять через приложения более высокого уровня, наподобие тех, что включены в пакет `acpid`.

  Вот наиболее интересные пункты:

  `battery`

  Показывает число аккумуляторов в ноутбуке и связанную с ними информацию, например, оставшийся на данный момент срок работы, максимальную емкость и т.п.

`button`

Позволяет вам управлять действиями, связанными со "специальными" кнопками: выключения питания, перехода в спящий режим, закрытия крышки и др.

`fan`

Показывает состояние вентиляторов вашего компьютера - работают ли они или нет, и позволяет вам запускать/останавливать их согласно определенным критериям. Возможность управления вентиляторами вашей машины зависит от материнской платы.

`processor`

Для каждого из CPU вашей машины имеется по одному подкаталогу. Опции управления варьируются в зависимости от типа процессора. Мобильные процессоры обладают большим числом опций, включая:

- возможность использования разных состояний питания, позволяя балансировать между производительностью и потребляемой мощностью;
- возможность изменения тактовой частоты для уменьшения потребляемой процессором мощности.

Обратите внимание, что существуют процессоры, которые не предоставляют таких возможностей.

`thermal_zone`

Информация о рабочей температуре вашей системы/процессора.

10.3. Подкаталог `/proc/sys`

Назначение этого каталога - сообщать о различных параметрах ядра, и позволить вам изменять некоторые из них в интерактивном режиме. В противоположность всем другим файлам каталога `/proc`, некоторые файлы из этого каталога могут быть открыты для записи, но только для `root`'а.

Вывод списка каталогов и файлов для описания был бы слишком долгим в основном из-за того, что содержимое этих каталогов зависит от системы, а большинство файлов будет полезно только для очень специализированных приложений. Тем не менее, вот два общих случая использования этого подкаталога:

1. Разрешение маршрутизации: даже если ядро от Mandrakelinux по умолчанию в состоянии выполнять маршрутизацию, вы должны явно разрешить ему делать это. Для этого вам под `root`'ом необходимо набрать следующую команду:

   ```
   $ echo 1 >/proc/sys/net/ipv4/ip_forward
   ```

   Замените 1 на 0, если вы хотите запретить маршрутизацию.

2. Предотвращение IP-спуфинга: IP-спуфинг заключается в том, чтобы путем подмены IP-адреса заставить сетевой интерфейс поверить в то, что пакет, пришедший из внешнего мира, пришел от него самого. Эта техника очень часто используется ***кракерами*** [3]. Вы можете сделать так, что ядро будет препятствовать такому виду вторжения. Введите:

   ```
   $ echo 1 >/proc/sys/net/ipv4/conf/all/rp_filter
   ```

   и все типы таких атак станут невозможными.

Эти изменения останутся в действии до тех пор, пока работает система. После ее перезагрузки эти значения вернутся в свои значения по умолчанию. Чтобы при загрузке они устанавливались во что-либо, отличное от значений по умолчанию, вы можете взять команды, которые вы вводили в командной строке, и добавить их в `/etc/rc.d/rc.local`. Таким образом отпадет необходимость каждый раз набирать их заново. Другим решением является изменение файла `/etc/sysctl.conf`, смотрите sysctl.conf(5).

3. Не путать с ***хакерами***!

Глава 11. Загрузочные файлы: init sysv

Схема загрузки системы System V унаследована от UNIX® **AT&T** и является одной из традиционных схем загрузки UNIX®. Она отвечает за запуск и остановку служб для приведения операционной системы в одно из стандартных состояний. Службы варьируются от базовой аутентификации пользователя до локального графического сервера или служб Интернета.

Утилита Mandrakelinux, которая позволяет вам вручную запускать или останавливать службы - это `drakxservices`. Она доступна в Центре управления Mandrakelinux в разделе "Система", значок Службы.

11.1. В начале был init

При запуске системы, когда ядро все настроило и примонтировало корневую файловую систему, она выполняет команду `/sbin/init`[1]. `init` является родителем для всех системных процессов и отвечает за перевод системы на необходимый ***уровень выполнения***. Мы рассмотрим эти уровни позже (см. Разд. 11.2).

Конфигурационный файл `init`'а называется `/etc/inittab` и для него есть отдельная страница руководства (inittab(5)), поэтому мы опишем только некоторые из возможных конфигурационных параметров.

Первая строка, на которую вы должны обратить внимание, это:

```
si::sysinit:/etc/rc.d/rc.sysinit
```

Эта строка сообщает `init`'у, что `/etc/rc.sysinit` будет выполнен сразу после инициализации системы (`si` означает *System Init*). Затем, чтобы определить уровень выполнения по умолчанию, `init` будет искать строку, содержащую ключевое слово `initdefault`:

```
id:5:initdefault:
```

В этом случае `init` знает, что уровнем выполнения по умолчанию является 5. Он также знает, что для перехода на уровень 5, он должен выполнить следующую команду:

```
l5:5:wait:/etc/rc.d/rc 5
```

Как видите, синтаксис для каждого уровня выполнения одинаков.

`init` также отвечает за перезапуск (`respawn`) некоторых программ, которые не могут быть запущены другим процессом. Например, каждая из программ login, которые работают в 6-ти виртуальных консолях, запущена `init`'ом[2]. Вторая виртуальная консоль определяется следующим образом:

```
2:2345:respawn:/sbin/mingetty tty2
```

11.2. Уровни выполнения

Все файлы, связанные с запуском системы, находятся в каталоге `/etc/rc.d`. Вот список его файлов:

```
$ ls /etc/rc.d
init.d/  rc0.d/  rc2.d/  rc4.d/  rc6.d/            rc.local*    rc.sysinit*
rc*      rc1.d/  rc3.d/  rc5.d/  rc.alsa_default*  rc.modules*
```

Как уже было отмечено, `rc.sysinit` - это первый файл, запускаемый системой. Он отвечает за установку базовой конфигурации машины: тип клавиатуры, настройка определенных устройств, проверка файловой системы и т.п.

---

1. Вот почему размещение `/sbin` не в корневой файловой системе было бы очень плохой идеей. На этом этапе ядро еще не примонтировало ни одного другого раздела и поэтому не смогло бы найти `/sbin/init`.
2. Если вам не нужны 6 виртуальных консолей, вы можете добавить или удалить их, изменив этот файл. Если вы хотите увеличить число консолей, то можете создать их до 64. Но не забудьте, что X тоже выполняется в виртуальной консоли, поэтому оставьте для него по крайней мере одну свободную консоль.

Затем запускается скрипт `rc` с нужным номером уровня выполнения в качестве аргумента. Как мы уже видели, уровень выполнения - это простое целое число, а для всех определенных уровней выполнения `<x>` должен быть соответствующий каталог `rc<x>.d`. В стандартной установке Mandrakelinux вы могли увидеть, что существует шесть уровней выполнения:

- 0: полная остановка машины;
- 1: ***однопользовательский*** режим. Используется в случае возникновения серьезных проблем или для восстановления системы.
- 2: ***многопользовательский*** режим без поддержки сети.
- 3: многопользовательский режим с поддержкой сети.
- 4: неиспользуемый.
- 5: аналогичен 3-му уровню выполнения, но запускается графический интерфейс для входа в систему.
- 6: перезагрузка.

Давайте взглянем на содержимое каталога `rc5.d`:

```
$ls rc5.d
K59dund@        S12syslog@    S20xfs@          S34mDNSResponder@  S90crond@
K59hidd@        S13partmon@   S24messagebus@   S40atd@            S95kheader@
K59pand@        S14hplip@     S25bluetooth@    S56ntpd@           S99local@
S01udev@        S15cups@      S25haldaemon@    S56rawdevices@
S05harddrake@   S15mdadm@     S25netfs@        S75keytable@
S10network@     S17alsa@      S30dm@           S80postfix@
S12pcscd@       S18sound@     S33nifd@         S85numlock@
```

Как видите, все файлы в этом каталоге являются символическими ссылками, и все они имеют весьма специфический вид. Их общий вид такой:

```
<S|K><порядок><имя_службы>
```

`S` означает запуск (*Start*) службы, а `K` означает остановку (*Kill*) службы. Скрипты запускаются в порядке возрастания номеров, а если два скрипта имеют одинаковый номер, тогда применяется прямой алфавитный порядок. Мы также можем видеть, что все символические ссылки указывают на определенные скрипты, находящиеся в каталоге `/etc/rc.d/init.d` (за исключением скрипта `local`, отвечающего за управление особой службой.)

Когда система переходит на заданный уровень выполнения, она начинает с того, что запускает по порядку ссылки `K`: команда `rc` ищет, куда указывает ссылка, затем вызывает соответствующий скрипт с одним аргументом `stop`. Затем она запускает скрипты `S`, используя такой же метод, за исключением тех скриптов, которые вызваны с аргументом `start`.

Итак, не рассматривая всех скриптов, мы можем увидеть, что когда система переходит на 5-й уровень выполнения, она сначала запускает команду `K15dund` (т.е. `/etc/rc.d/init.d/dund stop`). Затем `K59hidd`, затем `K59pand`, и так далее до последней команды. Далее она запускает все скрипты `S`: сначала `S01udev`, который в свою очередь вызывает `/etc/rc.d/init.d/udev start`, и так далее.

Вооружившись этой информацией, вы за несколько минут сможете создать свой собственный уровень выполнения (задействовав, например, 4-й), или предотвратить запуск или остановку службы, удалив соответствующую символическую ссылку. Для этого вы также можете воспользоваться программами с интерфейсом, в особенности drakxservices (см. *DrakXServices: Настройка загрузочных сервисов* в книге *Стартовое руководство*) или chkconfig для настройки в текстовом режиме.

Вы также можете воспользоваться командой `chkconfig` для вывода списка, добавления или удаления служб из определенного уровня выполнения. Смотрите chkconfig(8).

Глава 12. Сборка и установка свободного программного обеспечения

Нас часто спрашивают, как установить свободное программное обеспечение из исходных текстов. Сама по себе компиляция программ действительно проста, т.к. большинство выполняемых этапов является одинаковым вне зависимости от устанавливаемого программного обеспечения. Цель этого документа - предоставить новичку пошаговое руководство и объяснить значение каждого действия. Мы подразумеваем, что читатель обладает минимальным объемом знаний о системе UNIX® (например, `ls` или `mkdir`).

Этот документ всего лишь руководство, а не справочник. Вот почему в конце приведено несколько ссылок, которые помогут вам получить ответы на все оставшиеся вопросы. Эта глава наверняка может быть улучшена, поэтому мы будем признательны за любые замечания или пожелания по ее содержанию.

12.1. Введение

Разница между свободным и коммерческим (проприетарным) программным обеспечением заключается в доступности его исходного кода. Это означает, что свободное ПО поставляется в виде архивов с файлами исходного кода. Это может быть непривычным для новичков, потому как пользователи свободного ПО должны сами откомпилировать исходный код прежде, чем они смогут пользоваться программой.

Большинство свободного программного обеспечения существует в откомпилированном виде. Поэтому устанавливать эти предварительно откомпилированные бинарные файлы следует только, если вы очень спешите. В таком виде не распространяется некоторое свободное ПО или его более ранние версии. Кроме того, если вы используете экзотическую операционную систему или экзотическую архитектуру, много откомпилированных для вас программ будет отсутствовать. Более того, самостоятельная компиляция ПО позволит вам включить только нужные опции или расширить функциональность программы, чтобы максимально удовлетворить ваши потребности.

12.1.1. Требования

Для сборки ПО вам понадобится:

- компьютер с работающей операционной системой,
- базовые знания по используемой вами операционной системе,
- некоторый объем свободного дискового пространства,
- компилятор (обычно для языка программирования C) и архиватор (`tar`),
- некоторое количество еды (в тяжелых случаях процесс может занять много времени). Настоящие хакеры едят пиццу, а не quiche (французский пирог с заварным кремом и различной начинкой).
- запить что-нибудь (по тем же соображениям). Настоящие хакеры пьют газировку с кофеином.
- телефонный номер вашего друга, который перекомпилирует свое ядро каждую неделю,
- терпение и еще раз терпение!

Компиляция из исходных кодов как правило не вызывает много проблем, но если вы не изучали этого ранее, малейшая загвоздка может отпугнуть вас.

12.1.2. Компиляция

12.1.2.1. Принципы

Чтобы преобразовать исходный код в бинарный файл, должна быть выполнена ***компиляция*** (обычно из исходных кодов на C или C++, являющихся наиболее распространенными языками из тех, что используются в сообществе (UNIX®) свободного ПО). Некоторые свободные программы написаны на

языках, не требующих компиляции (например, на perl или shell), но для них все-таки требуется некоторая настройка.

Компиляция кода на C по логике вещей выполняется при помощи компилятора C, обычно `gcc` - свободного компилятор, написанного проектом GNU (`http://www.gnu.org/`). Компилирование всего программного пакета -сложная задача, выполняемая путем успешных компиляций различных исходных файлов (по многим причинам программисту проще поместить разные части своей работы в отдельные файлы). Чтобы упростить это для вас этими повторяющимися операциями руководит утилита под названием `make`.

12.1.2.2. Четыре этапа компиляции

Чтобы понять, как происходит компиляция (для решения возможных проблем), вы должны знать о ее четырех этапах. Задание заключается в постепенном преобразовании текстового файла, написанного на понятном для квалифицированного человека языке (т.е.на языке C), в язык, понятный для машины (или, в некоторых случаях, для **чрезвычайно** квалифицированного человека). `gcc` одну за другой выполняет четыре программы, каждая из которых делает один этап:

1. `cpp`: Первый этап заключается в замене директив (***препроцессоры***) чистыми инструкциями C. Обычно это означает вставку заголовка (`#include`) или определение макрокоманд (`#define`). В конце этого этапа генерируется чистый код C.
2. `cc1`: Этот этап заключается в преобразовании кода C в код ***языка ассемблера***. Сгенерированный код зависит от архитектуры, для которой он предназначен.
3. `as`: Этот этап заключается в генерации ***объектного кода*** (или ***бинарного*** кода) из кода на языке ассемблера. В конце этого этапа генерируется файл с расширением `.o`.
4. `ld`: На последнем этапе (***компоновке***) устанавливаются связи между всеми объектными файлами (`.o`) и их библиотеками и результате получается исполняемый файл.

12.1.3. Структура дистрибутива

Правильно организованный дистрибутив свободного ПО всегда имеет одинаковую структуру:

- Файл `INSTALL`, описывающий процедуру установки.
- Файл `README`, содержащий общую информацию, имеющую отношение к программе (короткое описание; имя автора; URL, с которого ее можно загрузить; связанную с ней документацию; полезные ссылки и т.п.). Если файл `INSTALL` отсутствует, то обычно в файле `README` находится краткая инструкция по установке.
- Файл `COPYING`, содержащий лицензию или описывающий условия распространения ПО. Иногда вместо него для этих целей используется файл `LICENSE` с таким же содержимым.
- Файл `CONTRIB` или файл `CREDITS`, содержащий список людей, имеющих отношение к этому программному продукту (активное участие, дельные комментарии и т.д.).
- Файл `CHANGES` (реже `NEWS`), содержащий последние изменения и исправления ошибок.
- Файл `Makefile` (см. раздел Разд. 12.4.1), управляющий компиляцией программы (нужен для `make`). Если этот файл изначально не существует, то он создается процессом конфигурирования перед компиляцией.
- Довольно часто файл `configure` или `Imakefile`, позволяющий сгенерировать новый файл `Makefile`, настроенный для конкретной системы (см. Разд. 12.3).
- Каталог, содержащий исходные коды, и в котором обычно сохраняется бинарный файл после завершения компиляции. Обычно он называется `src`.
- Каталог, содержащий документацию к программе (обычно в формате `man` или Texinfo), с именем `doc`.
- Иногда каталог, содержащий специфические для программы данные (обычно файлы настройки, примеры получаемых данных или файлы ресурсов).

12.2. Распаковка

12.2.1. Архив tar.gz

Стандартным[1] форматом сжатия в UNIX®-системах является формат `gzip`, разработанный проектом GNU и считающийся одним из самых лучших среди общих утилит сжатия.

`gzip` часто ассоциируется с утилитой с именем `tar`. `tar` выжил с тех незапамятных времен, когда компьютерщики хранили свои данные на магнитных лентах. Сейчас на замену магнитным лентам пришли CD-ROM и DVD, а дискеты постепенно вытесняются flash-накопителями. Однако для создания архивов все еще используется `tar`. Например, все файлы каталога могут быть объединены в один файл. Затем этот файл может быть легко сжат при помощи `gzip`.

Вот почему свободное ПО обычно распространяется в виде `tar`-архивов, сжатых `gzip`'ом. Поэтому они имеют расширение `.tar.gz` (или `.tgz` для краткости).

12.2.2. Использование GNU Tar

Для распаковки этого архива можно воспользоваться `gzip`'ом, а затем `tar`'ом. Но GNU-версия `tar` (`gtar`) позволяет нам ***"на лету"*** использовать `gzip` и распаковать файл архива без уведомления о выполнении каждого шага (и без использования дополнительного дискового пространства).

