

Справочное руководство

Limited Edition 2005



<http://www.mandriva.com>

Справочное руководство: Limited Edition 2005

Опубликовано Апрель 2005

Copyright © 2005 Mandriva

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, Aurelio Marinho Jargas, Felipe Arruda, Marcia Gawlak Hoshi, Bob Rye, Jean-Luc Borie, Roberto Patriarca

Юридическое замечание

Данное руководство может распространяться только согласно условиям, изложенным в Open Publication License, версии 1.0 или более поздней (последняя версия постоянно доступна на сайте [opencontent.org](http://www.opencontent.org/openpub/) (<http://www.opencontent.org/openpub/>)).

- Запрещается распространять существенно изменённые версии этого документа без явно выраженного разрешения владельца авторских прав.
- Запрещается распространять этот материал или его производных в виде (бумажных) книг любого стандартного формата без предварительного разрешения, полученного от владельца авторских прав.

Авторское право на русский перевод принадлежит Alice Lafox (<mailto:alice@lafox.net>) и Павлу Марьянову (<mailto:jack@lafox.net>). При воспроизведении, дублировании или распространении этого перевода ссылка на авторов перевода обязательна.

Как создавалось это руководство

Это руководство было написано и поддерживается компанией NeoDoc (<http://www.neodoc.biz>). Переводы выполнены компанией NeoDoc, Mandriva и другими переводчиками.

Этот документ написан в формате DocBook XML. 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` и `jadetex` (для электронной версии) с использованием собственной версии таблиц стилей Нормана Уолша (Norman Walsh). Снимки экрана были 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 Mandriva Linux.

Содержание

Предисловие	1
1. Информация о Mandriva Linux	1
1.1. Связь с сообществом Mandriva Linux	1
1.2. Вступайте в Клуб!	1
1.3. Подписка на Mandriva Online	2
1.4. Приобретение продуктов Mandriva	2
1.5. Содействие Mandriva Linux	2
2. Об этом Справочном руководстве	2
3. Примечание редактора	3
4. Соглашения, используемые в этой книге	4
4.1. Соглашения по набору текста	4
4.2. Основные соглашения	5
I. Система Linux	7
1. Основные понятия системы UNIX	7
1.1. Пользователи и группы	7
1.2. Основы работы с файлами	9
1.3. Процессы	11
1.4. Краткое введение в командную строку	11
2. Диски и разделы	17
2.1. Структура жёсткого диска	17
2.2. Соглашения при именовании дисков и разделов	19
3. Организация дерева файлов	21
3.1. Разделяемые/неразделяемые, статические/переменные данные	21
3.2. Корневой каталог: /	21
3.3. /usr: просто Большой каталог	22
3.4. /var: Изменяемые при использовании данные	23
3.5. /etc: Конфигурационные файлы	23
4. Файловая система Linux	25
4.1. Сравнение нескольких файловых систем	25
4.2. Всё является файлом	27
4.3. Ссылки	29
4.4. “Анонимные” каналы и именованные каналы	30
4.5. Специальные файлы: файлы символьного и блочного режима	31
4.6. Символические ссылки. Ограничения “жестких” ссылок	32
4.7. Атрибуты файлов	33
5. Файловая система /proc	35
5.1. Информация о процессах	35
5.2. Информация об аппаратном обеспечении	36
5.3. Отображение и изменение параметров ядра	40
II. Linux изнутри	41
6. Файловые системы и точки монтирования	41
6.1. Принципы	41
6.2. Разметка жесткого диска, форматирование разделов	43
6.3. Команды mount и umount	43
7. Введение в командную строку	47
7.1. Утилиты обработки файлов	47
7.2. Управление атрибутами файлов	49
7.3. Шаблоны подстановки в командном процессоре	51
7.4. Перенаправления и каналы	51
7.5. Завершение командной строки	53
7.6. Запуск и обработка фоновых процессов: управление заданиями	54
7.7. Заключительное слово	55
8. Редактирование текста: Emacs и VI	57
8.1. Emacs	57
8.2. Vi: предок	60
8.3. Заключительное слово	64
9. Утилиты командной строки	65
9.1. Операции с файлами и фильтрация	65

9.2. find : Поиск файлов по определенным критериям	70
9.3. Запуск команд по расписанию	72
9.4. Архивирование и упаковка данных	74
9.5. Больше, гораздо больше.....	76
10. Управление процессами	77
10.1. Подробнее о процессах	77
10.2. Информация о процессах: ps и ps tree	77
10.3. Отправка сигналов процессам: kill , killall и top	78
10.4. Установка приоритетов для процессов: nice , renice	79
11. Загрузочные файлы: init sysv	81
11.1. В начале был init	81
11.2. Уровни выполнения	81
12. Безопасный удалённый доступ	85
12.1. Настройка сервера SSH	85
12.2. Настройка клиента SSH	85
12.3. Обмен файлами между локальной и удалённой системами	86
13. Управление пакетами из командной строки	87
13.1. Установка и удаление пакетов	87
13.2. Управление источниками	87
13.3. Советы и подсказки	88
А. Глоссарий	91
Предметный указатель	111

Список таблиц

4-1. Характеристики файловой системы.....	26
---	----

Предисловие

1. Информация о Mandriva Linux

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



7^{го} апреля 2005 года компания Mandrakesoft сменила своё имя на **Mandriva**, чтобы отразить своё слияние с компанией Conectiva из Бразилии. Основной продукт **Mandriva** - Mandrake-Linux - получил название Mandriva Linux.

1.1. Связь с сообществом Mandriva Linux

Следующие разнообразные Интернет-ссылки указывают на различные ресурсы, связанные с Mandriva Linux. Если вы хотите побольше узнать о компании **Mandriva**, зайдите на наш веб-сайт (<http://www.mandriva.com/>). Вы также можете заглянуть на веб-сайт дистрибутива Mandriva Linux (<http://www.mandrivalinux.com/>) и всего, что к нему относится.

Mandriva Expert (<http://www.mandrivaexpert.com/>) - это платформа технической поддержки от **Mandriva**. Она предлагает поделиться опытом, основываясь на доверии и вознаграждении других за их содействие.

Мы также приглашаем вас подписаться на различные списки рассылок (<http://www.mandriva.com/en/community/resources/newsgroups>), в которых сообщество Mandriva Linux демонстрирует свою активность и проницательность.

Не забудьте также заглянуть на нашу страницу безопасности (<http://www.mandriva.com/security>). На ней собраны все материалы, касающиеся безопасности дистрибутивов Mandriva Linux. 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 Mandriva Online (<https://online.mandriva.com>). Этот ресурс содержит всё, что необходимо знать любому системному администратору или пользователю, заинтересованному в обеспечении безопасности своей системы.

1.2. Вступайте в Клуб!

Mandriva предлагает большое разнообразие привилегий через свой Mandriva Club (<http://club.mandriva.com>):

- загрузка коммерческого программного обеспечения, обычно доступного только в коробочных версиях, такого как драйверы 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 Mandriva Linux;
- получение скидок на продукты и услуги в Mandriva Store (<http://store.mandriva.com>);
- доступ к эксклюзивному списку зеркал, доступных только для членов Клуба;
- чтение форумов и статей на нескольких языках.
- доступ к Базе знаний (<https://club.mandriva.com/xwiki/bin/view/KB>) **Mandriva** - wiki-сайту, содержащему документацию на разнообразные темы: администрирование, взаимодействие, поиск и устранение неисправностей и другие;
- чат с разработчиками Mandriva Linux в Club Chat (<https://www.mandrivaclub.com/user.php?op=clubchat>);
- повышение своих знаний GNU/Linux с помощью курсов электронного обучения от **Mandriva** (<http://etraining.mandriva.com>)

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

1.3. Подписка на Mandriva Online

Mandriva предлагает очень удобный способ для поддержания вашей системы в актуальном состоянии в автоматическом режиме, оберегая её от ошибок и уязвимостей в безопасности. Посетите веб-сайт **Mandriva Online** (<https://online.mandriva.net/>), чтобы больше узнать об этой услуге.