Формат использования `tar`:

```
tar <опции_файла> <файл.tar.gz> [файлы]
```

Опция <файлы> является необязательной. Если она опущена, будет обработан весь архив. Этот аргумент не нужно указывать для извлечения всего содержимого архива `.tar.gz`

Например:

```
$ tar xvfz guile-1.3.tar.gz
-rw-r--r-- 442/1002      10555 1998-10-20 07:31 guile-1.3/Makefile.in
-rw-rw-rw- 442/1002       6668 1998-10-20 06:59 guile-1.3/README
-rw-rw-rw- 442/1002       2283 1998-02-01 22:05 guile-1.3/AUTHORS
-rw-rw-rw- 442/1002      17989 1997-05-27 00:36 guile-1.3/COPYING
-rw-rw-rw- 442/1002      28545 1998-10-20 07:05 guile-1.3/ChangeLog
-rw-rw-rw- 442/1002       9364 1997-10-25 08:34 guile-1.3/INSTALL
-rw-rw-rw- 442/1002       1223 1998-10-20 06:34 guile-1.3/Makefile.am
-rw-rw-rw- 442/1002      98432 1998-10-20 07:30 guile-1.3/NEWS
-rw-rw-rw- 442/1002       1388 1998-10-20 06:19 guile-1.3/THANKS
-rw-rw-rw- 442/1002       1151 1998-08-16 21:45 guile-1.3/TODO
...
```

Некоторые из опций `tar`:

- `v` делает вывод `tar` подробным. Это означает, что на экран будут выведены все найденные в архиве файлы. Если эта опция опущена, информация, выводимая в процессе обработки, будет ограничена.
- `f` является обязательной опцией. Без нее `tar` пытается использовать магнитную ленту вместо файла архива (т.е. устройство `/dev/rmt0`, стример).
- `z` позволяет вам обрабатывать архив, сжатый gzip'ом (с расширением `.gz`). Если вы забудете указать эту опцию, `tar` выдаст ошибку. И наоборот, эта опция не должна использоваться для несжатых архивов.

`tar` позволяет вам выполнять над архивом различные действия (извлекать, читать, создавать, добавлять...). Опции определяют тип используемого действия:

- `x`: позволяет вам извлекать файлы из архива.
- `t`: выводит список содержимого архива.
- `c`: позволяет вам создать архив. Вы можете использовать ее для создания резервных копий своих личных файлов, например.

1. На сегодняшний день в GNU/Linux стандартом является формат `bzip2`. `bzip2` более эффективен для текстовых файлов за счет использования большей вычислительной мощности. Смотрите раздел Разд. 12.2.3, посвященный специально этой программе.

- r: позволяет вам добавлять файлы в конец архива. Она не может быть использована для уже сжатых архивов.

12.2.3. Bzip2

Формат сжатия bzip2 уже заменил собой gzip в общем использовании, однако некоторые программы все еще распространяются в формате gzip в основном для совместимости с более старыми системами. Почти все свободное ПО сейчас распространяется в архивах с расширением .tar.bz2.

По отношению к команде tar bzip2 используется также, как и gzip. Единственным отличием является использование буквы j вместо z. Например:

```
$ tar xvjf foo.tar.bz2
```

Другой способ (который хоть и выглядит более понятным, но дольше набирается!):

```
$ tar --use-compress-program=bzip2 -xvf foo.tar.bz2
```

Перед запуском tar убедитесь, что bzip2 установлен в каталог, включенный в вашу переменную окружения PATH.

12.2.4. Просто сделайте это!

12.2.4.1. Самый простой способ

Теперь, когда вы готовы распаковать архив, не забудьте сделать это с правами администратора (root). Вам понадобится делать вещи, которые не разрешается делать обычному пользователю. И даже если вы можете сделать некоторые из них как обычный пользователь, лучше сразу стать root'ом, даже если это может быть не совсем безопасным[2].

Первым делом перейдите в каталог /usr/local/src и скопируйте в него архив. Впоследствии вы всегда сможете найти архив в случае утраты программного обеспечения. Если у вас не так много свободного дискового пространства, сохраните архив на дискету после установки программы. (Однако вряд ли вам это удастся для любых исходников, например, для KDE :) Видимо это руководство писалось в те времена, когда терабайтовые массивы IDE/SATA/SCSI-дисков были еще чем-то из области фантастики (прим. переводчика)) Вы также можете удалить его, но убедитесь, что в любой момент сможете найти его во Всемирной паутине. Также хорошей практикой является сохранение командной строки, использованной для настройки программного обеспечения, или всей последовательности команд, использованной для создания рабочей копии. В этом случае вам также может потребоваться повторить эти действия при обновлении.

Обычно при распаковке tar-архива должен быть создан новый каталог (вы можете заранее проверить это, благодаря опции t). Перейдите затем в этот каталог. Теперь вы готовы приступить к следующему этапу.

12.2.4.2. Самый безопасный путь

UNIX®-системы (примерами которых являются GNU/Linux и FreeBSD®) могут быть безопасными системами. Это означает, что обычные пользователи не смогут осуществить действия, способные подвергнуть систему опасности (отформатировать диск, например), или изменить файлы других пользователей. Это также обеспечивает систему неким иммунитетом против вирусов.

С другой стороны, root может сделать все - даже запустить вредоносную программу. Наличие доступных исходных текстов позволяет вам изучить их на наличие зловредного кода (вирусов и троянов). В этом отношении лучше быть осторожным[3].

Идея состоит в создании пользователя специально для администрирования (например **free** или **admin**) при помощи команды adduser. Этому пользователю должно быть предоставлено разрешение на запись

2. Обычно вам потребуется стать root'ом только для установки в систему сгенерированных файлов. Все остальные этапы (для "правильных" приложений) должны иметь возможность быть выполненными обычными пользователями.
3. Пословица из мира BSD гласит: "Никогда не доверяй пакету, для которого у тебя нет исходных кодов."

в следующие каталоги: `/usr/local/src`, `/usr/local/bin` и `/usr/local/lib`, а также во все подкаталоги `/usr/share/man` (ему также может понадобиться копировать файлы в какое-нибудь другое место). Мы рекомендуем, чтобы вы сделали этого пользователя владельцем необходимых каталогов или создали для него группу и открыли каталоги на запись для этой группы.

Как только приняты все эти предосторожности, вы можете переходить к инструкциям из раздела Разд. 12.2.4.1, заменив `root`'а новым созданным пользователем.

12.3. Конфигурирование

Факт создания авторами исходных кодов представляет собой чисто технический интерес для ***портирования*** программ. Свободное программное обеспечение, разработанное для UNIX®-систем, может быть использовано во всех существующих системах UNIX® (как свободных, так и собственнических) с незначительными изменениями или вообще без изменений. Для этого требуется сконфигурировать ПО непосредственно перед его компиляцией.

Существует несколько систем конфигурирования. Вы должны использовать ту, которую требует автор программного обеспечения (а иногда и несколько). Обычно вы можете:

- Использовать AutoConf (см. раздел Разд. 12.3.1), если в родительском каталоге дистрибутива имеется файл с именем `configure`.
- Использовать imake (см. раздел Разд. 12.3.2), если в родительском каталоге дистрибутива имеется файл с именем `Imakefile`.
- Запустить shell-***скрипт*** (например, `install.sh`) согласно содержимому файла `INSTALL` (или файла `README`).

12.3.1. Autoconf

12.3.1.1. Принципы

Программа AutoConf используется для корректной настройки ПО. Она создает необходимые для компиляции файлы (например, `Makefile`) и иногда изменяет непосредственно сами исходные тексты (например, при помощи файла `config.h.in`).

Принцип AutoConf прост:

- Разработчик ПО знает, какие проверки необходимы для настройки его программы (например: "какую версию этой ***библиотеки*** вы используете?"). Он записывает их в файл с именем `configure.in`, используя определенный синтаксис.
- Он запускает AutoConf, которая создает из файла `configure.in` конфигурационный скрипт с именем `configure`. Этот скрипт выполняет тесты, необходимые при настройке программы.
- Конечный пользователь запускает скрипт и AutoConf настраивает все, что необходимо для компиляции.

12.3.1.2. Пример

Пример использования AutoConf:

```
$ ./configure
loading cache ./config.cache
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for main in -lX11... yes
checking for main in -lXpm... yes
checking for main in -lguile... yes
```

```
checking for main in -lm... yes
checking for main in -lncurses... yes
checking how to run the C preprocessor... gcc -E
checking for X... libraries /usr/X11R6/lib, headers /usr/X11R6/include
checking for ANSI C header files... yes
checking for unistd.h... yes
checking for working const... yes
updating cache ./config.cache
creating ./config.status
creating lib/Makefile
creating src/Makefile
creating Makefile
```

Для лучшего управления информацией, генерируемой `configure`, из командной строки или через переменные окружения могут быть добавлены некоторые опции. Пример:

```
$ ./configure --with-gcc --prefix=/opt/GNU
```

или (при помощи bash):

```
$ export CC=`which gcc`
$ export CFLAGS=-O2
$ ./configure --with-gcc
```

или:

```
$ CC=gcc CFLAGS=-O2 ./configure
```

12.3.1.3. А что если... это не работает?

Обычно это ошибка подобного вида: `configure: error: Cannot find library guile` (подобным образом выглядит большинство ошибок скрипта `configure`).

Это означает, что скрипт `configure` не смог найти библиотеку (в этой примере библиотеку `guile`). Принцип заключается в том, что скрипт `configure` компилирует небольшую тестовую программу, использующую эту библиотеку. Если компиляция этой программы завершится неудачей, то невозможно будет откомпилировать и весь программный пакет. Затем возникает ошибка.

- Причину ошибки ищите в конце файла `config.log`, содержащего отчет обо всех этапах конфигурирования. Компилятор C выводит довольно четкие сообщения об ошибках. Обычно это поможет вам при решении возникающих проблем.
- Проверьте, правильно ли установлена названная библиотека. Если это не так - установите ее (из исходных кодов или в виде откомпилированного бинарного файла) и еще раз запустите `configure`. Эффективным способом проверки является поиск файла, содержащего символы библиотеки, коим всегда будет `lib<имя>.so`. Например,

  ```
  $ find / -name 'libguile*'
  ```

  или, как вариант:

  ```
  $ locate libguile
  ```

- Проверьте, доступна ли библиотека для компилятора. Это означает, что она находится в одном из каталогов: `/usr/lib`, `/lib`, `/usr/X11R6/lib` (или в одном из тех, что определены переменной окружения `LD_LIBRARY_PATH`, как описано в Разд. 12.4.5 пункт 5.b). Проверьте, является ли этот файл библиотекой, набрав `file libguile.so`.
- Проверьте, правильно ли установлены заголовочные файлы библиотеки (обычно в `/usr/include`, `/usr/local/include` или `/usr/X11R6/include`). Если вы не знаете, какие файлы заголовков вам нужны, проверьте, установлена ли у вас development-версия (для разработки) требуемой библиотеки (например, `libgtk+2.0-devel` вместо `libgtk+2.0`). Версия библиотеки для разработки предоставляет файлы "include", необходимые для компиляции ПО, использующего эту библиотеку.
- Проверьте, достаточно ли у вас свободного дискового пространства (для скрипта `configure` требуется некоторый объем для временных файлов). Воспользуйтесь командой `df -h` для вывода списка разделов вашей системы и обратите внимание на заполненные или почти заполненные разделы.

Если вы не понимаете сообщения, сохраненные в файле `config.log`, не стесняйтесь попросить помощи у сообщества свободного ПО (см. раздел Разд. 12.6.2).

Кроме того, проверьте, существует ли библиотека, даже если `configure` говорит, что ее нет (например, было бы очень странно, если бы в вашей системе отсутствовала библиотека `curses`). В этом случае, вероятно, повреждена переменная окружения `LD_LIBRARY_PATH`!

12.3.2. Imake

imake позволяет вам конфигурировать свободное ПО путем создания по простым правилам файла `Makefile`. Эти правила определяют, какие файлы должны быть откомпилированы для сборки бинарного файла, а imake генерирует соответствующий `Makefile`. Эти правила находятся в файле с именем `Imakefile`.

Интересная вещь, касающаяся imake, заключается в том, что последний использует ***зависимую от местонахождения*** (или зависимую от архитектуры) информацию. Это довольно удобно для приложений, использующих X Window System. Но imake используется также и для многих других приложений.

Простейшее использование imake заключается в переходе в главный каталог распакованного архива и последующем запуске скрипта `xmkmf`, который вызывает программу imake:

```
$ xmkmf -a
imake -DUseInstalled -I/usr/X11R6/lib/X11/config
$ make Makefiles
```

Если сайт установлен неправильно, перекомпилируйте и установите X11R6!

12.3.3. Разные shell-скрипты

Для получения дополнительной информации прочтите файлы `INSTALL` или `README`. Обычно вам нужно запустить файл `install.sh` или `configure.sh`. Затем инсталляционный скрипт будет либо бездиалоговым (и сам определит все, что ему нужно), либо будет запрашивать у вас информацию о вашей системе (пути, например).

Если вы не можете определить, какой файл нужно запустить, вы можете ввести `./` (в bash'е), а затем дважды нажать **TAB**. bash автоматически (в конфигурации по умолчанию) дополнит команду возможным исполняемым файлом из каталога (а, следовательно, возможным конфигурационным скриптом). Если выполнены могут быть несколько файлов, вам будет выведен их список. Затем вам нужно просто выбрать правильный файл.

Другим особым случаем является установка модулей perl. Установка таких модулей выполняется путем запуска конфигурационного скрипта, написанного на perl. Обычно выполняется команда:

```
$ perl Makefile.PL
```

12.3.4. Альтернативы

Некоторые дистрибутивы свободного ПО имеют плохо организованную структуру, особенно на начальных этапах разработки (но пользователя об этом предупреждают!). Для них иногда требуется, чтобы вы "вручную" изменили некоторые конфигурационные файлы. Обычно это файлы `Makefile` (см. раздел Разд. 12.4.1) и `config.h` (это просто условное имя).

Мы не советуем делать эти манипуляции, за исключением тех пользователей, которые действительно знают, что они делают. Для этого требуются реальные знания и немного стремления к успеху, однако практика - путь к совершенству.

12.4. Компиляция

Теперь, когда программное обеспечение корректно сконфигурировано, все, что осталось сделать - это откомпилировать его. Этот этап обычно прост и не вызывает серьезных проблем.

12.4.1. Make

В обществе свободного ПО любимой утилитой для компиляции исходных кодов является `make`. Она имеет два преимущества:

- Разработчик экономит время, потому что она позволяет ему эффективно управлять компиляцией своего проекта.
- Конечный пользователь может откомпилировать и установить программу при помощи нескольких команд, даже если у него отсутствуют знания о разработке программного обеспечения.

Действия, которые должны быть выполнены для получения из исходных кодов откомпилированной версии, хранятся в файле с именем `Makefile` или `GNUMakefile`. На самом деле, когда вызывается команда `make`, она считывает этот файл (если он существует) из текущего каталога. В противном случае файл может быть указан при помощи опции `-f` команды `make`.

12.4.2. Правила

`make` оперирует в соответствии с системой ***зависимостей***, поэтому компиляция бинарного файла (“***цели***”) требует прохождения нескольких этапов (“зависимостей”). Например, для создания (воображаемого) бинарного файла `glloq` должны быть откомпилированы и скомпонованы объектные файлы `main.o` и `init.o` (промежуточные файлы процесса компиляции). Эти объектные файлы также являются целями, чьими зависимостями являются соответствующие файлы исходных текстов.

Этот текст представляет собой только небольшое введение для выживания в жестоком мире `make`. Для получения исчерпывающей информации обратитесь к **O’Reilly Managing Projects with Make** (второе издание) авторов Andrew Oram и Steve Talbott.

12.4.3. Поехали!

Обычно при использовании `make` принято придерживаться некоторых соглашений. Например:

- `make` без аргумента просто компилирует программу, не устанавливая ее.
- `make install` компилирует программу (но не всегда), а затем устанавливает необходимые файлы в нужное место в файловой системе. Некоторые файлы не всегда устанавливаются корректно (`man`, `info`), пользователю может понадобиться скопировать их самому. Иногда команда `make install` должна быть выполнена повторно в подкаталогах. Обычно это касается модулей сторонних разработчиков.
- `make clean` удаляет все временные файлы, созданные в процессе компиляции, а также, в большинстве случаев, и исполняемые файлы.

Первым этапом является компиляция программы, а, следовательно, ввод команды (выдуманный пример):

```
$ make
gcc -c glloq.c -o glloq.o
gcc -c init.c -o init.o
gcc -c main.c -o main.o
gcc -lgtk -lgdk -lglib -lXext -lX11 -lm glloq.o init.o main.o -o glloq
```

Превосходно. Бинарный файл был корректно скомпилирован. Мы готовы перейти к следующему этапу, который представляет собой установку файлов дистрибутива (бинарные файлы, файлы данных и т.п.). Смотрите раздел Разд. 12.5.

12.4.4. Пояснения

Если вы достаточно любопытны, чтобы заглянуть в файл `Makefile`, вы найдете в нем известные команды (`rm`, `mv`, `cp` и др.), а также странные строки наподобие `$(CFLAGS)`.

Это ***переменные***, представляющие собой строки, которые обычно объявляются в начале файла `Makefile`, а затем заменяются их значениями. Это весьма полезно, если вы хотите использовать одни и те же опции компиляции несколько раз подряд.

Например, вывести на экран строку "`foo`" при помощи команды `make all` можно так:

```
TEST = foo
all:
        echo $(TEST)
```

В большинстве случаев установлены следующие переменные:

1. `CC`: это компилятор. Обычно это `cc`, синонимом которого в большинстве свободных систем является `gcc`. Если вы в нерешительности - используйте `gcc`.
2. `LD`: это программа, используемая для обеспечения последнего этапа компиляции (см. раздел Разд. 12.1.2.2). По умолчанию это программа `ld`.
3. `CFLAGS`: это дополнительные аргументы, передаваемые компилятору на первом этапе компиляции. Среди них:
   - `-I<путь>`: сообщает компилятору, где искать дополнительные заголовочные файлы (напр.: `-I/usr/X11R6/incl` разрешает добавление заголовков из каталога `/usr/X11R6/include`).
   - `-D<символ>`: определяет дополнительный символ; полезен для программ, компиляция которых зависит от определенных ранее символов (напр.: использование файла `string.h`, если определен `HAVE_STRING_H`).

   Часто строки компиляции выглядят следующим образом:

   ```
   $(CC) $(CFLAGS) -c foo.c -o foo.o
   ```

4. `LDFLAGS` (или `LFLAGS`): это аргументы, используемые на последнем этапе компиляции. Среди них:
   - `-L<путь>`: определяет дополнительный путь для поиска библиотек (напр.: `-L/usr/X11R6/lib`).
   - `-l<библиотека>`: определяет дополнительную библиотеку, используемую на последнем этапе компиляции.

12.4.5. А что если... это не работает?

Не паникуйте, это может случится с любым. Вот наиболее общие случаи:

1. `glloq.c:16: decl.h: No such file or directory`

   Компилятор не смог найти соответствующий заголовочный файл. Вообще-то эта ошибка должна была быть предупреждена на этапе конфигурирования программы. Решение этой проблемы:

   - Проверьте, действительно ли заголовок существует на диске в одном из следующих каталогов: `/usr/include`, `/usr/local/include`, `/usr/X11R6/include` или в одном из их подкаталогов. Если его там нет, сделайте поиск по всему диску (при помощи `find` или `locate`), и, если вы все еще не нашли его, проверьте, установлена ли у вас библиотека, соответствующая этому заголовку. Примеры использования команд `find` и `locate` вы можете найти в соответствующих им страницах руководства.
   - Проверьте, действительно ли этот заголовок доступен для чтения (чтобы проверить это, наберите `less <path>/<file>.h`)
   - Если это каталог типа `/usr/local/include` или `/usr/X11R6/include`, вам иногда придется добавить компилятору новый аргумент. Откройте соответствующий `Makefile` (будьте внимательны при открытии этого файла, т.к. он должен находиться в каталоге, где произошел сбой компиляции

[4]) в своем любимом текстовом редакторе (Emacs, Vi и т.п.). Взгляните на строку, вызвавшую ошибку, и добавьте строку `-I<путь>` (`<путь>` - это путь к каталогу, в котором может быть найден заголовочный файл) сразу после компилятора (`gcc` или `$(CC)`). Если вы не знаете, куда добавить эту опцию, добавьте ее в начало файла после `CFLAGS=<чего-то-там>` или после `CC=<чего-то-там>`.

- Запустите еще раз `make`, и если это все равно помогает, проверьте, чтобы эта опция (см. предыдущий пункт) во время компиляции была добавлена в строку со сбоем.
- Если это все равно не помогло, обратитесь с просьбой помочь решить проблему к местному гуру или к сообществу свободного ПО (см. раздел Разд. 12.6.2).

2. `glloq.c:28: 'struct foo' undeclared (first use this function)`

   Структуры - это специальные типы данных, используемые всеми программами. Многие из них определяются системой в заголовочных файлах. Это означает, что проблема, несомненно, была вызвана отсутствием или неправильным использованием заголовочного файла. Правильный способ решения проблемы:

   - Попробуйте проверить, определена ли структура программой или системой. Решением является использование команды grep для того, чтобы увидеть, определена ли структура в одном из заголовков.

     Например, если вы находитесь в корне дистрибутива:

     ```
     $ find . -name '*.h'| xargs grep 'struct foo' | less
     ```

     На экран может быть выведено множество строк (например, каждый раз, когда функция использует определенную структуру этого типа). Если она присутствует, извлеките строку, в которой определена структура, "grep'нув" заголовочный файл.

     Определение структуры:

     ```
     struct foo {
         <содержимое структуры>
     };
     ```

     Проверьте, соответствует ли она той, что имеется у вас. Если да, то это значит, что заголовочный файл не включен в сбойный файл `.c`. Есть два решения:

     - добавьте строку `#include "<имя_файла>.h"` в начало сбойного файла `.c`.
     - или скопируйте определение структуры в начало этого файла (на самом деле это не совсем то, что надо, но, по крайне мере, зачастую это срабатывает).

   - Если нет, проделайте то же самое с системными заголовочными файлами (которые обычно находятся в каталогах `/usr/include`,`/usr/X11R6/include` или `/usr/local/include`). Но на этот раз используйте строку `#include <<имя_файла>.h>`.
   - Если эта структура все-таки не существует, попробуйте выяснить, в какой из библиотек (т.е. наборе функций, собранных вместе в одном пакете) она должна была быть определена (взгляните на файл `INSTALL` или `README`, чтобы узнать, какие библиотеки используются программой и какие их версии требуются). Если версия нужной для программы библиотеки не соответствует той, что установлена в вашей системе - вам понадобится обновить эту библиотеку.
   - Если это все еще не помогает, проверьте, нормально ли работает эта программа с вашей архитектурой (некоторые программы еще не были портированы во все системы UNIX®). Проверьте также, правильно ли вы сконфигурировали программу (например, когда выполняли команду `configure`) для своей архитектуры.

3. `parse error`

4. Проанализируйте сообщение об ошибке, выданное `make`'ом. Обычно последние строки должны содержать название каталога (сообщение, типа `make[1]: Leaving directory '/home/queen/Project/foo'`). Выберите сообщение с самым старшим номером. Чтобы убедиться в том, что это именно тот каталог, зайдите в него и выполните `make`, чтобы получить ту же самую ошибку.

Это довольно сложная для решения проблема, т.к. зачастую это ошибка, находящаяся в определенной строке, но уже после того, как компилятор ее обнаружил. Иногда это просто не определенный ранее тип данных. Если вы встречаете сообщение об ошибке следующего вида:

```
main.c:1: parse error before `glloq_t
main.c:1: warning: data definition has no type or storage class
```

это означает, что не определен тип данных `glloq_t`. Решение этой проблемы более или менее похоже на решение предыдущей проблемы.

4. `no space left on device`

   Эту проблему легко решить: недостаточно свободного дискового пространства для создания бинарного файла из исходного кода. Решение заключается в освобождении места на разделе с каталогом установки (удалите временные файлы или файлы с исходными текстами, деинсталлируйте программы, которыми вы не пользуетесь, измените размер раздела и т.п.). Проверьте, чтобы он был распакован не в `/tmp`, а в `/usr/local/src`, во избежание бесполезной траты пространства на разделе `/tmp`. Кроме того, размер этого каталога обычно ограничен несколькими десятками мегабайт вследствие дисковых квот (прим. переводчика). Проверьте, нет ли на вашем диске файлов `core` [5]. Если таковые имеются - удалите их или сделайте так, чтобы они были удалены в том случае, если они принадлежат другому пользователю.

5. `/usr/bin/ld: cannot open -lglloq: No such file or directory`

   Это означает, что программа `ld` (используемая `gcc` на последнем этапе компиляции) не в состоянии найти библиотеку. Чтобы задействовать библиотеку, `ld` ищет файл, чье имя находится в аргументах типа `-l<библиотека>`. Этот файл - `lib<библиотека>.so`. Если `ld` не в состоянии найти ее - выводится сообщение об ошибке. Для решения проблемы, следуйте указанным ниже инструкциям:

   a. Проверьте существование файла на диске при помощи команды `locate`. Обычно, графические библиотеки могут быть найдены в `/usr/X11R6/lib`. Например:

      ```
      $ locate libglloq
      ```

      Если поиск ничего не дал, вы можете выполнить поиск при помощи команды `find` (т.е.: `find /usr -name libglloq.so*`). Если вы все еще не можете найти библиотеку, вам придется ее установить.

   b. Как только библиотека найдена, проверьте доступна ли она для программы `ld`: файл `/etc/ld.so.conf` определяет место поиска библиотек. Добавьте подозреваемый каталог в конец этого файла (вам может понадобиться перегрузить свой компьютер, чтобы изменения вступили в силу). Вы также можете добавить этот каталог, изменив содержимое переменной окружения `LD_LIBRARY_PATH`. Например, если нужно добавить каталог `/usr/X11R6/lib`, введите:

      ```
      export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/X11R6/lib
      ```

      (если ваш шелл - bash).

   c. Если это все рано не помогает, проверьте, чтобы библиотека имела формат исполняемого файла (или ELF) при помощи команды `file`. Если это символическая ссылка, проверьте, "живая" ли это ссылка, и не указывает ли она на несуществующий файл (например, при помощи `nm libglloq.so`). Файл может иметь неверные разрешения (например, если вы используете учетную запись, отличную от `root`, и если библиотека защищена от чтения).

6. `glloq.c(.text+0x34): undefined reference to `glloq_init'`

   Это проблема символа, который не был решен на последнем этапе компиляции. Обычно это проблема библиотеки. Причин может быть несколько:

   - сперва необходимо выяснить, **предполагалось** ли наличие символа в библиотеке. Например, если это символ, начинающийся с `gtk`, он принадлежит библиотеке `gtk`. Если имя библиотеки можно легко определить (`frobnicate_foobar`), вы можете вывести список символов библиотеки при помощи команды `nm`. Например,

     ```
     $ nm libglloq.so
     0000000000109df0 d glloq_message_func
     000000000010a984 b glloq_msg
     0000000000008a58 t glloq_nearest_pow
     ```

5. Файл, создаваемый системой, когда процесс пытается обратиться к области памяти, доступ к которой ему запрещен. Эти файлы используются для анализа причины такого поведения и устранения проблемы.

```
0000000000109dd8 d glloq_free_list
0000000000109cf8 d glloq_mem_chunk
```

Добавление опции `-o` к `nm` позволит вам вывести в каждой строке имя библиотеки, упростив тем самым поиск. Допустим, что мы ищем символ `bulgroz_max`, тогда простейшим решением будет следующий поиск:

```
$ nm /usr/lib/lib*.so | grep bulgroz_max
$ nm /usr/X11R6/lib/lib*.so | grep bulgroz_max
$ nm /usr/local/lib/lib*.so | grep bulgroz_max
/usr/local/lib/libfrobnicate.so:000000000004d848 T bulgroz_max
```

Превосходно! Символ `bulgroz_max` определен в библиотеке **`frobnicate`** (перед ее именем стоит заглавная буква `T`). Теперь вам осталось только добавить строку `-lfrobnicate` в строку компиляции, отредактировав файл **`Makefile`**: добавьте ее в конец строки, в которых определены переменные `LDFLAGS` или `LFGLAGS` (или, на худой конец, с `CC`), или в строку, соответствующую созданию конечного бинарного файла.

- компиляция производится с версией библиотеки, которая не подходит для данного программного продукта. Прочтите файлы **`README`** или **`INSTALL`**, чтобы узнать, какая версия должна быть использована.
- корректно скомпонованы не все объектные файлы дистрибутива. Отсутствует файл, в котором определена эта функция. Введите `nm -o *.o`, чтобы узнать, какой это файл, и добавьте соответствующий файл `.o` в строку компиляции, если он отсутствует.
- можеть быть проблемная функция или несуществующая переменная. Попробуйте удалить ее: отредактируйте проблемный исходный файл (его имя указано в начале сообщения об ошибке). Это не лучшее решение и оно может привести к непредвиденному поведению программы с ***нарушением сегментации*** при запуске и т.п.).

7. `Segmentation fault (core dumped)`

   Иногда компилятор немедленно “вываливается” и выводит это сообщение об ошибке. По этому поводу я могу только посоветовать вам установить более свежую версию компилятора.

8. no space on `/tmp`

   Для процесса компиляции на различных этапах необходимо временное дисковое пространство; если его не хватает, компиляция прерывается. Поэтому вам может потребоваться очистить раздел, но будьте осторожны, т.к. могут зависнуть некоторые выполняющиеся программы (X-сервер, каналы и др.), если вы удалите некторые файлы. Вы должны соображать, что вы делаете! Если **`/tmp`** является частью раздела, содержащего не только этот каталог (например, корневой раздел), найдите и удалите все файлы **`core`**. Одним из вариантов является изменение размера раздела, если для этого есть достаточно свободного пространства или может быть уменьшен какой-нибудь другой раздел.

9. `make`/`configure` in infinite recursion

   Зачастую это проблема со временем в вашей системе. Программе `make` необходимо знать дату в компьютере и дату проверяемых ею файлов. Она сравнивает даты и использует результат для того, чтобы определить, не является ли цель более старшей, чем зависимости.

   Некоторые проблемы с датой могут привести к тому, что `make` будет бесконечно собирать сам себя (или будет вновь и вновь выполнять сборку поддерева в бесконечной рекурсии). В этом случае применение `touch` (которая здесь используется для настройки проблемных файлов на текущую дату) обычно помогает решить проблему.

   Например:

   ```
   $ touch *
   ```

   Или так (грубо, но эффективно):

   ```
   $ find . | xargs touch
   ```

12.5. Установка

12.5.1. При помощи Make

Теперь, когда все скомпилировано, вы должны скопировать созданные файлы в соответствующее место (обычно это один из подкаталогов `/usr/local`).

Обычно `make` может выполнить эту задачу. Особой целью является цель `install`. Поэтому при использовании `make install` будет выполнена установка требуемых файлов.

Обычно процедура описана в файле `INSTALL` или `README`. Но иногда разработчик забывает предоставить его. В этом случае вы должны установить все самостоятельно.

Для этого скопируйте:

- Исполняемые файлы (программы) в каталог `/usr/local/bin`.
- Библиотеки (файлы `lib*.so`) в каталог `/usr/local/lib`
- Заголовки (файлы `*.h`) в каталог `/usr/local/include` (будьте осторожны - не удалите оригиналы).
- Файлы данных обычно отправляются в `/usr/local/share`. Если процедура инсталляции вам не знакома, вы можете попробовать запустить программы, не копируя файлы данных, и поместить их в нужное место тогда, когда они вас попросят сделать это для (например, в таком сообщении об ошибке: `Cannot open /usr/local/share/glloq/data.db`).
- С документацией дело обстоит немного по-другому:
  - Файлы `man` обычно помещаются в подкаталоги `/usr/local/man`. Обычно эти файлы имеют формат `troff` (или `groff`), а расширением является число или буква `n`. Их именами являются имена команд (например, `echo.1`). Если число - `n`, скопируйте файл в `/usr/local/man/man<n>`. То же самое касается страниц руководств серии `n`.
  - Файлы `info` помещаются в каталог `/usr/info` или `/usr/local/info`

Вот и все! Поздравляем! Теперь вы готовы к компиляции всей операционной системы!

12.5.2. Проблемы

Если вы только что установили свободный программный продукт, например, GNU tar, и если при его запуске запускается другая программа или она работает не так, как работала, когда вы тестировали ее непосредственно из каталога `src`, то это проблема переменной окружения `PATH`, т.к. поиск программ выполняется сначала в каталогах, перечисленных в этой переменной, а уж потом в том каталоге, в который вы установили новую программу. Проверьте это, выполнив `type -a <программа>`.

Решением будет помещение установочного каталога выше в переменной `PATH` и/или удаление/переименование файлов, которые выполнялись, когда их об этом не просили, и/или переименование ваших новых программ (в этом примере в `gtar`), чтобы такие недоразумения больше не возникали.

Вы также можете создать алиас, если шелл позволяет сделать это (например, сказать, что `tar` означает `/usr/local/bin/gtar`).

12.6. Поддержка

12.6.1. Документация

Различные источники с полезной документацией:

- HOWTO - небольшие документы, раскрывающие определенные вопросы (обычно далекие от того, что нам здесь нужно, но иногда довольно полезные). Ищите их на своем диске в `/usr/share/doc/HOWTO` (не всегда, иногда они оказываются где-нибудь в другом месте; проверьте это при помощи команды `locate HOWTO`).

- Страницы руководства. Введите `man <команда>`, чтобы получить документацию по команде `<команда>`,
- Страницы Info. Введите `info <команда>`, чтобы получить документацию по команде `<команде>`. Страницы Info представляют собой более исчерпывающие страницы руководства и содержат информацию с гиперссылками. Команда `info` также может отображать обычные страницы руководств.
- Специализированная литература. Некоторые крупные издатели начали публиковать книги о свободных системах (особенно о GNU/Linux). Часто это полезно в том случае, если вы новичок и не понимаете всех терминов существующей документации.

12.6.2. Техническая поддержка

Если ваш пакет Mandrakelinux включает в себя техническую поддержку, вы можете запросить у команды технической поддержки информацию о своей системе.

Вы также можете рассчитывать на помощь сообщества свободного ПО:

- ***группы новостей*** (в Usenet) comp.os.linux.* (`news:comp.os.linux.*`) отвечают на все вопросы о GNU/Linux. Группы новостей `comp.os.bsd.*` затрагивают вопросы систем BSD. Могут существовать и другие группы новостей, касающиеся других систем UNIX®. Помните, что сначала некоторое время нужно просто почитать их, а уж потом уже постить в них свои сообщения.
- Некоторые ассоциации или группы энтузиастов в сообществе свободного ПО добровольно предлагают свою техническую поддержку. Наилучший способ найти самую близкую к вам - проверить списки на специализированных веб-сайтах или в течение некоторого времени читать соответствующие группы новостей.
- Некоторые ***каналы IRC*** предлагают помощь ***гуру*** в режиме реального времени (но вслепую). Но не дай вам бог встретить там меня - порву в немецкий крест :) (прим. переводчика). Поищите, к примеру, канал `#linux` в большинстве сетей IRC или `#linuxhelp` в IRCNET.
- В качестве последнего средства задайте вопрос разработчику ПО (если он указал свое имя и свой адрес ***email*** в файле дистрибутива), если вы УВЕРЕНЫ, что нашли ошибку (которая может быть следствием только вашей архитектуры, хотя свободное программное обеспечение прежде всего подразумевается как портируемое).