1.4. Приобретение продуктов Mandriva

Пользователи **Mandriva Linux** могут приобрести продукты в нашем онлайн-интернет-магазине **Mandriva Store** (<http://store.mandriva.com/>). В нём вы найдете не только программное обеспечение **Mandriva Linux**, 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 **Mandriva**.

1.5. Содействие Mandriva Linux

Опыт и знания многих талантливых людей, использующих **Mandriva Linux**, могут очень пригодиться при создании системы **Mandriva Linux**:

- **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 **Mandriva**’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://qa.mandriva.com/>), чтобы больше узнать о том, как вы можете поспособствовать развитию **Mandriva Linux**.

2. Об этом Справочном руководстве

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

- В первой части (*Система Linux*) мы познакомим вас с системой GNU/Linux. Рассмотрим её архитектуру, доступные файловые системы и некоторые более специфические аспекты наподобие файловой системы **/proc**.

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.

В главе Гл. 3 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.

Следующая глава посвящена файловым системам GNU/Linux (Гл. 4). После представления доступных файловых систем, мы рассмотрим типы файлов и некоторые дополнительные понятия и утилиты типа **inode** и **pipe**. В следующей главе (Гл. 5) представлена специальная (и виртуальная) файловая система GNU/Linux - **/proc**.

- Во второй части (*Linux изнутри*) рассматриваются более практичные темы. Мы поговорим о взаимосвязи между файловыми системами и точками монтирования, узнаем как использовать командную строку в повседневных задачах, как редактировать конфигурационные файлы с помощью лёгких и мощных редакторов и др.

Мы рассмотрим **файловые системы** и **точки монтирования** (Гл. 6), дав определения обоим этим понятиям, а также поясним их на практических примерах.

Затем мы примемся за командную строку и её интерфейс (Гл. 7). Рассмотрим утилиты для работы с файлами (наподобие `mkdir` и `touch`), как перемещать, удалять и копировать файлы и каталоги в пределах файловой системы. Также мы поговорим об атрибутах файлов и как работать с ними с помощью таких команд, как `chown` и `chgrp`. Затем мы изучим шаблоны подстановки, перенаправления и каналы командного процессора, а также завершение и основы управления заданиями.

В следующей главе рассматривается редактирование текста (Гл. 8). Так как большинство конфигурационных файлов UNIX® представляют 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** (автор Ричард Столлман) 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. 9) и в общих чертах управление процессами (Гл. 10).

Следующая глава (Гл. 11) рассказывает о процедуре загрузки **Mandriva Linux** и о том, как её эффективно использовать. Мы поговорим об `init` (процесс, благодаря которому происходит загрузка вашей системы) и различных уровнях запуска, которые вы можете использовать (особенно для задач обслуживания системы). Также мы кратко объясним, как использовать **drakxservices** для управления службами.

В следующей главе (Гл. 12) мы объясним, как безопасно подключаться к удалённой системе (с помощью `ssh`) для выполнения задач обслуживания, запуска на ней программ и т.п. Мы представим вам краткий обзор схемы подключения, а затем опишем базовую настройку сервер/клиента `ssh`. Также будет рассмотрено использование `scp`.

Завершает эту книгу глава об управлении пакетами с помощью командной строки (Гл. 13). В ней вы узнаете, как пользоваться утилитой **urpmi b** её антиподом **urpme**. Мы также покажем, как управлять источниками пакетов.

3. Примечание редактора

В философии **open-source** добровольное содействие всегда приветствуется! Обновление пакета документации к **Mandriva Linux** - 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 **Mandriva Linux** обращайтесь, пожалуйста, к администратору документации (<mailto:documentation@mandriva.com>) или посетите веб-страницу Проекта документации **Mandriva Linux** (<http://www.mandrivalinux.com/en/doc/project/>).



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

4. Соглашения, используемые в этой книге

4.1. Соглашения по набору текста

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

Форматированный пример	Значение
<i>inode</i>	Используется для выделения технических терминов, описываемых в Прил. А.
<code>ls -lta</code>	Используется для команд и их аргументов (см. Разд. 4.2.1).
<code>a_file</code>	Используется для имён файлов. Также может использоваться для имён RPM-пакетов.
<code>ls(1)</code>	Ссылка на страницу руководства (man). Чтобы прочитать страницу, просто наберите в командной строке <code>man 1 ls</code> .
<code>\$ ls *.pid</code>	Форматирование, используемое для снимков текстовых областей, которые вы можете увидеть на своем экране, включая результаты выполнения команд, распечатки программы и т.д.
<code>localhost</code>	Буквенные данные, не подходящие под описание ни одной из ранее определенных категорий. Например, ключевое слово, взятое из конфигурационного файла.
<code>OpenOffice.org</code>	Обозначает названия приложений. В зависимости от контекста, приложение 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.
<u>Файлы</u>	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.
<i>Le petit chaperon rouge</i>	Обозначает слова на иностранном языке.
Предупреждение!	Зарезервировано для особых предупреждений, чтобы подчеркнуть важность слов. Читается вслух.



Обозначает примечание. Обычно в нём содержится дополнительная информация об определённом контексте.



Обозначает подсказку. Это может быть общий совет о том, как выполнить определённое действие, или подсказки о полезных возможностях, которые могут облегчить вам жизнь.