12.6.3. Как найти свободное программное обеспечение

При поиске свободного ПО вам могут помочь эти ссылки:

- огромный FTP-сервер ftp.ibiblio.org (`ftp://ftp.ibiblio.org`) или одно из его зеркал
- следующие веб-сайты образуют каталог из множества свободных программных продуктов, которые могут использоваться на UNIX®-платформах (но среди них можно найти и собственническое ПО):
  - FreshMeat (`http://www.freshmeat.net/`) - наверное, самый полный сайт.
  - SourceForge.net (`http://sourceforge.net/`) - самый большой в мире веб-сайт разработки ПО с открытыми исходными текстами с самым большим из доступных в Интернете репозиторием текстов и приложений Open Source.
  - GNU Software (`http://www.gnu.org/software/`) - исчерпывающий список всего программного обеспечения GNU. Конечно же, все оно является свободным, а на большинство распространяется действие GPL.

- вы также можете воспользоваться поиском в таких поисковых системах, как Google™ (`http://www.google.com/`) и Lycos™ (`http://www.lycos.com/`), и сделать запрос вида: **`+<software>+download`** или **`"download software"`**.

12.7. Подтверждения и благодарности

- Вычитка и противные комментарии (в алфавитном порядке): Sebastien Blondeel, Mathieu Bois, Xavier Renaut and Kamel Sehil.
- ***Бета-тестирование***: Laurent Bassaler
- Перевод на английский: Fanny Drieu; Редактор английской версии: Hoyt Duff

Глава 13. Компиляция и установка новых ядер

Наряду с монтированием файловых систем и сборкой программного обеспечения из исходных кодов, большинство затруднений у новичков несомненно вызывает компиляция ядра. В общем случае компилировать новое ядро вовсе необязательно, т.к. ядро, установленное с Mandrakelinux, поддерживает порядочное количество устройств (в действительности их даже больше, чем вы можете себе представить, или вам может когда-нибудь понадобиться), включая набор патчей и тому подобное. Однако...

Возможно, вы захотите сделать это только для того, чтобы узнать "что при этом происходит". Не говоря уже о том, чтобы заставить ваш ПК и кофеварку работать усерднее, чем обычно, но ненамного. Причины, по которым вы захотели бы откомпилировать свое собственное ядро, варьируются в диапазоне от отключения одной опции до повторной сборки нового экспериментального ядра. В любом случае цель этой главы - гарантировать дальнейшую работу вашей кофеварки и после компиляции.

Для перекомпиляции ядра существуют и другие веские причины. Например, вы узнали что в используемом вами ядре обнаружена ***ошибка*** безопастности, которая была исправлена в более поздней версии, или что в новое ядро включена поддержка нужного вам устройства. Несомненно, в этих случаях вы можете дождаться и бинарных обновлений, однако обновление исходных кодов ядра и сборка нового ядра своими руками обеспечивает более быстрое решение (по крайней мере на быстрой машине).

В любом случае, что бы вы ни делали, запаситесь кофе.

13.1. Обновление ядра при помощи бинарных пакетов

Перед тем, как погрузиться в компиляцию ядра из исходных кодов, мы подробно опишем простую процедуру, когда вы просто хотите или вам необходимо обновить свое ядро при помощи RPM-пакетов, скомпилированных для вашей версии Mandrakelinux. В примере будет принято, что новое ядро - `kernel-2.6.10-5mdk`, а старое (текущее) - `kernel-2.6.10-1mdk`.

1. **Установка нового ядра.** Выполните в окне терминала команду `urpmi kernel-2.6.10-5mdk`. Если вы не знаете версию ядра, просто введите `urpmi kernel` и выберите для своей системы соответствующее ядро из списка предложенных.
2. **Проверка работы ядра.** Только что установленное ядро становится ядром по умолчанию. Кроме того в меню начального загрузчика (LILO, GRUB, ELILO...) появится новый пункт с именем, типа `2610-5`. Перезагрузите свой компьютер и выберите этот пункт для загрузки нового ядра. Выполните все проверки, которые вы посчитаете нужными, чтобы убедиться в правильности работы нового ядра.
3. **Удаление старого ядра (необязательно).** После того, как вы убедились в том, что новое ядро нормально работает на вашем компьютере, вы можете удалить файлы, относящиеся к старому ядру. Для этого в окне терминала выполните команду `urpme kernel-2.6.10-1mdk`. Конфигурация начального загрузчика будет обновлена автоматически.

13.2. Из исходных кодов

В общем случае исходные коды вы можете получить из двух мест:

1. **Официальное ядро Mandrakelinux.** В каталоге SRPMS любого из зеркал (`http://www.mandrakelinux.com/en/cookerdevel.php3`) `Cooker` вы найдете следующие пакеты:

   kernel-2.6.??.?mdk-?-?mdk.src.rpm

   Исходные коды ядра для компиляции используемого в дистрибутиве ядра. Оно сильно изменено внесенными дополнительными функциональными возможностями.

   kernel2.6-linus2.6.??-?mdk.src.rpm

   Стандартное ядро в том виде, как оно было опубликовано разработчиками ядра GNU/Linux.

   Получение официального ядра Mandrakelinux является рекомендуемой опцией: просто загрузите RPM с исходными кодами, установите их (как `root`) и переходите к Разд. 13.4.

2. **Репозиторий официального ядра Linux.** Главным сервером с исходными текстами ядра является ftp.kernel.org (`ftp://ftp.kernel.org`), но существует огромное число зеркал с именами типа ftp.xx.kernel.org (`ftp://ftp.xx.kernel.org`), где xx (`xx`) представляет собой ISO-код страны. После официального объявления о доступности ядра вы должны подождать по крайней мере два часа, пока будут обновляться зеркала.

   На всех этих серверах FTP исходные тексты ядра находятся в каталоге `/pub/linux/kernel`. Перейдите в каталог с интересующей вас веткой: несомненно это будет `v2.6`. Никто не запрещает вам пробовать экспериментальные версии или использовать старые версии 2.4. Файлы с исходными кодами ядра носят называние `linux-<kernel_version>.tar.bz2`, например `linux-2.6.10.tar.bz2`.

   Вы также можете применить патчи к исходным кодам ядра, чтобы обновить их по нарастающей: так, если у вас уже есть исходные коды ядра версии 2.6.8, и вы хотите обновиться до ядра 2.6.10, вам не нужно загружать все исходные коды 2.6.10, вы можете загрузить просто ***патчи*** `patch-2.6.9.bz2` и `patch-2.6.10.bz2`[1]. Как правило, это будет хорошей идеей, т.к. в настоящее время исходные коды "весят" десятки МБ.

13.3. Распаковка исходных кодов, применение патчей к ядру (если необходимо)

Все этапы, описанные в этом и в следующих разделах этой главы, должны выполняться с правами `root`'а.

Исходные коды ядра должны находиться в `/usr/src`. Поэтому вы должны перейти в этот каталог и распаковать туда исходные коды:

```
$ cd /usr/src
$ mv linux linux.old
$ tar xjf /путь/к/linux-2.6.8.tar.bz2
```

Команда `mv linux linux.old` является необходимой: потому как у вас уже могут быть исходные коды другой версии ядра. Эта команда гарантирует вам, что вы не перезапишете их. После того, как архив распакован, вы получаете каталог `linux-<version>` (где `<version>` - это версия ядра) с исходными кодами нового ядра. Для удобства вы можете сделать на него ссылку (`ln -s linux-<version> linux`).

Теперь о патчах. Допустим, что вы хотите ***пропатчить*** ядро `2.6.6` до `2.6.10`, и вы уже загрузили все необходимые для этого патчи: переходите в новый созданный каталог `linux`, а затем применяйте патчи:

```
$ cd linux
$ bzcat /путь/к/patch-2.6.9.bz2 | patch -p1
$ bzcat /путь/к/patch-2.6.10.bz2 | patch -p1
$ cd ..
```

Вообще говоря, для переход от версии `2.6.x` к `2.6.y`, требуется, чтобы вы применили по порядку все патчи с номерами `2.6.x+1`, `2.6.x+2`, ..., `2.6.y-1`, `2.6.y`. Чтобы вернуться от `2.6.y` к `2.6.x`, точно также повторите эту же процедуру, но применяя патчи в обратном порядке с опцией `-R` для команды `patch` (`R` - это сокращение от *Reverse*). Таким образом, чтобы вернуться от ядра `2.6.10` назад к ядру `2.6.8`, вы должны сделать:

```
$ bzcat /путь/к/patch-2.6.10.bz2 | patch -p1 -R
$ bzcat /путь/к/patch-2.6.9.bz2 | patch -p1 -R
```

Если вы хотите проверить, будет ли патч применен корректно, перед тем как применить его на самом деле, добавьте в команду `patch` опцию `--dry-try`.

1. Этот путь имеет отклонение в нумерации версий - версия 2.6.8.1. Вам нет необходимости использовать или загружать эту версию. Если, конечно, вы вы собираетесь остановиться на 2.6.8.1...

Далее, для большей ясности (и для того, чтобы вы знали, где находитесь), вы можете переименовать `linux` с отображением в имени версии ядра и создать на него символическую ссылку:

```
$ mv linux linux-2.6.10
$ ln -s linux-2.6.10 linux
```

Теперь пора перейти к конфигурированию.

13.4. Конфигурирование ядра

Чтобы начать, перейдите в каталог `/usr/src/linux`.

Для начала небольшая хитрость: при желании вы можете изменить версию своего ядра. Она определяется четырьмя первыми строками файла `Makefile`:

```
$ head -4 Makefile
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 10
EXTRAVERSION = -1mdkcustom
```

Дальше в файле `Makefile` вы можете увидеть, что версия ядра сформирована следующим образом:

```
KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
```

Все, что вам нужно сделать для установки своей версии - изменить одно из этих полей. Желательно, чтобы вы изменяли только EXTRAVERSION. Скажем, к примеру, вы установили его в `-foo`. При этом новая версия вашего ядра будет 2.6.10-foo. Не стесняйтесь изменять это поле всякий раз, когда собираете новое ядро с разными версиями, так вы сможете протестировать различные опции с сохранением предыдущих версий.

Теперь приступим к конфигурированию. У вас есть выбор между:

- `make xconfig` - графический интерфейс на базе `qt`;
- `make gconfig` - графический интерфейс на базе `gtk+`;
- `make menuconfig` - интерфейс на базе `ncurses`;
- `make config` - самый простой текстовый интерфейс, строка за строкой, раздел за разделом;
- `make oldconfig` аналогично указанному выше, но на базе предыдущей конфигурации. Смотрите Разд. 13.5.

Вы можете осуществлять конфигурирование поэтапно раздел за разделом, но вы можете можете пропускать разделы и переходить к интересующим вас разделам, если вы используете `menuconfig`, `xconfig` или `gconfig`. Доступные опции: **y** для Yes (функция, компилируемая в виде неотъемлемой части ядра), **m** для Module (функция, компилируемая в виде модуля) или**n** для No (функция, не включаемая в ядро).

В командах `make xconfig`, `make gconfig` и `make menuconfig` опции сгруппированы в виде иерархии. Например, `Processor family` находится в `Processor type and features`.

Для команд `xconfig` и `gconfig` кнопка Main Menu используется для возвращения в главное меню из иерархической группы;кнопка Next выполняет переход в следующую группу опций; и кнопка Prev выполняет переход в предыдущую группу. В `menuconfig` для выбора раздела используйте кнопку **Enter**, а для изменения состояния опций используйте клавиши **y**, **m** или **n**, или же нажмите клавишу **Enter** для выбора опций из списка предлагаемых. Exit служит для выхода из раздела или из конфигурации, если вы находитесь в главном меню. Также еще имеется кнопка Help.

Мы не собираемся перечислять здесь все опции, так как их насчитывается несколько сотен. Более того, если вы добрались до этой главы, то вы наверняка уже знаете, что делаете. Поэтому вам остается просмотреть конфигурацию ядра и установить/снять любые опции на свое усмотрение. Однако вот несколько советов о том, как избежать сборки неработоспособного ядра:

1. если вы не используете начальный электронный диск (`initrd`), **никогда** не компилируйте в виде модулей драйверы, необходимые для монтирования своей корневой файловой системы (драйверы

оборудования и драйверы файловых систем)! И, если вы используете начальный электронный диск, установите **Y** для поддержки ext2FS, так как эта файловая система используется для ramdisk'ов. Вам также необходимо включить поддержку `initrd`;

2. если в вашей системе имеются сетевые карты, откомпилируйте их драйверы в виде модулей. В результате вы сможете выбрать, какая из карт будет первой, какая второй и так далее, путем помещения соответствующих алиасов в файл `/etc/modules.conf`. Если вы компилируете драйверы в тело ядра, порядок, в котором они будут загружены, будет зависеть от порядка компоновки, который может вас не устроить;
3. и в заключение: если вы не знаете, на что влияет опция - прочтите справку! Если текст справки все еще вас не удовлетворяет - тогда оставьте опцию как есть. (Чтобы получить справку по `config` и `oldconfig`, нажмите клавишу **?**.)

Вуаля! Конфигурирование полностью закончено. Сохраните свою конфигурацию и выходите.

13.5. Сохранение и повторное использование файлов конфигурации ядра

Конфигурация ядра хранится в файле `/usr/src/linux/.config`. Ее резервная копия находится в `/boot/config-<25@A8O>`, хранение ее в виде ссылки является хорошим решением. Но сохраняйте также и свои собственные конфигурации для различных ядер, ведь это просто вопрос назначения разных имен файлам конфигураций.

Одним из вариантов является именование файлов конфигурации по версии ядра. Скажем, вы изменили версию ядра как показано в Разд. 13.4, тогда вы можете сделать следующее:

```
$ cp .config /root/config-2.6.10-foo
```

Если вы решите выполнить обновление до `2.6.11` (к примеру), вы сможете повторно использовать этот файл, т.к. различия в конфигурациях этих двух ядер будет очень незначительными. Просто воспользуйтесь резервной копией:

```
$ cp /root/config-2.6.10-foo .config
```

Но копирование резервной копии не означает, что ядро уже готово к компиляции. Вы должны снова выполнить команду `make menuconfig` (или любую другую на ваше усмотрение), потому что для успешной компиляции этими командами должны быть созданы и/или изменены некоторые файлы.

Однако, не говоря уже о рутинной работе по повторному прохождению всех меню, вы могли пропустить какие-нибудь новые интересные опции. Вы можете избежать этого при помощи команды `make oldconfig`. Она обладает двумя преимуществами:

1. она довольно быстрая;
2. если в ядре появилась новая опция, и ранее она отсутствовала в вашем файле конфигурации, произойдет остановка в ожидании вашего выбора.

После того, как вы скопируете свой `.config` в домашний каталог `root`'а, как описано выше, запустите команду `make mrproper`. Она обеспечит, что от старой конфигурации ничего не останется, а вы получите чистое ядро.

Теперь пора запустить компиляцию.

13.6. Компиляция ядра и модулей, установка зверя

Сначала небольшое примечание: если вы перекомпилируете ядро той же самой версии, что уже имеется в вашей системе, то тогда сначала должны быть удалены старые модули. Например, если вы перекомпилируете ядро `2.6.10`, вы должны удалить каталог `/lib/modules/2.6.10`.

Компиляция ядра и модулей, а затем и установка модулей, выполняется при помощи следующих команд:

```
make clean
make all
make modules_install install
```

Небольшой словарик: любые аргументы типа `clean`, `all` и т.п. называются **целями**. Обратите внимание, что, начиная с ядра `2.6`, существует цель под названием `all`. Выполнение этой цели - то же самое, что выполнение (на архитектуре x86) целей `bzImage` и `modules`. Эта новая опция будет создавать предпочтительные цели для любой заданной архитектуры. До `2.6` для каждой архитектуры для компиляции ядра имелась своя опция с отдельным именем. Если вы задаете для `make` как показано выше несколько целей, они будут выполнены в порядке указания. Но в случае сбоя одной из целей, `make` дальнейшую сборку продолжать не будет[2].

Давайте взглянем на различные цели и узнаем, что же они делают:

- `bzImage`: эта цель соберет ядро. Обратите внимание, что эта цель верна только для процессоров **x86** и **x86_64**. Эта цель также создает файл `System.map` для этого ядра. Позже мы увидим, для чего используется этот файл;
- `modules`: эта цель создаст модули для только что собранного вами ядра. Если вы выбрали сборку без модулей, эта цель ничего делать не будет;
- `all`: эта цель создаст образ ядра предпочтительного типа для заданной архитектуры и модулей;
- `modules_install`: эта цель установит модули. По умолчанию модули будут установлены в каталог `/lib/modules/<25@A8O-O4@O>`. Эта цель также вычисляет зависимости модулей;
- `install`: эта последняя цель в конце скопирует ядро и модули в надлежащие места и изменит конфигурацию начального загрузчика так, чтобы новое ядро было доступно при загрузке. Не используйте ее, если вы предпочитаете выполнять установку вручную, как описано в Разд. 13.7.

Важно соблюдать порядок целей `modules_install installc` тем, чтобы на самом деле сначала устанавливались модули. В противном случае `initrd` окажется неправильным и ядро не сможет нормально загрузиться.

На данный момент у нас все откомпилировано, корректно установлено и готово к проверке! Просто перегрузите свою машину и выберите новое ядро в меню загрузки. Обратите внимание, что старое ядро остается доступным, поэтому вы можете воспользоваться им, если у вас возникнут проблемы с новым ядром. Тем не менее вы можете выбрать ручную установку ядра и изменить меню загрузки вручную. Мы рассмотрим это в следующем разделе.

13.7. Ручная установка нового ядра

Процедуры в этом разделе применимы к архитектуре **x86**. Если у вас другая архитектура, расположение файлов и сами устанавливаемые файлы могут быть другими.

Ядро находится в `arch/i386/boot/bzImage`. Стандартный каталог, в который устанавливается ядро - `/boot`. Вам также необходимо скопировать файл `System.map`, чтобы обеспечить правильную работу некоторых программ (например, `top`). И опять же хорошая практика: давайте этим файлам имена по версии ядра. Допустим, что ваше ядро версии `2.6.8-foo`. Последовательность ваших команд следующая:

```
$ cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.8-foo
$ cp System.map /boot/System.map-2.6.8-foo
```

Теперь вам необходимо сообщить начальному загрузчику о своем новом ядре. Существует два начальных загрузчика: GRUB или LILO. Учтите, что Mandrakelinux по умолчанию настроен на использование LILO.

2. В этом случае, если компиляция не удалась, это означает, что в ядре имеется ошибка... Если это так и есть, пожалуйста, сообщите о ней!

13.7.1. Обновление LILO

Самым простой способ для обновления LILO - использование drakboot(смотрите главу в книге *Стартовое руководство*). Или же вы можете вручную отредактировать файл конфигурации, как описано ниже.

Конфигурационный файл LILO - `/etc/lilo.conf`. Так выглядит типичный файл `lilo.conf`:

```
boot=/dev/hda
map=/boot/map
default="linux"
keytable=/boot/es-latin1.klt
prompt
nowarn
timeout=50
message=/boot/message
menu-scheme=wb:bw:wb:bw
image=/boot/vmlinuz
        label="linux"
        root=/dev/hda1
        initrd=/boot/initrd.img
        append="devfs=mount resume=/dev/hda5"
        read-only
other=/dev/fd0
        label="floppy"
        unsafe
```

Файл `lilo.conf` состоит из главного раздела, после которого следуют разделы для каждой из операционных систем. В приведенном выше примере главный раздел состоит из следующих директив:

```
boot=/dev/hda
map=/boot/map
default="linux"
keytable=/boot/es-latin1.klt
prompt
nowarn
timeout=50
message=/boot/message
menu-scheme=wb:bw:wb:bw
```

Директива `boot=` сообщает LILO, куда установлен загрузочный сектор; в данном случае это MBR (*Master Boot Record - главная загрузочная запись*) на первом жестком диске IDE. Если вы хотите сделать загрузочную дискету с LILO, просто замените `/dev/hda` на `/dev/fd0`. Директива `prompt` заставляет LILO при запуске показывать меню. Т.к. установлен тайм-аут, LILO запустит образ по умолчанию через 5 секунд (`timeout=50`). Если вы удалите директиву `timeout`, LILO будет ждать до тех пор, пока вы не введете что-нибудь.

Далее следует раздел `linux`:

```
image=/boot/vmlinuz
        label="linux"
        root=/dev/hda1
        initrd=/boot/initrd.img
        append="devfs=mount resume=/dev/hda5"
        read-only
```

Раздел для загрузки ядра GNU/Linux всегда начинается с директивы **`image=`**, после которой следует полный путь к действительному ядру GNU/Linux. Как и любой другой раздел он содержит директиву `label=` в качестве уникального идентификатора, здесь это `linux`. Директива `root=` сообщает LILO, на каком из разделов находится файловая система для этого ядра. Ваша конфигурация может быть другой... Директива `read-only` сообщает LILO, что при запуске корневая файловая система должна быть примонтирована в режиме только для чтения: если этой директивы здесь нет - вы получите сообщение с предупреждением. Строка `append` определяет опции, передаваемые ядру во время загрузки.

Затем следует раздел для дискеты:

```
other=/dev/fd0
        label="floppy"
        unsafe
```

В действительности раздел, начинающийся с `other=`, используется LILO для запуска любой операционной системы, отличной от GNU/Linux: аргумент этой директивы определяет местонахождение загрузочного сектора этой системы, и в данном случае это загрузка с дискеты.

Ну что ж, теперь настало время добавить раздел для нашего нового ядра. Вы можете поместить этот раздел куда угодно после главного раздела, но не помещайте его внутрь другого раздела. Вот как это должно выглядеть:

```
image=/boot/vmlinuz-2.6.10-foo
        label="foo"
        root=/dev/hda1
        read-only
        append="devfs=mount resume=/dev/hda5"
```

Лишний раз напомним: настройте пункт согласно конфигурации своей системы.

Итак, вот как будет выглядеть наш `/etc/lilo.conf` после внесения изменений, снабженный несколькими дополнительными комментариями (все строки, начинающиеся с #), которые будут проигнорированы программой LILO:

```
#
# Главный раздел
#
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
message=/boot/message
# Что будет загружено по умолчанию. Давайте укажем здесь наше новое ядро:
default="foo"
# Выводим на экран приглашение...
prompt
# ... ожидаем 5 секунд
timeout=50
#
# Наше новое ядро: образ по умолчанию
#
image=/boot/vmlinuz-2.6.10-foo
        label="foo"
        root=/dev/hda1
        read-only
        append="devfs=mount resume=/dev/hda5"
#
# Оригинальное ядро
#
image=/boot/vmlinuz
        label="linux"
        root=/dev/hda1
        read-only
        append="devfs=mount resume=/dev/hda5"
#
# Раздел для дискеты
#
other=/dev/floppy
        label="floppy"
        unsafe
```

Было бы неплохо, чтобы так выглядел ваш файл `/etc/lilo.conf`... но, опять же, не забудьте настроить его согласно своей собственной конфигурации.

Теперь, когда файл соответствующим образом был изменен, но, в отличие от GRUB, который в этом не нуждается, вы должны сообщить LILO, чтобы тот изменил загрузочный сектор:

```
$ lilo
Added foo *
Added linux
Added floppy
$
```

Таким образом, вы можете откомпилировать сколько угодно ядер, добавив нужное количество разделов. Все, что вам нужно теперь сделать - перезагрузить свою машину и протестировать новое ядро. Заметьте, что если LILO при установке показывает какие-либо ошибки, это означает, что никакие изменения не были сделаны. LILO изменят вашу конфигурацию только, если в процессе не возникает никаких ошибок.

13.7.2. Обновление Grub

Понятное дело, что нужно сохранить возможность загрузки вашего текущего ядра! Самым простым способом обновления GRUB является использование drakboot (смотрите главу Изменение загрузочной конфигурации в книге *Стартовое руководство*). Или же вы можете вручную отредактировать файл конфигурации, как описано ниже.

Вам нужно отредактировать файл `/boot/grub/menu.lst`. Так выглядит типичный файл `menu.lst` после установки вашего дистрибутива Mandrakelinux и до внесения изменений:

```
timeout 5
color black/cyan yellow/cyan
i18n (hd0,4)/boot/grub/messages
keytable (hd0,4)/boot/fr-latin1.klt
default 0

title linux
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5

title failsafe
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5  failsafe

title floppy
root (fd0)
chainloader +1
```

Этот файл состоит из двух частей: заголовка с общими параметрами (первые пять строк), и образов, каждый из которых соответствует различному ядру GNU/Linux или другой ОС. `timeout 5` определяет время (в секундах), в течение которого GRUB ожидает ввода данных перед тем, как автоматически загрузить образ по умолчанию (он определяется директивой `default 0` в общих параметрах, т.е. в данном случае первый образ). Директива `keytable` (если она присутствует) определяет, где найти раскладку вашей клавиатуры. В этом примере это французская раскладка. Для русской раскладки можно установить, например, `keytable (hd0,4)/boot/ru4.klt`. Если не определена ни одна из раскладок, под клавиатурой будет подразумеваться простая клавиатурой `QWERTY`. Все видимые вами `hd(x,y)` ссылаются на раздел под номером `y` на диске под номером `x` (как видит их BIOS). Обратите внимание на то, что их перечисление начинается с нуля!

Затем следуют различные образы. В этом примере определено три образа: `linux`, `failsafe` и `floppy`.

- Раздел `linux` начинается с сообщения GRUB'у о загружаемом ядре (`kernel hd(0,4)/boot/vmlinuz`), после него следуют опции, передающиеся ядру.В данном случае `root=/dev/hda5` сообщит ядру, что корневая файловая система находится на `/dev/hda5`. В действительности `/dev/hda5` - это эквивалент `hd(0,4)` для GRUB, но никто не запрещает ядру находиться на другом разделе, отличающемся от содержащего корневую файловую систему;
- Раздел `failsafe` очень похож на предыдущий за тем исключением, что мы передадим ядру аргумент (`failsafe`), который сообщит ему перейти в "однопользовательский" режим или режим "спасения";
- Раздел `floppy` просто загружает вашу систему с дискеты в первом дисководе, какая бы система на ней не была установлена. Это может быть загрузочный диск Windows® или даже ядро GNU/Linux на дискете;

В зависимости от уровня безопасности, который вы используете в своей системе, некоторые описанные здесь разделы могут отсутствовать в вашем файле.

Теперь к делу. Нам нужно добавить новый раздел, чтобы сообщить GRUB'у новом ядре. В этом примере мы он будет размещен перед другими разделами, но ничто не мешает вам поместить его куда-нибудь в другое место:

```
title foo
kernel (hd0,4)/boot/vmlinuz-2.6.10-foo root=/dev/hda5
```

Не забудьте настроить файл согласно вашей конфигурации! Здесь корневая файловая система GNU/Linux - `/dev/hda5`, но в вашей системе она может быть и где-нибудь в другом месте.

Ну вот и все. В отличии от LILO, как мы увидим ниже, больше ничего делать не надо. Просто перезагрузите свой компьютер и вы увидите свою, только что добавленную, запись. Просто выберите ее из меню, и будет загружено ваше новое ядро.

Если вы скомпилировали ядро с поддержкой видеобуфера, вы, вероятно, захотите его задействовать: в этом случае вам нужно добавить директиву для ядра, которая сообщит ему, с каким разрешением вы хотите загрузиться. Список режимов доступен в файле `/usr/src/linux/Documentation/fb/vesafb.txt` (но только в случае видеобуфера VESA! Иначе обратитесь к соответствующему файлу). Для режима 800x600 в 32 бита[3] номер режима равен `0x315`, поэтому вам нужно добавить следующую директиву:

```
vga=0x315
```

и ваша запись теперь станет похожей на:

```
title foo
kernel (hd0,4)/boot/vmlinuz-2.6.8-foo root=/dev/hda5 vga=0x315
```

Для получения дополнительной информации обратитесь, пожалуйста, к информационным страницам о GRUB (`info grub`).

3. 8 бит означает 2^8 цветов, т.е. 256; 16 бит означает 2^{16} цветов, т.е. 64k или 65536; в 24 битах, как и в 32 битах, цвет закодирован в 24 битах, т.е. 2^{24} возможных цветов, другими словами точно 16M или чуть больше 16 миллионов.

Приложение A. Глоссарий

account

учетная запись, аккаунт; в системе UNIX® это комбинация, состоящая из имени, личного каталога, пароля и shell'а, которая позволяет пользователю подключиться к этой системе.

alias

алиас, псевдоним; механизм, используемый в shell'е для замены одной строки на другую перед выполнением команды. Вы можете увидеть все определенные в текущем сеансе алиасы, набрав в консоли `alias`.

ACPI

(*Advanced Configuration and Power Interface, усовершенствованный интерфейс конфигурирования системы и управления энергопитанием*) возможность, используемая для определения и настройки оборудования и управления питанием. В отличие от APM, который полагается только на BIOS, ACPI полагается также на операционную систему, упрощая его контроль со стороны пользователя. ACPI также несет в себе возможности управления питанием для серверов и рабочих станций.

APM

(*Advanced Power Management, расширенное управление питанием*) возможность, используемая некоторыми BIOS'ами для перевода машины в состояние standby ("ожидание") после заданного промежутка времени бездействия. На портативных компьютерах (laptop) APM также отвечает за возврат состояния батарей и (если это поддерживается) оставшийся срок службы батарей. Однако более современные ноутбуки основаны на ACPI, а не на APM.
См. также: ACPI.

ARP

(*Address Resolution Protocol, протокол разрешения адресов*) Интернет-протокол, используемый для преобразования Интернет-адреса в физический (на уровне оборудования) адрес в локальной вычислительной сети. Его использование ограничено сетями, которые поддерживают широковещательные запросы на аппаратном уровне (hardware broadcasting).

ASCII

(*American Standard Code for Information Interchange, Американский стандартный код для обмена информацией*) стандартный код, используемый для хранения на компьютере символов, включая управляющие символы. Многие 8-битные коды (такие как ISO 8859-1 - стандартный набор символов Linux, если вы не выбрали ничего другого, наподобие UTF-8), содержат ASCII в своей нижней половине.
См. также: ISO 8859, UTF-8.

assembly language

язык ассемблера; язык программирования, наиболее близкий к компьютеру, и называемый поэтому "низкоуровневым" языком программирования. Его преимущество заключается в скорости, т.к. программы на ассемблере написаны на основе инструкций процессора или вообще не нуждаются в трансляции при создании исполняемого кода. Его главный недостаток - зависимость от процессора (или архитектуры). Также написание сложной программы является весьма трудоемким процессом. Таким образом ассемблер является самым быстрым языком программирования, но его невозможно портировать с одной архитектуры на другую.

ATAPI

(*AT Attachment Packet Interface, пакетный интерфейс периферийных устройств для AT-совместимых компьютеров*) расширение спецификации ATA (*Advanced Technology Attachment*, более известной как IDE - *Integrated Drive Electronics, встроенный интерфейс накопителей*), которое предоставляет дополнительные команды для управления приводами CD-ROM и накопителями на магнитной ленте. Контроллеры IDE, оснащенные этим расширением, также называются контроллерами EIDE (*Enhanced IDE*).
См. также: IDE.

ATM

(**Asynchronous Transfer Mode, асинхронный режим передачи**) технология коммутации сетевых пакетов фиксированной длины, ориентированная на высокоскоростные (мультимегабитные) оптические сети. Сеть ATM разбивает данные на блоки стандартного размера (53 байта: 48 для данных и 5 для заголовка), что позволяет эффективно передавать их из одной точки в другую.

atomic

элементарный, атомарный; набор операций считается элементарным, когда все они выполняются одновременно и не могут быть прерваны. Обычно это используется для наборов "всего или ничего": либо все операции выполнились успешно, либо ни одна из них не принималась во внимание.

background

фоновый режим; в контексте shell'а, процесс выполняется в фоновом режиме, если вы можете вводить команды, которые захватывались процессом во время его выполнения. Антоним приоритетного режима процесса.
См. также: job, foreground.

backup

резервное копирование, резервирование, бэкап; означает сохранение важных данных на безопасный носитель или в безопасное место. Резервное копирование должно выполняться регулярно, в особенности это касается критической информации и конфигурационных файлов (наиболее важные каталоги для резервирования: `/etc`, `/home` и `/usr/local`). Традиционно для резервирования каталогов и файлов многие люди используют `tar` в сочетании с `gzip` или `bzip2`. Вы можете использовать эти утилиты или программы типа `dump` и `restore`, а также многие другие свободные или коммерческие решения для резервного копирования.

batch

пакетный режим; режим выполнения, когда отправленные процессору задания выполняются последовательно до тех пор, пока не будет выполнено последнее задание.

beep

звуковой сигнал, бип; негромкий писк, издаваемый динамиком вашего компьютера, предупреждая вас о неоднозначной ситуации, когда вы используете завершение команды и, например, когда доступно более одного варианта завершения. Возможно, другие программы тоже будут подавать звуковые сигналы, давая вам знать о некоторых определенных ситуациях.

beta testing

бета-тестирование; название процесса тестирования бета-версии программы. Программы обычно выпускаются в "альфа-", "бета-" и "release candidate"-состояниях для тестирования перед выпуском финального релиза.

binary

бинарный (двоичный) файл, бинарник; в контексте программирования, бинарные файлы представляют собой откомпилированные рабочие программы.

bit

(*BInary digiT, двоичная цифра*) **бит;** однозначное число, принимающее значение 0 или 1, потому что вычисления выполняются по основанию два. Это самая простая единица цифровой информации.

block mode files

файлы блочного режима; файлы, содержимое которых буферизуется. Все операции чтения/записи для таких файлов выполняются через буферы, которые разрешают асинхронную запись на используемое оборудование, а при чтении позволяют избежать обращения диску, если данные уже находятся в буфере.
См. также: buffer, buffer cache, character mode files.

boot

загрузка; процедура, происходящая при включении компьютера, когда выполняется последовательное определение периферийных устройств с последующей загрузкой в память операционной системы.

boot disk

загрузочный диск; диск, содержащий машинный код, необходимый для загрузки операционной системы с жесткого диска (а иногда и с самого диска).

bootloader

начальный загрузчик, загрузчик ОС; программа, запускающая операционную систему. Многие загрузчики предоставляют вам возможность загрузить на выбор одну из нескольких операционных систем, предлагая список в виде меню. Наиболее популярными загрузчиками являются GRUB и LILO, обладающие этой возможностью, и являющиеся очень полезными в системах с двойной или мульти-загрузкой.

BSD

(*Berkeley Software Distribution, программное изделие Калифорнийского университета*) вариант UNIX®, разработанный на факультете вычислительной техники Калифорнийского университета в Беркли, США. Эта версия всегда считалась более технически усовершенствованной, чем другие, и внесла множество новаторских идей в мир вычислительной техники вообще и в UNIX® частности.

buffer

буфер; небольшой кусок памяти фиксированного размера, который может быть связан с файлом блочного режима, системной таблицей, процессом и т.п. Логическую связь между всеми буферами обеспечивает буферный кеш.
См. также: buffer cache.

buffer cache

буферный кэш; важная часть ядра операционной системы, отвечающая за поддержание всех буферов в актуальном состоянии, уменьшение кэша при необходимости, очистку ненужных буферов и др.
См. также: buffer.

bug

ошибка, баг; в особом случае нелогичное или непоследовательное поведение программы или поведение, которое не следует из документации или принятых для программы стандартов. Часто новые возможности программ вносят в них новые ошибки. Согласно истории, этот термин появился во времена перфокарт: мотылек (от англ. bug - жук) заснул в дырке перфокарты, и это нарушило правильную работу программы. Адмирал Грейс Хоппер (Grace Hopper), обнаружив его, воскликнул "Это жук!" ("It's a bug!"), и с тех пор этот термин и прижился. Имейте в виду, что это только одна из многих историй, которые пытаются объяснить причину возникновения термина *bug*.

byte

байт; непрерывная последовательность обычно из восьми бит, результатом интерпретации которой по основанию десять, является число целое число от 0 до 255. Байт всегда является "атомарным элементом" системы, что значит, что это наименьшая единица информации, имеющая адрес.
См. также: bit.

case

регистр; применительно к строкам, регистр - это разница между строчными (маленькими) и прописными (большими) буквами.