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={arg1, arg2, arg3}] [опциональный арг. ...]
```

Эти соглашения являются стандартными и вы будете встречать 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**.



Мы используем заглавные буквы для обозначения алфавитных клавиш; это не означает, что вы должны вводить их в верхнем регистре. Однако существуют программы, в которых ввод **R** и **r** - это не одно и то же. Вы будете проинформированы при работе с такими программами.

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.

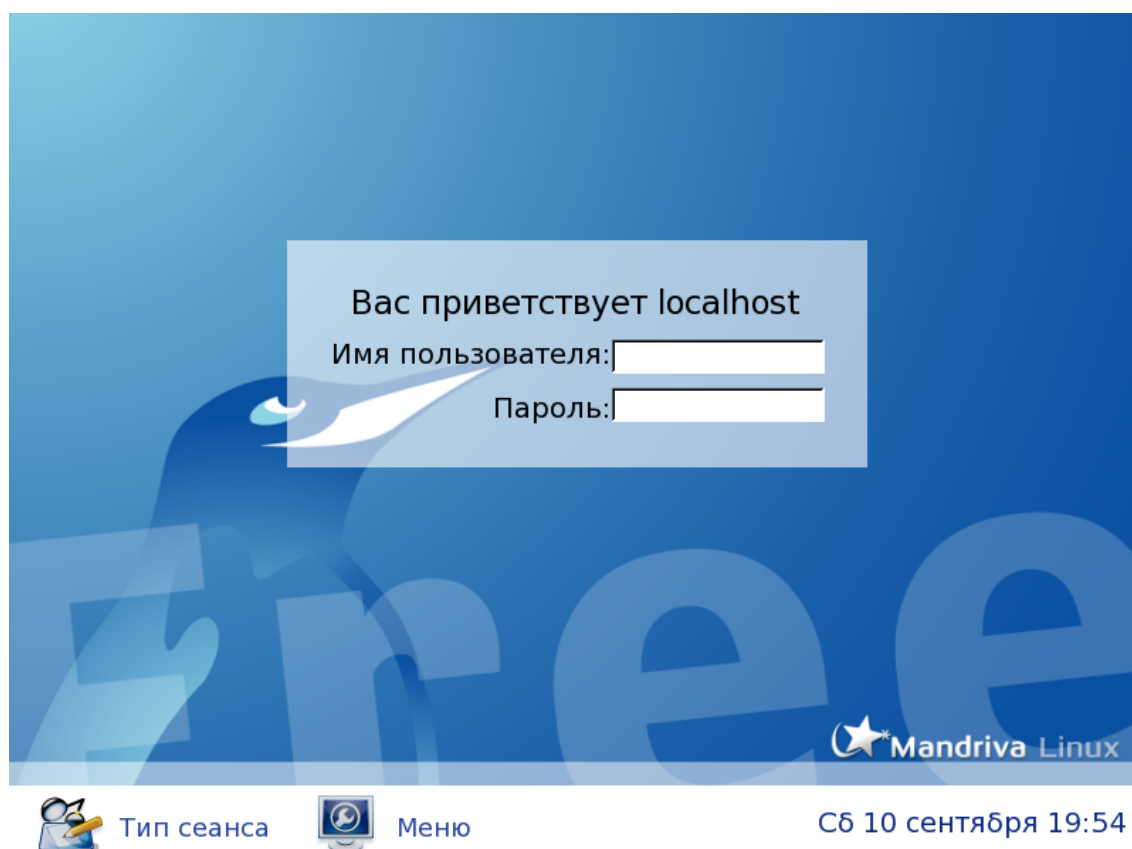


Рисунок 1-1. Сеанс входа в систему в графическом режиме

Чтобы войти в систему, вы должны сначала выбрать из списка свою учетную запись. Появится новое диалоговое окно, предлагающее вам ввести 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:

```
Mandriva Linux release 2006.0 (CodeName) for i586
Kernel 2.6.12-6mdk 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 Mandriva Linux 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. 4.

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 прочесть 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 `rw-r-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. 10).

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 (process 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 GNU/Linux, работающие в как файл-серверы для машин Windows® при помощи пакета программ `Sam-ba` (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. 10 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

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/`.

вам надо запустить эмулятор терминала. Зайдите в главное меню и вы найдете несколько эмуляторов в меню Система+Терминалы. Выберите эмулятор на свой вкус, например Konsole или RXvt. 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'a; (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 добавлять, 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 esssolol
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 частям 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. 4).
- **-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 -ls 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 **Mandriva Linux - 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 одновременно 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 РС. Следовательно, наиболее полезной она будет в том случае, если вы планируете вручную настроить разделы своего жёсткого диска.

2.1. Структура жёсткого 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 жёсткий диск.

2.1.1. Секторы

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

2.1.2. Разделы

Использование нескольких разделов позволяет вам создать много виртуальных жёстких 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 жёстком диске позволяет вам установить разные операционные системы на один физический жёсткий 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 жёстком 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 жёсткого 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. 6 и Гл. 4.

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 файлов¹, называемый **>@=5< (root)** и обозначающийся знаком **/**.



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 несколько разделов, чтобы отделить систему и программы от данных. Системный раздел будет содержать 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. 6).

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 Mandriva Linux 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 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 другие каталоги.

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.



Существует возможность изменить размер большинства разделов или использовать другую схему разметки диска 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 жёсткий 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 котором они находятся. Вот как именуются диски:

- Первичный ведущий (**primary master**) и первичный ведомый (**primary slave**) устройства IDE (будь то жёсткие 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 следующим образом:

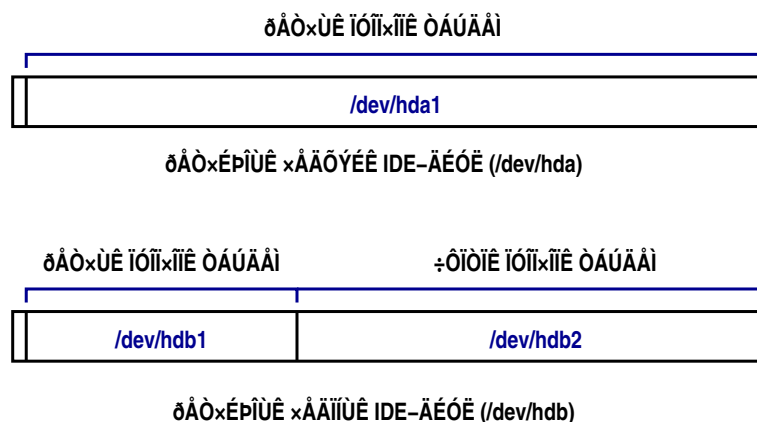


Рисунок 2-1. Первый пример именования разделов в GNU/Linux

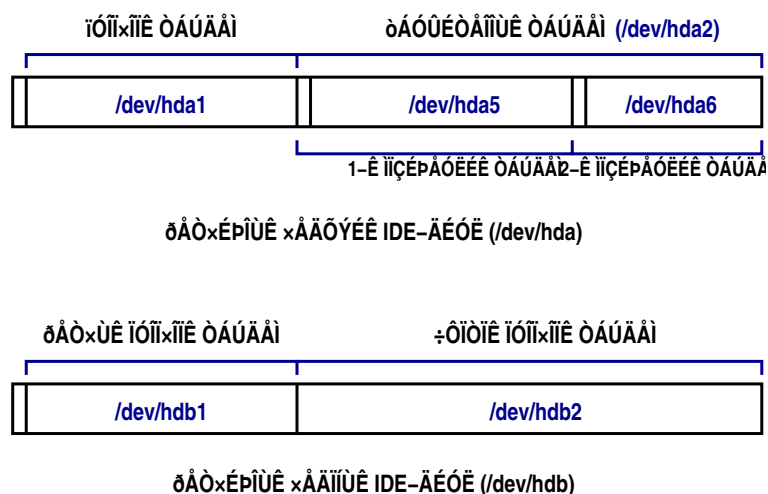


Рисунок 2-2. Второй пример именования разделов в GNU/Linux

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



В Mandriva Linux 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. 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.

3.1. 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 не обязательно 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.

3.2. 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. 3.3. 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.

3.3. /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 **Mandriva Linux** вместо **X Window System** теперь используется **Xorg**.

3.4. /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).

3.5. /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 4. 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.

4.1. 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 Mandriva Linux.

4.1.1. Различные используемые файловые системы

4.1.1.1. 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.



Предупреждение: перед изменением размера раздела, он должен быть размонтирован.

4.1.1.2. 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 журналируемой 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.

4.1.1.3. ReiserFS

В отличие от `ext3` `reiserfs` была написана с нуля. Это журналируемая файловая система наподобие `ext3`, но её 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 её оптимальной для работы с несколькими (тысячами или сотнями тысяч) небольших файлов. Она также хорошо ведёт себя при работе с большими файлами, что позволяет использовать её для разнообразных целей.



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

4.1.1.4. 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. Её внутренняя структура близка к структуре `reiserfs`.



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

4.1.1.5. 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. Её внутренняя структура имеет много разнообразных возможностей, таких как поддержка пропускной способности реального времени, экстенды (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.

4.1.2. 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 ^а	Очень хорошее	Очень хорошее
Поддержка ACL	Да	Да	Нет	Нет	Да
Примечания: а. Имеется возможность улучшить результаты восстановления после падения путем журналирования данных , а не только метаданных , посредством добавления опции <i>data=journal</i> в <i>/etc/fstab</i> .					