CHAP

(*Challenge-Handshake Authentication Protocol, протокол аутентификации с предварительным согласованием вызова*) протокол, используемый провайдерами услуг Интернета для аутентификации своих клиентов. Согласно этой схеме, клиенту (устанавливающей соединение машине) отправляется некоторое значение, на основании которого он вычисляет хэш (hash. Клиент отправляет назад серверу этот хэш для сравнения с хэшем, вычисленным сервером. Этот метод аутентификации в отличие PAP периодически выполняет повторную аутентификацию после первой установки соединения.
См. также: PAP.

character mode files

файлы символьного режима; файлы, содержимое которых не буферизуется. По отношению к физическим устройствам это значит, что все операции ввода/вывода данного устройства производятся немедленно. В операционной системе существуют несколько специальных символьных устройств (`/dev/zero`, `/dev/null` и других), которые соответствуют потокам данных.
См. также: block mode files.

CIFS

(*Common Internet FileSystem, общий протокол доступа к файлам Интернет;*) наследник файловой системы SMB, используемой в системах DOS.
См. также: SMB.

client

клиент; программа или компьютер, которая нерегулярно подключается к другой программе или компьютеру на определенный период времени для отправки управляющих сигналов или получения информации. В случае **одноранговых** систем (peer-to-peer), таких как SLIP или PPP, под клиентом принимается сторона, которая инициализирует соединение, а удаленная сторона, которая принимает

запрос, называется сервером. Клиент является составляющим компонентом **системы клиент-сервер**.
См. также: server.

client/server system
система клиент-сервер; система или протокол, состоящая из **сервера** и одного или нескольких **клиентов**.

command line
командная строка; предоставляется командным процессором и позволяет пользователю непосредственно вводить команды. Также является темой бесконечного "противостояния флейма" ("flame war") между ее приверженцами и противниками.

command mode
командный режим; в Vi или его клонах, это состояние программы, при котором нажатие на клавишу не вставляет символ в редактируемый файл, а выполняет действие, связанное с этой клавишей (если только в вашем клоне нет переназначенных команд и вы не перенастраивали свою конфигурацию). Вы можете выйти из этого режима, набрав одну из команд "возврата в режим вставки": **i**, **I**, **a**, **A**, **s**, **S**, **o**, **O**, **c**, **C** ...

compilation
компиляция; процесс преобразования исходного кода, читабельного для человека, (естественно, после некоторой тренировки) и написанного на одном из языков программирования (например, на C), в бинарный файл, пригодный для считывания машиной.

completion
завершение; способность командного процессора автоматически дополнять вводимую подстроку (обычно по нажатию на клавишу Tab) до имени файла, имени пользователя или другого объекта до тех пор, пока имеет место совпадение.

compression
сжатие, упаковка, компрессия; способ уменьшения размера файлов или уменьшения числа символов, отправленных по каналу связи. Некоторые из программ сжатия файлов: compress, `zip`, `gzip` и `bzip2`.

console
консоль; то, что раньше называли терминалами. Это были машины (экран с клавиатурой), подключенные к центральному мэйнфрейму (большая мощная ЭВМ коллективного пользования). Применительно к PC, физический терминал - это клавиатура и экран.
См. также: virtual console.

cookies
кукисы, печенье :) ; временные файлы, записанные на локальный жесткий диск удаленным веб-сервером. Они позволяют серверу узнать настройки пользователя, когда он снова подключается к серверу.

datagram
дейтаграмма, датаграмма; дискретный блок данных и заголовков с адресами, являющийся основной единицей передачи по IP-сети. Вы также могли слышать другое название - "пакет".

dependencies
зависимости; этапы компиляции, которые должны быть удовлетворены перед переходом к другим этапам для успешной компиляции программы. Этот термин также используется в случае, когда набор программ, которые вы хотите установить, зависит от других программ, которые могут быть установлены или отсутствовать в вашей системе. В этом случае вы можете получить сообщение, сообщающее вам, что системе для продолжения установки необходимо "удовлетворить зависимости".

desktop
рабочий стол, десктоп; Если вы используете X Window System, рабочим столом является область экрана, в которой вы работаете, и в которой отображаются ваши окна и значки (иконки). Также его называют фоном, и обычно он заполнен простым цветом, градиентом или даже изображением.
См. также: virtual desktops.

DHCP

(*Dynamic Host Configuration Protocol, протокол динамической конфигурации хоста*) протокол, разработанный для машин локальной сети для динамического получения IP-адреса и других параметров сети от сервера.

directory

каталог, директория, папка; часть структуры файловой системы. Файлы или другие каталоги могут храниться внутри каталога. Иногда внутри каталога находятся подкаталоги (или ветви). Часто такую структуру называют деревом каталогов. Если вам нужно увидеть содержимое другого каталога, вы должны вывести его список или зайти в него. Файлы внутри каталога подобны листьям дерева, а подкаталоги являются аналогами ветвей. На каталоги распространяются те же ограничения, что и на файлы, хотя разрешения для них имеют несколько другой смысл. Специальные каталоги . и . . ссылаются, соответственно, на сам каталог и на родительский каталог. В графических средах он также известен как папка.

discrete values

дискретные значения; значения, не являющиеся непрерывными. Т.е. между последовательными значениями имеется некий вид "промежутка".

distribution

дистрибутив, процесс распространения; термин, используемый для отделения продуктов одного производителя GNU/Linux от другого. В состав дистрибутива входят: ядро Linux и утилиты, а также программы установки, программы сторонних разработчиков, а иногда и собственническое программное обеспечение.

DLCI

(*Data Link Connection Identifier, идентификатор соединения канального уровня*) идентификатор уникального виртуального соединения типа точка-точка в сети ретрансляции кадров (Frame Relay). DLCI обычно назначается поставщиком сети Frame Relay.

DMA

(*Direct Memory Access, прямой доступ к памяти*) возможность, используемая в архитектуре PC, позволяющая периферийным устройствам выполнять чтение или запись в ОЗУ, минуя CPU. Периферийные PCI-устройства используют захват шины (bus mastering) и не нуждаются в DMA. Мастеринг шины позволяет контроллеру общаться с другими устройствами без использования CPU.

DNS

(*Domain Name System, система доменных имен*) Распределенный механизм, используемый в Интернете для сопоставления имен и адресов. Этот механизм позволяет вам привязать доменное имя к IP-адресу, упрощая тем самым поиск сайта по более удобному имени домена. DNS также позволяет выполнять обратный поиск для получения IP-адреса машины по ее имени.

DPMS

(*Display Power Management System, система управления энергопотреблением дисплеев*) протокол, используемый всеми современными мониторами для управления функциями энергосбережения. Мониторы с поддержкой этих функций обычно называются "green"-мониторами (экологически чистыми).

echo

эхо; возникает, когда набираемые вами символы, например, в поле имени пользователя, отображаются на экране. Также некоторые программы могут скрывать вводимые символы по соображениям безопасности. Примером является приглашение для ввода пароля, показывающее * (или вообще ничего) вместо каждого вводимого символа.

editor

редактор; термин, используемый обычно для программ, редактирующих текстовые файлы (aka текстовые редакторы). Наиболее известные редакторы GNU/Linux - редактор GNU Emacs (Emacs) и редактор UNIX® Vi.

ELF

(*Executable and Linking Format, формат исполняемых и компонуемых модулей*) бинарный формат, используемый в большинстве дистрибутивов GNU/Linux.

email

electronic mail, электронная почта, и-мэйл, "мыло"; способ обмена сообщениями в электронном виде. По аналогии с обычной (бумажной) почтой для нормальной доставки электронное письмо должно иметь адрес получателя и адрес отправителя. Отправитель должен иметь адрес в виде "отправитель@домен.отправителя", а получатель должен иметь адрес типа "получатель@домен.получателя". Электронная почта - это очень быстрый метод связи и обычно доставка письма адресату занимает несколько минут, в какой бы точке мира он не находился. Чтобы написать электронное письмо, вам нужно воспользоваться почтовым клиентом, таким как pine или `mutt` (текстовый режим), или графическими клиентами наподобие KMail.

environment

окружение, среда; совокупность условий, в которых выполняется процесс. В нее включается вся информация, необходимая операционной системе для управления процессом, и все, что требуется процессору для корректного выполнения процесса.
См. также: process.

environment variables

переменные окружения; часть окружения процесса. Переменные окружения могут быть непосредственно просмотрены в командном процессоре.
См. также: process.

escape

экранирование, эскейп; в контексте shell'а, заключение в кавычки строки для предотвращения ее интерпретации командным процессором. Например, если вам нужно использовать в командной строке пробелы и перенаправить результат по каналу в другую команду, вам потребуется заключить первую команду в кавычки или поставить перед пробелами знак \ (команда "escape"), в противном случае shell неверно ее проинтерпретирует и вы не получите ожидаемого результата.

ext2

("Extended 2 file system", **вторая расширенная файловая система**) родная файловая система GNU/Linux, обладающая всеми характеристиками любой файловой системы UNIX®: поддержка специальных файлов (символьных устройств, символических ссылок и т.д.), назначение разрешений и владельцев файлов и другие возможности.

FAQ

(*Frequently Asked Questions, часто задаваемые вопросы*) документ, содержащий ряд вопросов с ответами по определенной теме. Согласно истории, FAQ'и появились в тематических конференциях (newsgroup), то теперь документы такого типа имеются на различных веб-сайтах, и даже коммерческие продукты тоже имеют свои FAQ. В общем случае это очень хороший источник информации.

FAT

(*File Allocation Table, таблица размещения файлов*) файловая система, используемая в DOS и Windows®.

FDDI

Fiber Distributed Data Interface, распределенный интерфейс передачи данных по волоконно-оптическим каналам физический уровень для высокоскоростных сетей, использующий для передачи данных оптоволоконный кабель. Используется, преимущественно, в больших сетях, в основном из-за своей стоимости. Для подключения ПК к сетевым коммутаторам используется довольно редко.

FHS

(*File system Hierarchy Standard, стандарт иерархии файловой системы*) документ, содержащий рекомендации для организации логически последовательного дерева файлов в системах UNIX®. В большинстве аспектов Mandrakelinux совместим с этим стандартом.

FIFO

(*First In, First Out, "первым пришел - первым обслужен"*) структура данных или аппаратный буфер, из которых объекты выводятся в порядке их поступления. Наиболее общим примером FIFO являются каналы в UNIX®.

filesystem

файловая система; схема, используемая для хранения файлов в упорядоченном виде на физическом носителе (жесткий диск, дискета). Примеры файловых систем: FAT, ext2fs в GNU/Linux, ISO9660 (используемая CD-ROM-ами) и т.п. Пример виртуальной файловой системы - файловая система `/proc`.

firewall

файервол, брандмауэр, межсетевой экран защиты; машина или специальный аппаратный комплекс, который в топологии локальной сети является единственной точкой, подключенной к внешней сети, и который фильтрует и контролирует активность на некоторых портах или обеспечивает доступ к внешнему миру только некоторым особых интерфейсам.

flag

флаг; индикатор (обычно бит), используемый для уведомления программы о некотором состоянии. Например, у файловой системы, кроме всего прочего, есть флаг, отмечающий, было ли выполнено ее резервное копирование. Поэтому, если флаг активен, файловая система резервируется, а если он не активен - не резервируется.

focus

фокус; состояние окна, при котором оно принимает события от клавиатуры (такие как нажатия и отпускания клавиш и щелчки по клавишам мыши), если они не перехватываются оконным менеджером.

foreground

приоритетный режим; в контексте shell'а, процессом приоритетного режима является выполняемый в данный момент процесс и управляемый посредством клавиатуры и экрана. Вы должны дождаться завершения работы такого процесса, чтобы иметь возможность снова вводить команды.
См. также: job, background.

Frame Relay

ретрансляция кадров; сетевая технология, идеально подходящая для пропускания пульсирующего или случайного трафика. Стоимость сети уменьшается благодаря наличию большого количества абонентов Frame Relay, совместно использующих одну и ту же пропускную способность, и ретрансляции через них кадров переменной длины для того, чтобы использовать сеть в разные моменты времени.

framebuffer

видеобуфер, буфер кадров, фреймбуфер; проекция ОЗУ видеокарты на адресное пространство машины. Это позволяет приложениям обращаться к видеопамяти без необходимости работы непосредственно с самой картой. Все профессиональные графические рабочие станции используют видеобуфер.

FTP

(*File Transfer Protocol, протокол передачи файлов*) стандартный Интернет-протокол, используемый для пересылки файлов с одной машины на другую.

full-screen

полный экран; этот термин используется для приложений, захватывающих всю видимую область вашего монитора.

gateway

шлюз, гейт; машина или устройство, предоставляющее доступ к внешней сети из локальной сети.

GFDL

(*GNU Free Documentation License, GNU Лицензия свободной документации*) лицензия, действие которой распространяется на всю документацию Mandrakelinux.

GIF

(*Graphics Interchange Format, формат графического обмена*) формат графического файла, широко используемый в вебе. Изображения GIF могут быть сжатыми или анимированными. Из-за проблем с его авторским правом, их использование является не очень хорошей идеей, поэтому, по возможности, рекомендуется заменять их форматом PNG.
См. также: PNG.

globbing

(подстановка) в shell, это возможность группировать некоторый набор имен файлов по шаблонам подстановки.
См. также: globbing pattern.

globbing pattern

универсализация файловых имен, шаблон подстановки; строка, составленная из обычных и специальных символов. Специальные символы интерпретируются и расширяются shell'ом.

GNU

(*GNU's Not Unix, GNU - это не UNIX*) Проект GNU был основан Ричардом Столлменом (Richard Stallman) в начале 80-х годов. Целью этого проекта была разработка свободной операционной системы ("свободной" в смысле "свободы слова"). В настоящий момент для нее существуют все утилиты, кроме... ядра. Ядро проекта GNU - Hurd - еще не является полностью монолитным. Linux заимствует у GNU, кроме всего прочего, две вещи: его компилятор C - `gcc` - и его лицензию - GPL.
См. также: GPL.

GPL

(*General Public License, Общедоступная лицензия*) лицензия ядра GNU/Linux, она противопоставляется всем собственническим (проприетарным) лицензиям в том, что она не налагает таких ограничений, как копирование, изменение и дальнейшее распространение программного обеспечения, до тех пор, пока доступен исходный код. Единственным ограничением является то, что человек, которому вы передаете код, также должен получить те же самые права.

GUI

(*Graphical User Interface, графический интерфейс пользователя*) интерфейс к компьютеру, состоящий из окон с меню, кнопками, значками и т.п. Подавляющее большинство пользователей предпочитает использовать GUI вместо CLI (*Command Line Interface, интерфейс командной строки*), из-за простоты его использования, даже несмотря на то, что последний является более универсальным.

guru

гуру; эксперт. Используется для определения какого-либо чрезвычайно опытного высококвалифицированного специалиста, оказывающего также неоценимую помощь другим.

hardware address

аппаратный адрес; номер, однозначно идентифицирующий в физической сети хост на уровне сетевой архитектуры. Примеры - **адреса Ethernet** и **адреса AX.25**.

hidden file

скрытый файл; файл, который не может быть "увиден" при выполнении без параметров команды `ls`. Имена скрытых файлов начинаются с `.` и используются для хранения личных параметров и конфигураций пользователя к различным программам. Например, история команд bash хранится в скрытом файле `.bash_history`.

home directory

домашний каталог, "хомяк"; часто сокращается до "home", это название личного каталога данного пользователя.
См. также: account.

host

хост, сервер, узел; относится к компьютерам и используется обычно, когда речь идет о подключенных к сети компьютерах.

HTML

(*HyperText Markup Language, язык гипертекстовой разметки*) язык, используемый для создания веб-документов.

HTTP

(*HyperText Transfer Protocol, протокол передачи гипертекстовых файлов*) протокол, используемый для подключения к веб-сайтам и получения HTML-документов или файлов.

icon

значок, иконка, пиктограмма; маленькое изображение (обычно имеющее размер 16x16, 32x32, 48x48, а и иногда и 64x64 пикселей), которое является представлением документа, файла или программы в графической среде.

IDE

(*Integrated Drive Electronics, встроенный интерфейс накопителей*) наиболее широко используемая в современных ПК шина для жестких дисков. Шина IDE может содержать до двух устройств, а ее скорость ограничена скоростью устройства с более медленной очередью команд (но не более медленной скоростью передачи!).
См. также: ATAPI, SATA, S-ATA.

IMAP

(*Internet Message Access Protocol, протокол доступа к сообщениям в Интернете*) протокол, позволяющий вам получать доступ к своим электронным сообщениям на удаленном сервере без необходимости предварительной их загрузки их; противопоставляется протоколу получения почты POP.
См. также: POP.

inode

информационный узел, инод; точка входа, приводящая к содержимому файла в UNIX®-подобных файловых системах. Инод идентифицируется уникальным образом посредством числа и содержит такую мета-информацию о файле, на который он ссылается, как время последнего доступа, его тип, его размер, **но не его имя!**

insert mode

режим вставки; в Vi или в любом из его клонов, состояние программы, в котором нажатие на клавишу вставляет ее символ в редактируемый файл (за исключением особых случаев типа завершения аббревиатуры, выравнивания по правому краю в конце строки,..). Выход из него осуществляется по нажатию клавиши **Esc** (или **Ctrl-[**).