Таблица 4-1. Характеристики файловой системы

Максимальный размер файла зависит от многих параметров (т.е. размер блока для ext2/ext3), а также возможно дальнейшее развитие, в зависимости версии ядра и архитектуры.

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 ёмкости в многие ТБ без специальных без специальных “примочек” файловой системы, как это сделано в JFS для размера файловой системы.

4.1.3. А как насчёт производительности?

Сравнивать производительность файловых систем всегда очень сложно. Все тесты имеют свои ограничения, и результаты должны толковаться очень осторожно. 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.

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 эти четыре файловые системы дают приблизительно одинаковые результаты и все они обладают различными опциями для настройки под определённые задачи. Пожалуйста, обратитесь к соответствующей документации по файловым системам.

4.2. Всё 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 содержанию предоставлено приложениям.

4.2.1. 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 символического 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 `s`.
2. **Файлы блочного режима:** эти файлы являются периферийными устройствами, и, в отличие от символьных файлов, их содержимое **буферизуется**. Файлами этой категории являются, например, жёсткие диски, разделы жёсткого диска, дисководы, приводы CD-ROM и другие устройства хранения данных. 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 Mandriva Linux (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'a`, 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. 4.4.
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=
$
```

4.2.2. 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 символического 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 inode¹**. Причина этого заключается в том, что один и тот же файл может иметь несколько имен или вообще не

1. **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.

иметь имени. В UNIX® имя файла - это просто пункт в каталоге **inode**. Такой пункт называется ссылкой. Давайте рассмотрим ссылки более подробно.

4.3. 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-r--r-- 1 queen queen 0 Aug  6 19:26 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-r--r-- 2 queen queen 0 Aug  6 19:26 a
32555 -rw-r--r-- 2 queen queen 0 Aug  6 19:26 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-r--r-- 1 queen queen 0 Aug  6 19:26 b
$
```

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 имён) являются сетевые подключения. Вы никогда не увидите файла, соответствующему вашему подключению к веб-сайту **Mandriva Linux** (www.mandrivalinux.com), в своём 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 создаёте 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.

4.4. “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*'ax) 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)². Команда для создания именованного канала - *mkfifo*:

```
$ mkfifo a_pipe
$ ls -il
total 0
169 prw-rw-r-- 1 queen queen 0 Aug  6 19:37 a_pipe|
#
# Вы можете видеть, что счётчик ссылок равен 1,
# а файл является каналом ('p').
#
# Вы также можете использовать здесь ln:
#
# You can also use ln here:
#
$ ln a_pipe the_same_pipe
$ ls -il
total 0
169 prw-rw-r-- 2 queen queen 0 Aug  6 19:37 a_pipe|
169 prw-rw-r-- 2 queen queen 0 Aug  6 19:37 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...
#
[1]+  Stopped                  ls -F --show-control-chars --color=auto -d /proc/[0-9] >a_pipe
#
# ...Затем отправьте его в фоновый режим:
#
$ bg
[1]+  ls -F --show-control-chars --color=auto -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 -F --show-control-chars --color=auto -d /proc/[0-9] >a_pipe
```

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

```
$
```

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

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.

4.5. 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. 4.7 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 root root 6, 0 Feb 23 19:17 /dev/lp0
```

Here, the major and minor of `/dev/hdc` are 22 and 0, whereas for `/dev/lp0`, they are 6 and 0. Note that these numbers are unique per file category, which means that there can be a character mode file with major 22 and minor 0, and similarly, there can be a block mode file with major 6 and minor 0. These numbers exist for a simple reason: it allows the kernel to associate the correct operations to these files (that is, to the peripherals these files refer to): you don't handle a floppy drive the same way as, say, a SCSI hard drive.

4.6. 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 `lrwxrwxrwx`. 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.

4.7. 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. Ниже представлены некоторые из доступных атрибутов, полный список смотрите в `chattr(1)`:

1. **A ("no Access 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 ("append 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 dump")**: `dump` - это стандартная утилита UNIX[®] для резервного копирования. Она делает дампы любой файловой системы, для которой счетчик дампов равен 1 в файле `/etc/fstab` (см. Гл. 6). 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 ("immutable")**: файл или каталог с этим атрибутом вообще не может быть изменен: он не может быть переименован, на него не может быть создана ссылка³ 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 ("secure deletion")**: когда удаляется файл или каталог с этим атрибутом, блоки, которые он занимал на диске перезаписываются нулями.
6. **S ("Synchronous 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: это позволит избежать многих дисковых операций, и, в частности, может немного продлить жизнь аккумуляторов портативных компьютеров.

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

Глава 5. 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 её 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 другие 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.

5.1. 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
-r----- 1 root root 0 Aug 15 18:14 auxv
-r--r--r-- 1 root root 0 Aug 15 18:14 cmdline
lrwxrwxrwx 1 root root 0 Aug 15 18:14 cwd -> //
-r----- 1 root root 0 Aug 15 18:14 environ
lrwxrwxrwx 1 root root 0 Aug 15 18:14 exe -> /sbin/init*
dr-x----- 2 root root 0 Aug 15 18:14 fd/
-rw-r--r-- 1 root root 0 Aug 15 18:14 loginuid
-r--r--r-- 1 root root 0 Aug 15 18:14 maps
-rw----- 1 root root 0 Aug 15 18:14 mem
-r--r--r-- 1 root root 0 Aug 15 18:14 mounts
-rw-r--r-- 1 root root 0 Aug 15 18:14 oom_adj
-r--r--r-- 1 root root 0 Aug 15 18:14 oom_score
lrwxrwxrwx 1 root root 0 Aug 15 18:14 root -> //
-rw----- 1 root root 0 Aug 15 18:14 seccomp
-r--r--r-- 1 root root 0 Aug 15 18:14 stat
-r--r--r-- 1 root root 0 Aug 15 18:14 statm
-r--r--r-- 1 root root 0 Aug 15 18:14 status
dr-xr-xr-x 3 root root 0 Aug 15 18:14 task/
-r--r--r-- 1 root root 0 Aug 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 её аргументами нет пробелов, а в конце

строки нет символа окончания 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, определённые 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 символа окончания 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 - 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 мы получим следующее:

```
# ls -l /proc/8141/fd/
total 4
lrwx----- 1 peter peter 64 Aug  4 09:05 0 -> /dev/tty1
lrwx----- 1 peter peter 64 Aug  4 09:05 1 -> /dev/tty1
lrwx----- 1 peter peter 64 Aug  4 09:05 2 -> /dev/tty1
l-wx----- 1 peter peter 64 Aug  4 09:05 3 -> /home/peter/seti32/lock.sah
#
```

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 дескриптором¹. 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.

5.2. Информация об аппаратном обеспечении

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

```
$ ls -d [a-z]*
acpi/      diskstats  iomem      locks      pci         sysvipc/
asound/    dma        ioports    mdstat     scsi/       tty/
buddyinfo  driver/    irq/       meminfo    self@       uptime
bus/       execdomains kallsyms   misc       slabinfo    version
cmdline    fb         kcore      modules    splash       vmstat
config.gz  filesystems keys        mounts@    stat
cpuinfo    fs/        key-users  mtrr       swaps
```

1. Если вы помните, о чем говорилось в разделе Разд. 7.4, 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/
devices     interrupts  loadavg     partitions  sysrq-trigger
$