Internet

Интернет; огромная сеть, соединяющая компьютеры по всему миру.

IP-04@5A

числовой адрес, состоящий (в версии 4 также называемый IPv4) из четырех частей, который идентифицирует ваш компьютер в Интернете. IP-адреса имеют иерархическую структуру: верхний уровень и национальные домены, домены, поддомены и персональный адрес каждой машины. IP-адрес выглядит примерно так - `192.168.0.1`. Персональный адрес машины может быть статическим или динамическим. Статические IP-адреса присваиваются навсегда, т.е. они никогда не меняются. Использование динамических IP-адресов означает, что он будет меняться каждый раз при подключении к сети. Большинство домашних пользователей обычно имеют динамические IP-адреса, в то время как большинство корпоративных пользователей обычно имеют постоянные IP-адреса.

IP masquerading

IP-маскарадинг; метод, при котором файервол используется для того, чтобы скрыть настоящий IP-адрес вашего компьютера для внешнего мира. Зачастую любые подключения из внешней сети, устанавливаемые вами через файервол, будут наследовать его IP-адрес. Это полезно в случаях, если у вас есть быстрое подключение к Интернету только одним IP-адресом, но вы хотите задействовать несколько машин из своей внутренней локальной сети.

IRC

(*Internet Relay Chat, ретрансляция разговоров в Интернете, "ирка"*) один из нескольких Интернет-стандартов для живого общения. Он позволяет создавать каналы, вести частные беседы и обмениваться файлами. Он также позволяет серверам подключаться друг к другу, благодаря чему сегодня существует несколько сетей IRC: **Undernet**, **DALnet**, **EFnet** и другие.

IRC-:0=0;K

"места" внутри IRC-серверов, где можно пообщаться с другими людьми. Каналы создаются на IRC-серверах и пользователи могут присоединяться к ним, чтобы общаться друг с другом. Написанные в канал сообщения видны только людям, подключенным к этому каналу. Два или более пользователей могут создать "частный" канал, чтобы их не беспокоили другие пользователи. Имена каналов начинаются с #.

ISA

(*Industry Standard Architecture, архитектура шины промышленного стандарта*) самая первая шина, используемая в PC. ISA все еще встречается на SCSI-картах, поставляемых со сканерами, CD-рекордерами и некоторым другим старым оборудованием.

ISDN

(*Integrated Services Digital Network, цифровая сеть связи с комплекснъыми услугами*) набор стандартов связи для передачи голоса, видео и служб цифровых сетей. Был разработан для замены существующей системы телефонной связи, известной как PSTN (*Public Switched Telephone Network, телефонная коммутируемая сеть общего пользования*) или POTS (*Plain Old Telephone Service, простая старая телефонная служба*). ISDN известна как сеть передачи данных с коммутацией каналов.

ISO

(*International Standards Organization, международная организация по стандартизации*) группа компаний, консультантов, университетов и других источников, разрабатывающая стандарты в различных областях, включая вычислительную технику и связь. Документы, описывающие стандарты, пронумерованы. Например, стандарт с номером iso9660 описывает файловую систему, используемую на носителях CD-ROM.

ISO 8859

стандарт, добавляющий несколько 8-битных расширений к набору символов ASCII. Наиболее важным является ISO 8859-1 - "Latin Alphabet No. 1" (латинский алфавит номер 1), - который получил широкое распространение и уже может рассматриваться как замена *де факто* стандарта ASCII.
ISO 8859-1 поддерживает следующие языки: африкаанс, баскский, каталонский, датский, голландский, английский, фаррский, финский, французский, галицийский, немецкий, исландский, ирландский, итальянский, норвежский, португальский, шотландский, испанский и шведский.
Обратите внимание, что символы ISO 8859-1 также являются первыми 256 символами ISO 10646 (Unicode). Однако, в нем не хватает символа EURO и он не полностью поддерживает финский и французкий. ISO 8859-15 представляет собой модификацию от ISO 8859-1 для удаления этих недостатков.
См. также: ASCII, UTF-8.

ISP

(*Internet Service Provider, поставщик услуг Интернета*) компания, продающая своим клиентам доступ к Интернету по телефонным линиям или каналам с высокой пропускной способностью, таким как выделенные линии T-1, DSL или кабельные соединения.

JPEG

(*Joint Photographic Experts Group, объединенная группа экспертов в области фотографии*) еще один очень распространенный формат файлов изображений. JPEG больше всего подходит для сжатия фотографий реального мира и не очень для нереалистичных изображений.

job

задание; в контексте shell'а, задание - это процесс, выполняемый в фоновом режиме. У вас может быть несколько заданий в одном командном процессоре, и вы можете управлять ими независимо друг от друга.
См. также: foreground, background.

journaling

журналирование; увеличивает живучесть файловой системы, делая ее работу основанной на транзакциях. Таким образом, вместо физической записи данных в момент запроса, это действие заносится в журнал, а данные записываются "целиком" несколько позже, что также значительно повышает производительность и уменьшает время, необходимое для анализа и, если необходимо, для исправления файловой системы.

kernel

ядро; сердце операционной системы. Ядро отвечает за распределение ресурсов и отделение процессов друг от друга. Оно обрабатывает все низкоуровневые операции, что позволяет программам взаимодействовать непосредственно с оборудованием вашего компьютера, управляет буферным кэшем и так далее.

kill ring

в Emacs это набор текстовых областей, вырезанных или скопированных с момента запуска редактора. Текстовые области могут быть вызваны для повторной вставки, а вся структура напоминает кольцо.

LAN

(*Local Area Network, локальная вычислительная сеть*) собирательное название, под которым подразумевается сеть машин, подключенных к одной физической шине, в пределах небольшой географической территории типа офиса или здания.
См. также: WAN.

launch

запуск; действие, выполняющее вызов или старт программы.

library

библиотека; совокупность процедур и функций в бинарном виде, используемых программистами в своих программах (пока лицензия на библиотеку позволяет им это делать). Программа, отвечающая

за загрузку совместно используемых библиотек во время выполнения, называется динамическим компоновщиком.

link

ссылка, линк; ссылка на инод в каталоге, дающая таким образом имя (файла) для инода. Вот примеры инодов, не имеющих ссылок (а, следовательно, не имеющих имен): анонимные трубы (используемые командным процессором), сокеты (сетевые соединения), сетевые устройства и т.п.

linkage

компоновка, связывание, линковка; последний этап процесса компиляции, состоящий из связывания воедино всех объектных файлов с целью получения исполняемого файла, и поиск нераспознанных символов в динамических библиотеках (только если не была запрошена статическая компоновка, в случае которой код этих символов будет включен в исполняемый файл).

Linux

Линукс; UNIX®-подобная операционная система, работающая на самых разнообразных компьютерах, и являющая свободной для всех в плане использования или изменения. Ядро Linux было написано Линусом Торвальдсом (Linus Torvalds).

login

логин, регистрационное имя; имя пользователя в UNIX® для входа в систему, а также сам процесс подключения.

lookup table

таблица поиска, таблица соответствий; таблица, в которой хранятся коды (или теги) и соответствующие им значения. Зачастую это файл данных, используемый программами для получения дополнительной информации об определенном элементе.

Например, HardDrake использует такую таблицу для хранения кодов продукта производителя и соответствующей конфигурационной информации. Вот одна строка из таблицы, дающая информацию о продукте `CTL0001`

```
"CTL0001"       "sb"    "Creative Labs|SB16"    "sound" "HAS_OPL3|HAS_MPU401|HAS_DMA16|HAS_JOYSTICK"
```

loopback

обратная связь, возвратная петля, закольцовывание; виртуальный сетевой интерфейс машины, замкнутый сам на себя, и позволяющий выполняющимся программам не принимать во внимание особый случай, когда два сетевых объекта на самом деле являются одной и той же машиной.

major

старший, главный; специфический номер для класса устройств.

manual page

страница руководства; небольшой документ, содержащий описание команды и ее использование и вызываемый командой `man`. Это первое, что нужно прочитать при изучении команды, с которой вы не знакомы.

MBR

(*Master Boot Record, главная загрузочная запись*) название первого сектора загрузочного жесткого диска. MBR содержит код, используемый для загрузки в память операционной системы, или начальный загрузчик (например, LILO), а также таблицу разделов этого жесткого диска.

MIME

(*Multipurpose Internet Mail Extensions, многоцелевые расширения электронной почты в сети Интернет*) строка в виде `тип/подтип`, описывающая содержимое файла, прикрепленного к электронному письму. Это позволяет почтовым клиентам с поддержкой MIME определять действия, зависящие от типа вложенного файла.

minor

младший, второстепенный; номер, идентифицирующий определенное устройство, о котором идет речь.

MPEG

(*Moving Pictures Experts Group, Экспертная группа по вопросам движущегося изображения*) комитет ISO, устанавливающий стандарты для сжатия видео и аудио. MPEG - это также название алгоритмов

сжатия. К сожалению лицензия для этого формата весьма ограниченна и, как следствие, пока еще не существует MPEG-проигрывателей, полностью удовлетворяющих модели Open Source...

mount point

точка монтирования; каталог файловой системы GNU/Linux, к которому подключается раздел диска или другое устройство. Например, ваш CD-ROM монтируется в каталог `/mnt/cdrom`, в котором вы можете просмотреть содержимое любых примонтированных компакт-дисков.

mounted

примонтирован; устройство является примонтированным, если оно подключено к файловой системе GNU/Linux. Когда вы монтируете устройство, вы можете просмотреть его содержимое. Этот термин несколько устарел благодаря появлению возможности "супермонтирования" ("supermount"), которая позволяет пользователям монтировать съемные накопители автоматически, а не вручную.
См. также: mount point.

MSS

(*Maximum Segment Size, максимальный размер сегмента*) максимально возможный объем данных, который может быть передан интерфейсом за один раз. Если вы хотите предотвратить локальную фрагментацию, MSS должен быть равен размеру MTU заголовка IP.

MTU

(*Maximum Transmission Unit, максимальный передаваемый блок данных*) параметр, определяющий наибольший размер датаграммы, которая может быть передана IP-интерфейсом без необходимости разбиения ее на более мелкие блоки.MTU должен превышать размер самой большой датаграммы, которую вы хотите передать без фрагментации. Обратите внимание, что это предотвращает только локальную фрагментацию, некоторые другие узлы на пути передачи могут иметь меньший MTU и на них датаграмма будет фрагментирована. Типичные значения: 1500 байт для Ethernet-интерфейса или 576 байт для PPP-интерфейса.

multitasking

многозадачность; способность операционной системы распределять процессорное время между несколькими процессами. На низком уровне это также называется мультипрограммированием. Для переключения от одного процесса к другому требуется, чтобы все, что связано с текущим процессом, было сохранено, а затем восстановлено при возобновлении работы процесса. Эта операция называется контекстным переключением и выполняется она несколько раз в секунду. Это происходит так быстро, что пользователю кажется, будто операционная система выполняет несколько приложений одновременно. Существует два типа многозадачности: вытесняющая и кооперативная. В первом случае система отвечает за захват процессора и передачу его ресурсов другому процессу. При кооперативной (совместной) многозадачности процесс сам уступает процессор операционной системе. Первый вариант (используемый в GNU/Linux), очевидно, является лучшим выбором, потому что ни одна программа не может захватить все процессорное время и заблокировать другие процессы. Политика выбора выполняемого процесса зависит от нескольких параметров и называется распределением (машинного времени).

multiuser

многопользовательский; используется для описания операционной системы, которая позволяет входить в систему нескольким пользователям одновременно, каждый из которых может работать независимо от других пользователей. Для обеспечения многопользовательской поддержки требуется многозадачная операционная система. В этом отношении GNU/Linux, как и любая другая UNIX®, является одновременно и многозадачной, и многопользовательской операционной системой.

named pipe

именованный канал; канал UNIX®, на который имеется ссылка, в отличие от каналов, используемых в командный процессорах.
См. также: pipe, link.

naming

именование, присваивание имен; слово, обычно используемое в вычислительной технике по отношению к способу идентификации объектов. Вы часто будете слышать о "соглашениях о присваивании имен" для файлов, функций в программах и т.п.

NCP

(*NetWare Core Protocol, основной протокол NetWare*) протокол, определенный **Novell**'ом для доступа к файловым службам и службам печати Novell NetWare.

NFS

(*Network FileSystem, сетевая файловая система*) файловая система, созданная в **Sun Microsystems**, для прозрачного совместного использования файлов по сети.

newsgroups

группы новостей, сетевые конференции; места для обсуждений и новостей, доступ к которым можно получить при помощи клиента новостей или USENET, чтобы можно было читать или писать сообщения с той или иной обсуждаемой темой. Например, конференция `alt.os.linux.mandrake` является альтернативной конференцией (alt), посвященной операционной системе (os) GNU/Linux (linux), а в частности - Mandrakelinux (mandrake). Конференции разбиваются таким образом для того, чтобы упростить поиск определенной темы.

NIC

(*Network Interface Card, сетевая интерфейсная карта*) вставленный в компьютер адаптер, который обеспечивает физическое подключение к сети, такой как Ethernet-карта.

NIS

(*Network Information System, сетевая информационная система*) NIS также известна как "Желтые страницы" ("Yellow Pages"), но авторское право на это имя принадлежит **British Telecom**. NIS - это протокол, разработанный в **Sun Microsystems** для совместного использования общей информации в **домене** NIS, который может охватывать всю локальную сеть или только ее часть. Он может экспортировать базы данных с паролями, базы данных служб, информацию о группах и другое.

null, character

пустой символ, знак пробела; символ или байт под номером 0, используемый для отметки конца строки.

object code

объектный код; машинный код, генерируемый в процессе компиляции, компонуемый с другими объектными кодами и библиотеками для формирования исполняемого файла. Объектный код предназначен для чтения машиной.
См. также: compilation, linkage.

on the fly

налету; что-либо делается "налету", если это делается наряду с чем-то другим, не уведомляя вас об этом или без явного запроса.

open source

открытый исходный код, опенсорс; имя, данное открытому исходному коду программы, которая была сделана доступной для сообщества разработчиков и широкой публики. Лежащая в основе этого теория, заключается в том, что разрешение на использование и изменение исходного кода широким кругом программистов в результате приведет к созданию более полезного для всех продукта. Некоторые популярные программы с открытым исходным кодом: Apache, sendmail и GNU/Linux.

operating system

операционная система; интерфейс между приложениями и оборудованием, на котором они работают. Первостепенной задачей для любой операционной системы является управление всеми ресурсами машины. В системе GNU/Linux за это отвечает ядро и загружаемые модули. Другие хорошо известные операционные системы: Amiga®OS, Mac OS®, FreeBSD®, OS/2®, UNIX®, Windows NT® и все их варианты.

owner

владелец; в контексте пользователей и их файлов, владельцем файла является пользователь, создавший этот файл.

owner group

группа-владелец, владелец группы; в контексте групп и их файлов, группой, владеющей файлом, является группа, к которой принадлежит пользователь, создавший этот файл.

PAP

(*Password Authentication Protocol, протокол аутентификации паролей*) протокол, используемый многими Интернет-провайдерами для аутентификации (не путать с авторизацией!) своих клиентов. Согласно этой схеме клиент (вы) отправляет серверу пару идентификатор/пароль, но ни бита

информации при этом не шифруется. CHAP является более безопасным (а значит более предпочтительным) протоколом аутентификации.
См. также: CHAP.

pager
пейджер; программа, показывающая текстовый файл по одному экрану за раз, упрощая тем самым перемещение вперед и назад и поиск строк в этом файле. Мы рекомендуем вам использовать `less`.

password
пароль; секретное слово или комбинация слов или букв, используемая для защиты чего-либо. Пароли используются при входе пользователей в многопользовательские операционные системы, на веб-сайты, FTP-сайты и т.д. Пароли должны быть трудными для угадывания фразами или комбинациями букв и цифр и никогда не должны основываться на общих словах из словарей. Пароли гарантируют, что другие люди не смогут войти в компьютер или сайт с вашей учетной записью.

patch, to patch
патч, патчить; файл, содержащий список исправлений к исходному коду для добавления новых возможностей, удаления ошибок или изменения его согласно каким-нибудь требованиям или пожеланиям. Действие состоит в применении этих исправлений к архиву с исходным кодом (aka "патчинг").

path
путь; привязка файлов и каталогов к файловой системе. Отдельные элементы пути разделяются "слэшем" (символ '/'). В системах GNU/Linux существует два типа путей. **Относительный** путь - это местоположение файла или каталога относительно текущего каталога. **Абсолютный** (или **полный**) путь - это местоположение файла или каталога относительно корневого каталога.

PCI
(*Peripheral Components Interconnect, локальная шина соединения периферийных устройств*) шина, созданная в **Intel**, и которая на сегодня является стандартной шиной для PC и других архитектур. Она является потомком ISA-шины и предоставляет множество услуг: идентификацию устройства, информацию о конфигурации, совместное использование IRQ, захват шины и многое другое.

PCMCIA
(*Personal Computer Memory Card International Association, Международная ассоциация производителей карт памяти для персональных компьютеров*) все чаще и чаще называемый просто "PC Card", это стандарт для внешних карт, подключаемых к портативным компьютерам: модемы, жесткие диски, карты памяти, Ethernet-карты и др. Эту аббревиатуру иногда в шутку расшифровывают как *People Cannot Memorize Computer Industry Acronyms* (люди не могут запомнить акронимы компьютерной промышленности)...

pipe
канал, труба, конвейер, пайп; специальный тип файла UNIX®. Одна программа пишет данные в канал, а другая программа читает данные на другом конце. Каналы в UNIX® работают согласно принципу FIFO, т.е. данные на другом конце считываются в том порядке, в каком они были отправлены. Очень широко используются в командном процессоре. Смотрите также **именованный канал**.

pixmap
(*pixel map, карта пикселей*) еще одно название побитового изображения.

plugin
подключаемый модуль, плагин; дополнительная программа, используемая для отображения или воспроизведения некоторого мультимедийного контента, найденного в веб-документе. Обычно она легко может быть загружена, если ваш браузер еще не в состоянии отобразить или воспроизвести такой тип информации.

PNG
Portable Network Graphics (переносимая сетевая графика). Формат файлов изображений, созданный преимущественно для использования в вебе. Он был разработан в качестве замены GIF, свободной от патентов, и обладает несколькими дополнительными возможностями.

PnP
(*Plug'N'Play, "подключи и работай"*) первое расширение для ISA, добавляющее к устройствам конфигурационную информацию. Этот термин стал более распространенным, охватывая все устройства, способные сообщить о параметрах своей конфигурации. Все PCI-устройства относятся к Plug'N'Play.

POP

(*Post Office Protocol, протокол почтового офиса*) один общий протокол, используемый для получения почты от ISP. Смотрите IMAP как пример другого почтового протокола удаленного доступа.
См. также: IMAP.

porting

портирование, перенос; один из двух способов для запуска программы в системе, для которой она не была изначально предназначена. Например, чтобы запустить программу из Windows® в GNU/Linux (естественным образом), сначала она должна быть портирована в GNU/Linux.

PPP

(*Point to Point Protocol, протокол соединения "точка-точка"*) протокол, используемый для отправки данных по последовательным линиям. Обычно он используется для отправки IP-пакетов в Интернет, но также может быть использован вместе с другими протоколами, такими как Novell'овский протокол IPX.

precedence

старшинство, приоритет; задает порядок вычисления операндов в выражении. Например: результатом выражения `4 + 3 * 2` будет 10, т.к. умножение имеет более высокий приоритет, чем сложение. Если вы хотите вычислить сначала сумму, вам нужно добавить скобки: `(4 + 3) * 2`. В этом случае результатом будет 14, т.к. скобки имеют более высокий приоритет, чем сложение или умножение, поэтому сначала вычисляются операции в скобках.

preprocessors

препроцессор; директив компиляции, приказывающие компилятору заменить эти директивы кодом языка программирования, используемого в исходном файле. Примеры препроцессоров для C: `#include`, `#define` и т.д.

process

процесс; в контексте операционной системы, процесс - это выполняющаяся копия программы вместе со своим окружением.

prompt

приглашение, подсказка; в контексте shell'а, это строка перед курсором. Когда вы его видите, вы можете вводить свои команды.

protocol

протокол; протоколы организуют обмен данными между различными машинами в сети при помощи аппаратного или программного обеспечения. Они определяют формат передаваемых данных, какая из машин управляет другой и т.п. Наиболее известные протоколы: HTTP, FTP, TCP и UDP.

proxy

прокси; машина, находящаяся между сетью и Интернетом, чья задача состоит в ускорении передачи данных по наиболее широко используемым протоколам (например, HTTP и FTP). Она содержит кэш из предыдущих запросов, поэтому машина, запрашивающая что-либо, что уже было закэшировано, быстро получит ответ, потому что эта информация находится в локальном кэше. Прокси очень полезны в сетях с низкой пропускной способностью (таких как модемные соединения). Иногда прокси является единственной машиной, которая имеет доступ к внешней сети.

pulldown menu

ниспадающее меню; меню, которое может быть "развернуто" кнопкой в одном из углов. Когда вы нажимаете на эту кнопку, меню "выезжает" из своего заголовка, показывая вам полное меню.

quota

квота; метод для ограничения использования дискового пространства и установки лимитов для пользователей. Администраторы могут ограничить размер домашних каталогов пользователей, установив предельные квоты на отдельные файловые системы.

RAID

(*Redundant Array of Independent Disks, матрица независимых дисковых накопителей с избыточностью*) проект, начало которому было положено на факультете вычислительной техники Калифорнийского университета в Беркли, США. В нем хранимые данные распределяются по дисковому массиву согласно различным схемам. Вначале он был реализован с использованием дешевых старых дисков, откуда изначально и произошла аббревиатура *Redundant Array of Inexpensive Disks (матрица недорогих дисковых накопителей с избыточностью)*.

RAM

(*Random Access Memory, оперативная память*) термин, используемый для обозначения главной памяти компьютера. "Random" ("случайный") в данном случае означает, что к любой части памяти может быть получен непосредственный доступ.

read-only mode

режим "только для чтения"; для файла это означает, что в него не может быть выполнена запись. Вы можете прочитать содержимое файла, но не можете его изменить.
См. также: read-write mode.

read-write mode

режим "чтения-записи"; для файла это означает, что в него может быть выполнена запись. Вы можете прочитать содержимое файла и изменить его.
См. также: read-only mode.

regular expression

регулярное выражение; мощный абстрактный инструмент, используемый для поиска и сопоставления текстовых строк. Он определяет шаблоны, которым должны удовлетворять эти строки. Многие утилиты UNIX® используют регулярные выражения: `sed`, awk, grep, perl и другие.

RFC

(*Request For Comments, запрос на комментарии*) документы официальных Интернет-стандартов, опубликованные IETF'ом (*Internet Engineering Task Force, Целевая группа инженерной поддержки Интернет*). Они описывают все протоколы, их использование, требования и так далее. Если вы хотите узнать, как работает протокол, обратитесь к соответствующему RFC.

root

рут; суперпользователь любой системы UNIX®. Обычно рутом (aka системным администратором) является человек, отвечающий за обслуживание и управление системой UNIX®. Также этот человек имеет полный доступ ко всему, что есть в системе.

root directory

корневой каталог; каталог верхнего уровня файловой системы. У него нет родительского каталога, поэтому для корневого каталога '..' указывает на него самого. Корневой каталог обозначается как '/'.

root filesystem

корневая файловая система; файловая система верхнего уровня, в которую GNU/Linux монтирует дерево своего корневого каталога. Для корневой файловой системы необходимо, чтобы она находилась на своем собственном разделе. На ней находится корневой каталог.

route

маршрут; путь, который ваши датаграммы проходят по сети, чтобы достичь своего назначения. Это путь между двумя машинами сети.

RPM

(*RPM Package Manager, менеджер пакетов RPM*) формат упаковки, разработанный в **Red Hat** для создания пакетов программного обеспечения. Он используется во многих дистрибутивах GNU/Linux, в том числе и в Mandrakelinux.

run level

уровень запуска; конфигурация системного программного обеспечения, позволяющая существовать только выбранным процессам. Разрешенные файлы для каждого уровня запуска определены в файле `/etc/inittab`. Обычно существует семь предопределенных уровней запуска: 0, 1, 2, 3, 4, 5, 6 - а переключение между ними может выполняться только привилегированным пользователем при помощи команд `init` и `telinit`.

SATA, S-ATA

(*Serial ATA*) наследник спецификации ATA. первое поколение SATA имело пропускную способность в 1.5 Гбит/с, но последовательный канал связи и положенная в основу технология позволяют значительно увеличить пропускную способность, в то время как ATA достигла своего практического предела на UDMA133.
См. также: ATAPI, IDE.

script

скрипт, сценарий; последовательность выполняемых команд, как если бы они последовательно вводились в консоли. Shell-скрипты UNIX® являются (неким) эквивалентом пакетных файлов DOS.

SCSI

(*Small Computers System Interface, интерфейс малых вычислительных систем*) шина с высокой пропускной способностью, разработанная для возможности подключения к ней периферийных устройств различных типов. В отличии от IDE скорость SCSI-шины не ограничивается скоростью приема команд периферийными устройствами. Только в машинах "high-end"-класса SCSI-шина интегрирована непосредственно в материнскую плату, поэтому для большинства PC требуются дополнительные карты.

security levels

уровни безопасности; уникальная функция Mandrakelinux, позволяющая вам установить различные уровни ограничений в зависимости от того, в какой степени вы хотите обезопасить свою систему. Существует 6 предопределенных уровней в диапазоне от 0 до 5, где 5-й уровень является самым защищенным. Вы также можете определить свои собственные уровни безопасности.

segmentation fault

нарушение сегментации; исключительная ситуация, возникающая при попытке программы обратиться к памяти, выделенной не для нее. Обычно это приводит к немедленной остановке программы.

server

сервер; программа или компьютер, предоставляющие услуги или возможности и ожидающие подключений от **клиентов** для выполнения их команд или предоставления запрошенной ими информации. В случае систем с **одноранговой связью**, таких как SLIP или PPP, в качестве сервера выступает вызываемая сторона, а вызывающая сторона выступает в качестве клиента. Это одна из составляющих **системы клиент-сервер**.
См. также: client, client/server system.

shadow passwords

теневые пароли; программный комплекс для управления паролями в системах UNIX®, в котором файл с зашифрованными паролями нельзя прочитать "из мира", в отличие от обычной системы паролей, где это возможно. Также она предлагает другие возможности, например, срок действия пароля.

shell

командный процессор, оболочка, шелл; базовый интерфейс к ядру операционной системы, предоставляющий пользователям командную строку, где они могут вводить системные команды и команды для запуска программ. Все шеллы предоставляют скриптовые языки, которые могут быть использованы для автоматизирования задач или упрощения часто используемых сложных задач. Эти shell-скрипты похожи на пакетные файлы из операционной системы DOS, но они намного мощнее. Примеры командных процессоров: bash, sh и tcsh.

single user

однопользовательский; используется для описания состояния операционной системы или даже самой операционной системы, которое позволяет входить в систему и работать в ней только одному пользователю.

site dependent

зависимый от местонахождения; означает, что информация, используемая такими программами, как imake и `make` для компиляции некоторых файлов с исходным кодом, зависит от местонахождения, архитектуры компьютера, установленных на нем библиотек и т.п.

SMB

(*Server Message Block, блок сообщений сервера*) протокол, используемый Windows®-машинами для совместного использования по сети файлов и принтеров.
См. также: CIFS.

SMTP

(*Simple Mail Transfer Protocol, простой протокол пересылки почты*) общий протокол для передачи электронных писем. SMTP используют агенты пересылки сообщений (MTA), такие как sendmail или postfix. Иногда их называют SMTP-серверами.

socket
сокет; тип файла, соответствующий любому сетевому подключению.

soft links
См.: symbolic links

standard error
стандартный поток ошибок; файловый дескриптор номер 2, открываемый любым процессом; условно принят для использования в качестве файлового дескриптора, в который процесс выводит сообщения об ошибках. Обычно это экран компьютера.
См. также: standard input, standard output.

standard input
стандартный ввод; файловый дескриптор номер 0, открываемый любым процессом; условно принят для использования в качестве файлового дескриптора, из которого процесс получает данные.
См. также: standard error, standard output.

standard output
стандартный вывод; файловый дескриптор номер 1, открываемый любым процессом; условно принят для использования в качестве файлового дескриптора, в который процесс выводит свои выходные данные. Обычно это экран компьютера.
См. также: standard error, standard input.

streamer
стример; устройство, работающее с "потоками" символов (непрерывными или не разбитыми на более короткие части) в качестве входных данных. Типичный пример стримера - накопитель на магнитной ленте.

SVGA
((*Super Video Graphics Array, матрица супер видеографики*) стандарт видеодисплея, определенный VESA для PC-архитектуры. Первое разрешение было 800x600x16 цветов, затем оно быстро увеличилось до 1024x768x16 цветов и выше.

switch
ключ, опция; ключи используются для управления поведением программ и также называются опциями или аргументами командной строки. Чтобы определить, имеет ли программа опциональные ключи, прочтите `man`-страницы или попробуйте запустить ее с ключом `--help` (т.е. `program --help`).

symbolic links
символические ссылки; специальные файлы, не содержащие ничего, кроме ссылки на другой файл. Любое обращение к ним означает обращение к файлу, чье имя является текстом ссылки. Этот файл может существовать или не существовать, и путь к нему может быть указан в абсолютном или относительном виде.

target
цель; объект компиляции, т.е. бинарный файл, который будет создан компилятором.

TCP
(*Transmission Control Protocol, протокол управления передачей*) наиболее общий надежный протокол, использующий интернет-протокол (IP) для передачи сетевых пакетов. TCP добавляет необходимые проверки поверх IP, чтобы гарантировать доставку пакетов. В отличие от UDP он работает в режиме подключения, означающем, что две машины перед обменом данными должны установить соединение.

telnet
telnet, телнет; создает подключение к удаленному хосту и позволяет вам войти в систему, на которой у вас есть учетная запись. Телнет - это наиболее широко используемый метод для удаленного входа в систему, однако существуют улучшенные и более защищенные альтернативы, такие как `ssh`. Основной недостаток телнета - незашифрованный трафик (прим. переводчика).

theme-able
поддерживающий темы; графическое приложение поддерживает темы, если оно может изменить свой внешний вид в реальном времени. Многие оконные менеджеры поддерживают использование тем.

TLDP

(*The Linux Documentation Project, Проект документации по Linux*) некоммерческая организация, обеспечивающая поддержку документации по GNU/Linux. Она известна в основном благодаря своим документам HOWTO, но она также выпускает FAQ'и и даже некоторые книги.
См. также: FAQ.

traverse

прохождение; для каталога в системе UNIX® это означает, что пользователю разрешено проходить через этот каталог, а возможно и через его подкаталоги. Для этого необходимо, чтобы пользователь имел разрешение на выполнение для этого каталога.

URL

(*Uniform Resource Locator, унифицированный указатель информационного ресурса*) строка специального формата, используемая для уникальной идентификации ресурса в Интернете. Ресурс может быть файлом, сервером или чем-то другим. Синтаксис URL:
`протокол://пользователь:пароль@имя.сервера[:порт]/путь/к/ресурсу`.
Если указано только имя машины и протокол `http://`, загружается файл, настроенный на сервере на получение по умолчанию, обычно это файл `index.html`.

username

имя пользователя; имя (или в более общем виде - слово), идентифицирующее пользователя в системе. Каждое имя пользователя закрепляется за уникальным и единственным UID'ом (идентификатором пользователя).
См. также: login.

UTF-8

(*Unicode Transformation Format 8; преобразование Unicode, формат 8*) восьмеричное (8-битное) кодирование без потерь символов Unicode. UTF-8 кодирует каждый символ Unicode как переменное число длиной от 1 до 4 октетов, где число октетов зависит от целого значения, присвоенного символу Unicode. Это эффективное кодирование для документов Unicode, использующих в основном символы US-ASCII, потому что оно представляет каждый символ в диапазоне от U+0000 до U+007F в виде одного октета. UTF-8 - кодировка по умолчанию для XML.
См. также: ISO 8859, ASCII.

variables

переменные; строки, используемые в файлах `Makefile` с целью их замены на соответствующие им значения. Обычно они находятся в начале файла `Makefile`. Используются они для упрощения самого `Makefile` и облегчения управления деревом файлов с исходными текстами.
Более обобщенно, переменные в программировании - это слова, ссылающиеся на другие элементы (числа, строки, таблицы и т.д.), которые скорее всего будут изменяться во время работы программы.

verbose

подробный, многословный; для команд подробный режим означает, что команда сообщает на стандартный вывод (или, возможно, на стандартный поток ошибок) обо всех выполняемых действиях и результаты этих действий. Иногда у команд есть способ для определения "уровня подробности", означающего, что объем выводимой командой информации может быть контролируемым.

VESA

(*Video Electronics Standards Association, Ассоциация по стандартизации в области видеотехники и микроэлектроники*) Ассоциация по промышленным стандартам для архитектуры PC. Например, она является автором стандарта SVGA.

virtual console

виртуальная консоль; то, что раньше называлось терминалом. В системах GNU/Linux у вас есть так называемые виртуальные консоли, позволяющие вам использовать один экран или монитор для нескольких независимых работающих сеансов. По умолчание у вас есть шесть виртуальных консолей, перейти в которые можно при помощи клавиш **ALT-F1** до **ALT-F6**. Есть еще седьмая виртуальная консоль - **ALT-F7**, разрешающая попасть вам в X Window System. Из X перейти в текстовые консоли можно при помощи комбинаций **CTRL-ALT-F1** до **CTRL-ALT-F6**.
См. также: console.

virtual desktops

виртуальные рабочие столы; В X Window System оконный менеджер может предоставить вам несколько рабочих столов. Эта удобная функция позволяет вам организовать свои окна, избегая

проблем с наложением множества окон поверх друг друга. Это работает так, как если бы у вас было несколько разных экранов. Вы можете переключаться из одного виртуального рабочего стола в другой способом, зависящем от используемого вами оконного менеджера.
См. также: window manager, desktop.

WAN
(*Wide Area Network, глобальная сеть*) эта сеть, хоть и похожая на LAN, соединяет компьютеры в сеть, которые физически не подключены к одной физической шине и разнесены на большие расстояния.
См. также: LAN.

wildcard
знак подстановки, символ обобщения; символы '*' и '?' используются как знаки подстановки и могут означать что угодно. Символ '*' означает любое количество символов, включая отсутствие символов. Символ '?' означает только один символ. Знаки подстановки часто используются в регулярных выражениях.

window
окно; в сетях **окно** - это наибольший объем данных, допускаемый принимающей стороной в заданной точке за раз.
В контексте графической среды пользователя, окно - это прямоугольник, в котором выполняется определенное приложение, который обычно содержит заголовок, меню, строку состояния и рабочую область приложения.

window manager
оконный менеджер; программа, отвечающая за "внешний вид" графической среды, работающая с панелями окон, рамками, кнопками, главными меню и некоторыми комбинациями клавиш быстрого вызова. Без оконного менеджера было бы трудно или вообще невозможно работать с виртуальными рабочими столами, изменять размеры окон, перемещать их...

workspace switcher
переключатель рабочих областей; небольшой апплет, позволяющий вам переключаться между доступными виртуальными рабочими столами. Также известен как пейджер.
См. также: virtual desktops.

Предметный указатель