```

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:      543488      XT-PIC  timer
2:         0      XT-PIC  cascade
5:       109      XT-PIC  ohci_hcd:usb2, eth1
7:         1      XT-PIC  parport0
8:         0      XT-PIC  rtc
9:       3432      XT-PIC  acpi, NVidia CK8
10:      52855      XT-PIC  ehci_hcd:usb3, eth0
11:       7538      XT-PIC  libata, ohci_hcd:usb1
12:       1386      XT-PIC  i8042
14:         20      XT-PIC  ide0
15:       5908      XT-PIC  ide1
NMI:         0
LOC:         0
ERR:         0
MIS:         0

$ cat ioports
0000-001f : dma1
0020-002f : pic1
0040-004f : timer0
0050-005f : timer1
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00af : pic2
00c0-00cf : dma2
00f0-00ff : fpu
0170-017f : ide1
01f0-01ff : ide0
0376-037f : ide1
0378-037f : parport0
037b-037f : parport0
03c0-03cf : vesafb
03f6-03ff : ide0
03f8-03ff : serial
0778-077f : parport0
0970-097f : 0000:00:0b.0
0970-097f : sata_nv
09f0-09ff : 0000:00:0b.0
09f0-09ff : sata_nv
0b70-0b7f : 0000:00:0b.0
0b70-0b7f : sata_nv
0bf0-0bf3 : 0000:00:0b.0
0bf0-0bf3 : sata_nv
0cf8-0cff : PCI conf1
4000-407f : motherboard
4000-4003 : PM1a_EVT_BLK
4004-4005 : PM1a_CNT_BLK
4008-400b : PM_TMR
4020-4027 : GPE0_BLK
4080-40ff : motherboard
4080-40ff : pnp 00:00
4200-427f : motherboard
4200-427f : pnp 00:00
4280-42ff : motherboard
4280-42ff : pnp 00:00
4400-447f : motherboard
4400-447f : pnp 00:00
4480-44ff : motherboard
44a0-44af : GPE1_BLK
5000-503f : motherboard
5000-503f : pnp 00:01

```

```
5100-513f : motherboard
5100-513f : pnp 00:01
9000-9fff : PCI Bus #02
9000-907f : 0000:02:07.0
9000-907f : 0000:02:07.0
ac00-ac0f : 0000:00:0b.0
ac00-ac0f : sata_nv
b000-b07f : 0000:00:0b.0
b000-b07f : sata_nv
b800-b8ff : 0000:00:06.0
b800-b8ff : NVidia CK8
bc00-bc7f : 0000:00:06.0
bc00-bc7f : NVidia CK8
c000-c007 : 0000:00:04.0
c000-c007 : forcedeth
c400-c41f : 0000:00:01.1
f000-f00f : 0000:00:09.0
f000-f007 : ide0
f008-f00f : ide1
```

```
$cat dma
 3: parport0
 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
-----
0000:00:01.1          c400-c41f
0000:00:04.0          c000-c007
0000:00:06.0      b800-b8ff bc00-bc7f
0000:00:09.0      f000-f00f
0000:00:0b.0      0970-0977 0b70-0b73 0bf0-0bf3 ac00-ac0f b000-b07f
0000:02:07.0      9000-907f      9000-907f
cascade             4       2
CK8                  9
dma                  0080-008f
dma1                  0000-001f
dma2                  00c0-00df
eth0                  10
eth1                   5
forcedeth             c000-c007
fpu                   00f0-00ff
GPE0_BLK              4020-4027
GPE1_BLK              44a0-44af
i8042                 12
ide0                  14 01f0-01f7 03f6-03f6 f000-f007
ide1                  15 0170-0177 0376-0376 f008-f00f
keyboard              0060-006f
motherboard           4000-407f 4080-40ff 4200-427f 4280-42ff 4400-447f 4480-44ff 5000-503f 5100-513f
NVidia                b800-b8ff bc00-bc7f
ohci_hcd:usb1         11
parport0              3       7 0378-037a 037b-037f 0778-077a
PCI                   0cf8-0cff 9000-9fff
pic1                  0020-0021
pic2                  00a0-00a1
PM1a_CNT_BLK          4004-4005
PM1a_EVT_BLK          4000-4003
PM_TMR                4008-400b
pnp                   4080-40ff 4200-427f 4280-42ff 4400-447f 5000-503f 5100-513f
rtc                   8 0070-0077
sata_nv              0970-0977 0b70-0b73 0bf0-0bf3 ac00-ac0f b000-b07f
serial               03f8-03ff
timer                 0
timer0               0040-0043
timer1               0050-0053
vesafb               03c0-03df
$
```

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

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 АРМ 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 АРМ. 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 и каталоги, 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 пакеты `acpid` и `kapacity`.

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. Степень 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.

5.3. Отображение и изменение параметров ядра

Назначение подкаталога `/proc/sys` - 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 Mandriva Linux 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 её 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 отпадёт необходимость каждый раз набирать их заново. Другим решением является изменение файла `/etc/sysctl.conf`. Дополнительную информацию смотрите в `sysctl.conf(5)` и `sysctl(8)`.

3. Не путать с *хакерами*!

Глава 6. Файловые системы и точки монтирования

Как мы уже видели в главе Гл. 2, все файлы в системе организованы в виде единого дерева. И на самом деле, всякий раз, когда мы обращаемся к съёмному устройству наподобие CD-ROM или к удалённому каталогу на файловом сервере, его содержимое будет одной из “веток” в этом дереве.

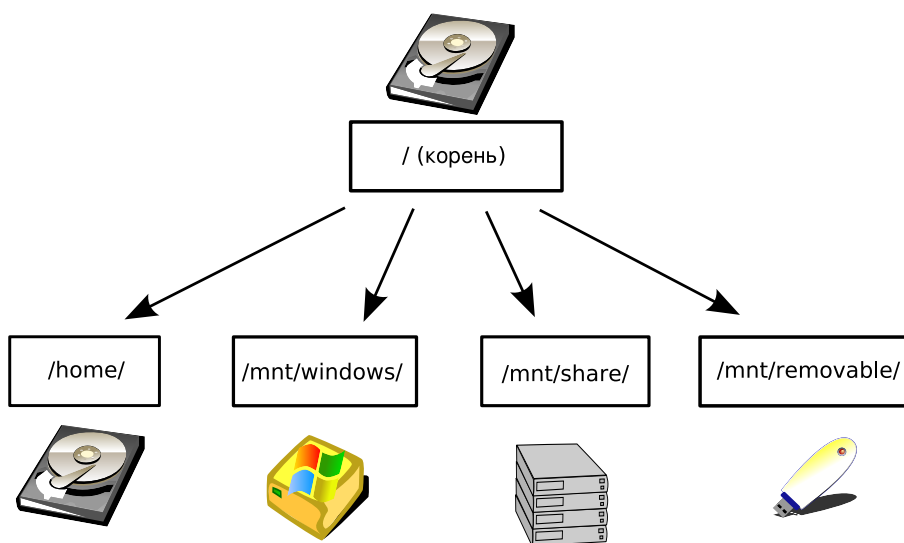


Рисунок 6-1. Иллюстрация точек монтирования

Рис. 6-1 демонстрирует следующее: корень - раздел GNU/Linux - содержит другой раздел для каталога `/home/`, а также раздел `Windows®`, удалённый общий ресурс с файлового сервера (`Windows®` или `UNIX®`) и USB-ключ. В настоящее время многие устройства могут быть примонтированы к файловой системе GNU/Linux, включая практически все существующие типы файловых систем, WebDAV и даже такие экзотические вещи, как почта Google™...

Наилучший способ разобраться с этими понятиями - рассмотреть практическое применение. Предположим, вы только что приобрели новый жёсткий диск и на нём нет никаких разделов. Ваш раздел Mandriva Linux заполнен до отказа и вместо того, чтобы начинать все сначала, вы решаете перенести целый раздел древовидной структуры¹ на новый диск. Так как новый диск обладает большей ёмкостью, вы решаете перенести самый большой каталог на него: `/usr`.

Мы будем использовать этот пример на протяжении всей главы Разд. 6.2, но сначала немного теории.

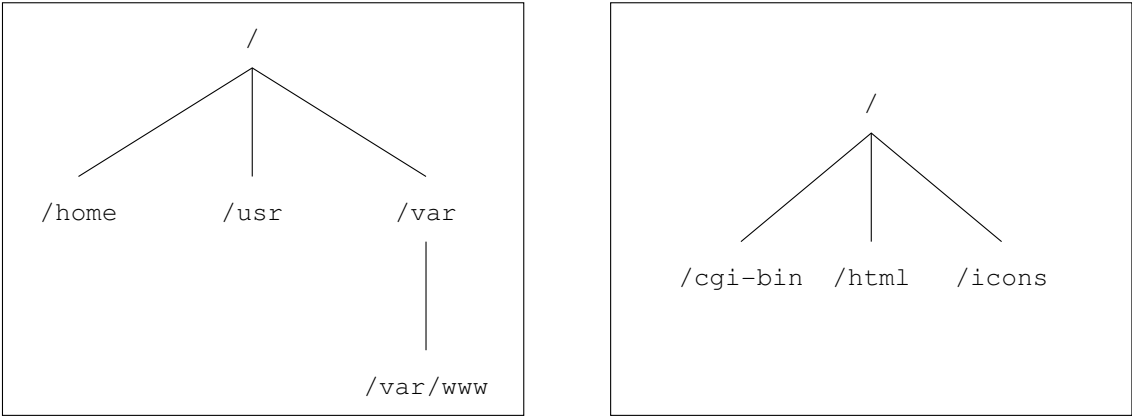
6.1. Принципы

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. 6-2 и Рис. 6-3.

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



είναι ένα σύστημα αρχείων που έχει ως βάση το / (όχι το /usr ή /var) και ως σημείο монтирования το /var/www. Αυτό σημαίνει ότι το /var/www είναι το σημείο όπου θα γίνει η σύνδεση με το σύστημα αρχείων.

Рисунок 6-2. Файловая система еще не примонтирована

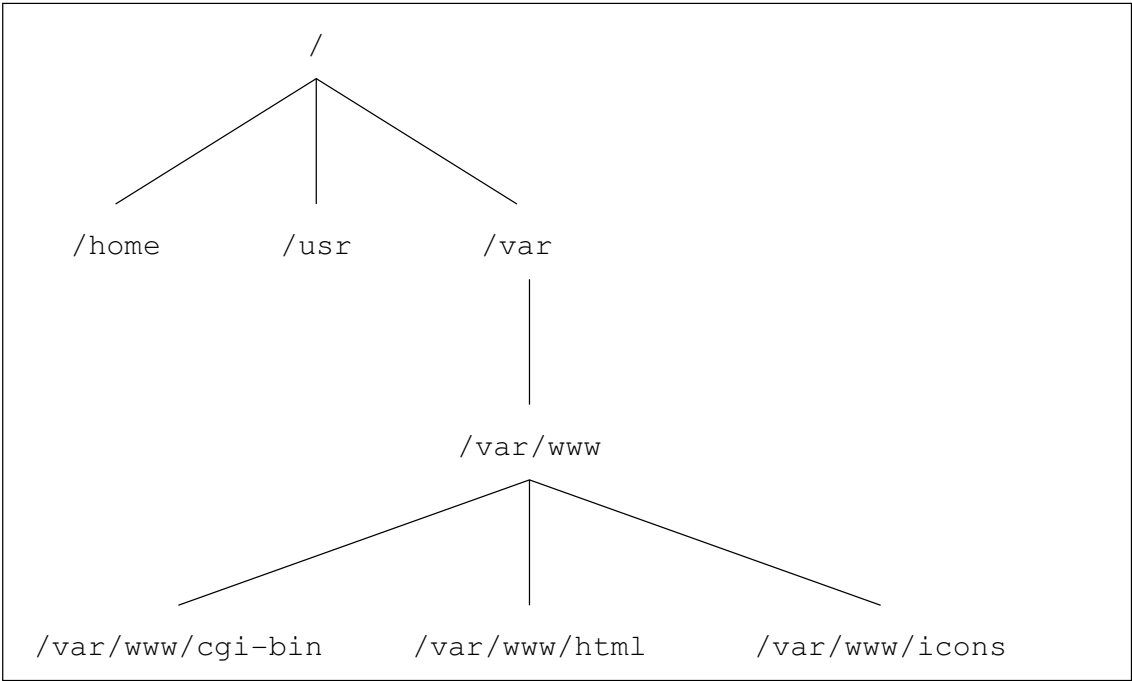


Рисунок 6-3. Файловая система теперь примонтирована

Можете себе представить, сколько это даёт преимуществ: древовидная структура всегда будет оставаться одинаковой, находишься ли вы на одной или на нескольких файловых системах. 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
```

6.2. 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 известен под именем `/dev/hdb`². 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 разметать диск и создавать файловые системы, мы сможем продолжить дальше.

6.3. 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> [-о 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` попытается самостоятельно определить, какая файловая система находится в разделе, путем считывания суперблока.

Опция `-о` 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:

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

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

или:

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



Иногда может случиться 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>
```

Эта 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.

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=0022,user,ioccharset=utf8,noauto,ro,exec,users 0 0
/dev/fd0 /mnt/floppy supermount dev=/dev/fd0,fs=ext2:vfat,--,umask=0022,ioccharset=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` (*File System Check*).

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. 5. 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 (большого размера) еще не примонтирован. `rm -Rf /usr/*`.
- И, наконец, мы монтируем новый каталог `/usr`: `mount /usr/`.

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

Глава 7. Введение в командную строку

В главе Гл. 1 вы увидели как запускать `shell`. В этой 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.

7.1. 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).

7.1.1. `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. создаёт родительские каталоги, если они не существовали до этого. Без этой опции `mkdir` просто вызовет сбой, жалуюсь 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, жалуюсь 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 изменения¹. 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.

7.1.2. `rm`: Удаление файлов или каталогов

Команда `rm` (*ReMove* - удалить) является аналогом команд `DOS del` и `deltree`, и содержит дополнительные опции. Её 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 **Mandriva Linux**, `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 файлы².

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.

7.1.3. `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`: принудительное выполнение 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.

7.1.4. **ср**: Копирование файлов и каталогов

Команда **ср** (*CoPy* - копировать) является аналогом команд **DOS** **сору** и **хсору**, и содержит дополнительные опции. Её синтаксис выглядит следующим образом:

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

Вот наиболее общие опции **ср**:

- **-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 **ср**.

Несколько примеров:

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

7.2. 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.

7.2.1. **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); её 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**.

7.2.2. 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 определён 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, `rw`x) + `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. Вообще, 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 всём 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.

7.3. Шаблоны подстановки в командном процессоре

Вы, вероятно, уже использовали символы *подстановка*, 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 регистре³;
- {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/mandriva/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.

7.4. Перенаправления и каналы

7.4.1. 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...

7.4.2. Перенаправления

Допустим вам надо 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`.

7.4.3. 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 Терминал⁵:

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

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

7.5. 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.

7.5.1. Пример

Допустим, что в вашем личном каталоге имеется `D09;_A>G5=L_4;8==K<_8_A;>6=K<_4;0_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 всё, что вам нужно сделать - это нажать клавишу **Enter** для подтверждения и приняться за чтение файла.



Используйте клавишу **q** для завершения просмотра файлов.

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

7.5.2. 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 `ns1<TAB>` будет `nslookup`), а имя файла - для всех остальных слов, если только перед словом не идёт “магический” символ типа `~`, `@` или `$`. В данном случае `shell` попытается завершить имя пользователя, имя машины или имя переменной окружения соответственно⁶. 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 определённые переменные окружения будет ввести в пустой строке последовательность **Ctrl-X-\$**. Другой пример: если вам нужно просмотреть страницу `man` для команды `nslookup`, просто введите `man ns1`, а затем **Esc-!**, и командный процессор автоматически завершит команду до `man nslookup`.

7.6. 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 завершён, 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'a 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.

7.7. Заключительное слово

Как видите, **shell** обладает очень широкими возможностями и эффективное его использование является делом практики. В этой относительно длинной главе мы упомянули лишь о нескольких доступных командах: **Mandriva Linux** 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 8. 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.).

8.1. 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>).

8.1.1. Краткое представление

Запускается Emacs из командной строки 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.

8.1.2. 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.

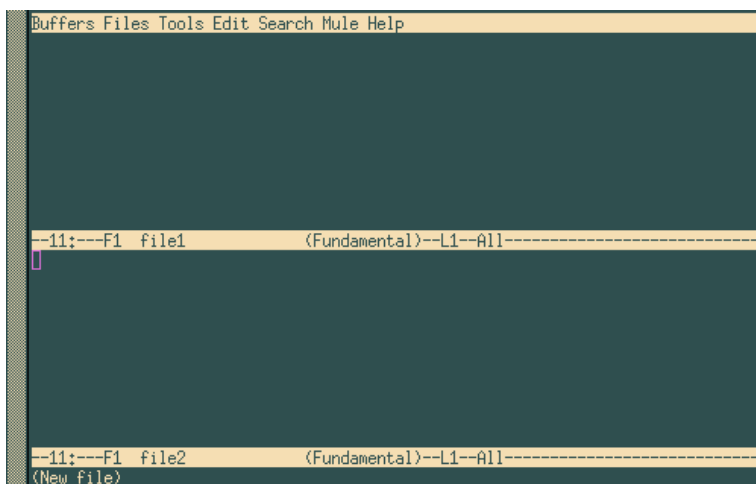


Рисунок 8-1. Редактирование двух файлов одновременно

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 следующие комбинации 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 клавиш со стрелками².

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*, воспользовавшись для этого клавишей **Tab** как в **bash**.

8.1.3. 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.

8.1.4. Копирование, вырезание, вставка, поиск

Допустим, что вы находитесь в следующей ситуации Рис. 8-2.

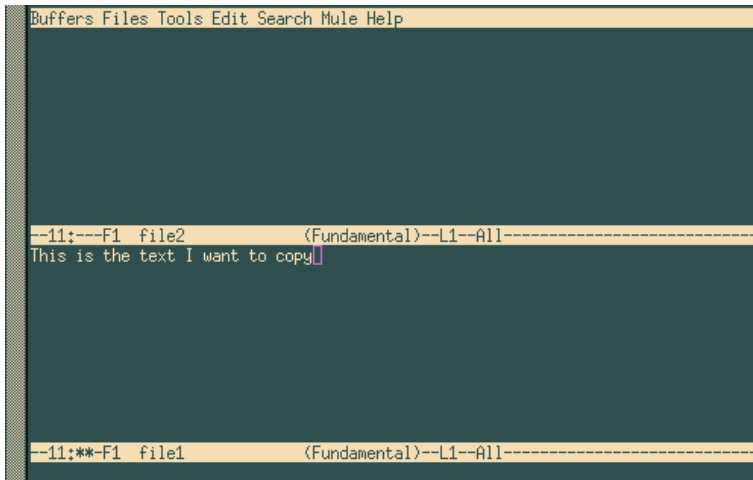


Рисунок 8-2. Emacs, перед копированием текстового блока

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. 8-2, 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:

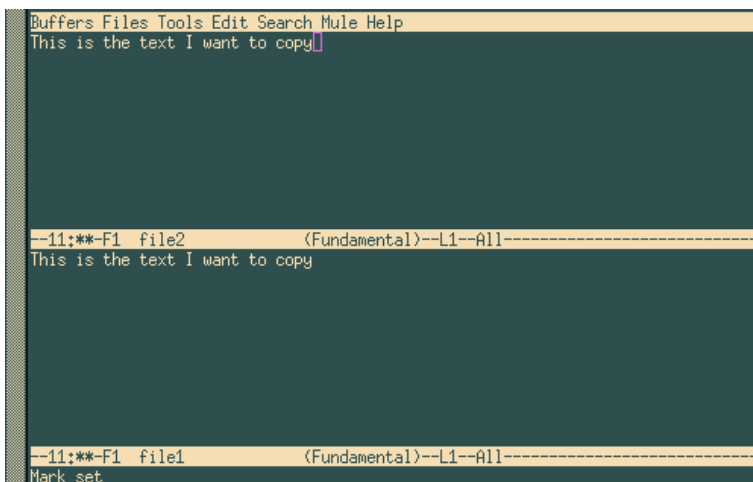


Рисунок 8-3. Копирование текста в emacs

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.

8.1.5. 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.

8.2. 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 Mandriva Linux - 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/>).

8.2.1. 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:

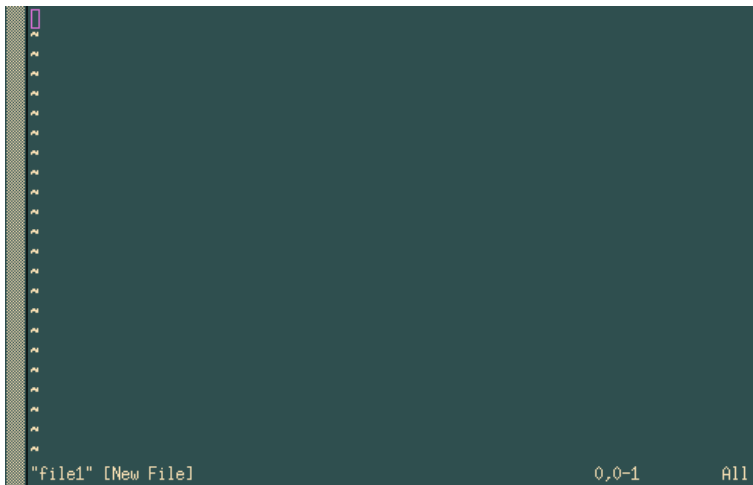


Рисунок 8-4. Исходная позиция в VIM

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:



Пожалуйста, обратите внимание, что клавиатурные сокращения должны вводиться точно так, как показано, Vi различает символы верхнего и нижнего регистров в командах, поэтому команда **a** - это не одно и то же, что команда **A**.

- **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. Чтобы вернуться 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. Всё, что вы введёте до нажатия на клавишу **Enter**, Vi будет интерпретировать как команду `ex`. Если вы удалите введённую вами команду и сам символ **:**, вы будете возвращены в командный режим, а курсор вернется на исходную позицию в тексте.



В режиме `ex` вам доступно автозавершение команд: введите первые символы команды и нажмите клавишу **Tab**, чтобы завершить её набор.

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

8.2.2. 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!**.

8.2.3. 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, вверх 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.

3. Сокращённая клавиатурная команда для **dl** (удаление одного символа справа) - **x**; сокращённая клавиатурная команда для **dh** (удаление одного символа слева) - **X**; **dd** удаляет текущую строку.

8.2.4. 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 английского алфавита в нижнем регистре.

Для использования анонимной ячейки памяти введите команду “как есть”. Так, команда **y12w** скопирует в анонимную память 12 слов после курсора⁴. 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.

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:

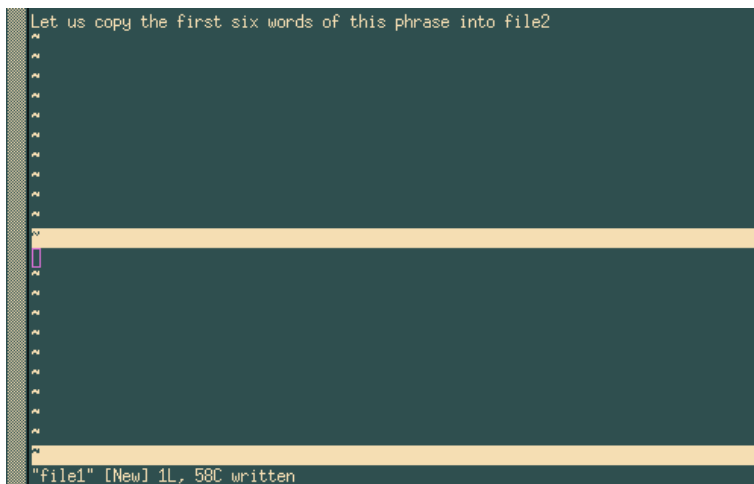


Рисунок 8-5. VIM, перед копированием текстового блока

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. 8-6.

4. Но только, если курсор находится в начале первого слова!

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

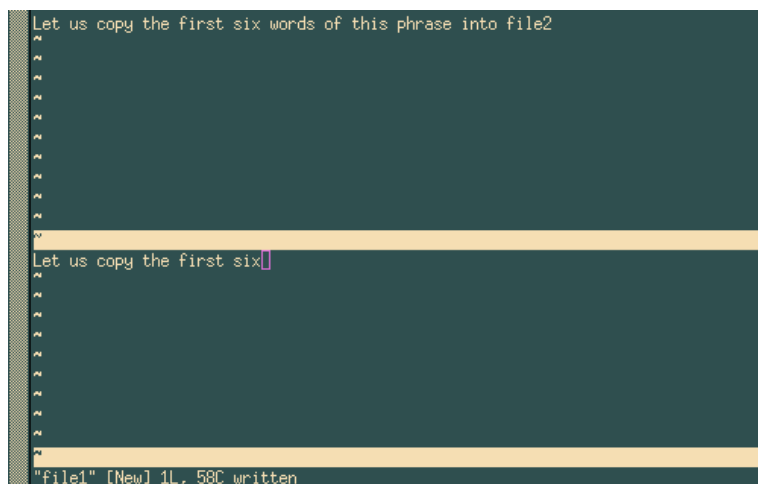


Рисунок 8-6. ViM, после копирования текстового блока

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 **?**.

8.2.5. 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.

8.3. 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 9. 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. 7.4). 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!

9.1. 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.

9.1.1. 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 вывод¹. 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 размер² 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 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`. Её 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.

9.1.2. `grep`: Поиск строк в файлах

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

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

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

Команду **grep** можно использовать в каналах.

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@mandriva.com`, достаточно будет добавить этот электронный адрес в наш файл `./patterns.txt`, выполнив следующую команду:

```
# echo 'peter@mandriva.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@mandriva.com`, 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.

9.1.3. **egrep**: Регулярные выражения и фильтрация

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.

3. Хотя можно выполнить фильтрацию просто по шаблону состояния, пожалуйста, позвольте нам показать вам в этом примере новую команду.

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. 7.3 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 (верхний и нижний регистры) и все цифры. 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, т.е. соответствует ни одному или одному вхождению, но не более одного.

*

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\nabcl abcl\nabcedef\nabcedabcd\nabcedef 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 определённых символов вы должны их “заэкранировать”, вставив перед ними обратную косую черту. Вот эти символы: ?, +, {, |, (,) и конечно же \. Для использования их в своих 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.

9.1.4. **wc**: Подсчет элементов в файлах

Команда **wc** (*Word Count* - подсчёт слов) используется для подсчёта числа строк, слов и символов в файлах. Также она полезна для вычисления самой длинной строки. Её 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
```

9.1.5. **sort**: Сортировка содержимого файла

Ниже представлен синтаксис этой мощной утилиты для сортировки⁴:

```
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`). Чтобы 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.

9.2. **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. 7.3).
- `-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` или `c2` и удовлетворяющие критерию `c3`, вам понадобится использовать скобки и ввести `(<c1> -o <c2>) -a <c3>`. Вы должны **закранировать** (деактивировать) 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 -l`.
- `-exec <командная_строка>`: для каждого найденного файла выполняет <командную_строку>. <Командная_строка> должна заканчиваться символом `;`, который вы должны закранировать, 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 следующее⁵:

```
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'a**: если бы мы ввели `(..)` вместо `\(.. \)`, командный процессор интерпретировал бы их и попытался выполнить `-name "*.htm" -o -name "*.html"` в **sub-shell'e**... 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 выполнялась регулярно. Простым способом периодического запуска команды является использование демона **cron**, как показано в следующем разделе.

9.3. Запуск команд по расписанию

9.3.1. 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!

9.3.2. 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\(1\)](#).

9.4. Архивирование и упаковка данных

9.4.1. 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'y`, 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 находиться в каталоге,
# файлы которого вы хотите заархивировать!
#
$ cd /shared
$ tar cjf ~/images.tar.bz2 images/
```

Как видите, здесь мы использовали три опции: с 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.

9.4.2. **bzip2** и **gzip**: Программы упаковки данных

Мы уже говорили об этих двух программах, когда рассматривали **tar**. В отличие 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 его работы требуется больше ресурсов. Несмотря на это, **gzip** всё ещё 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`.

9.5. 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 10. 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.

10.1. 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 об этих процессах.

10.1.1. 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.

10.1.2. 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`.

10.2. 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`.

10.2.1. `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 ещё 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.

10.2.2. 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.

10.3. Отправка сигналов процессам: kill, killall и top

10.3.1. 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.

10.3.2. 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. 10-1.

```
top - 22:54:53 up 15:10, 0 users, load average: 0.02, 0.06, 0.01
Tasks: 80 total, 1 running, 79 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.7% us, 0.7% sy, 0.0% ni, 97.7% id, 0.0% wa, 0.0% hi, 0.0% si
Mem: 515640k total, 484920k used, 30720k free, 39856k buffers
Swap: 506008k total, 4k used, 506004k free, 244752k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
16666	reine	15	0	25232	14m	23m	S	0.7	2.8	0:51.21	kscd
1732	root	15	0	57860	21m	38m	S	0.3	4.3	21:14.37	X
13510	reine	16	0	2172	1036	1964	R	0.3	0.2	0:00.03	top
13512	reine	15	0	9364	2580	8912	S	0.3	0.5	0:00.01	import
1	root	16	0	1580	516	1424	S	0.0	0.1	0:03.45	init
2	root	34	19	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/0
3	root	5	-10	0	0	0	S	0.0	0.0	0:00.55	events/0
4	root	5	-10	0	0	0	S	0.0	0.0	0:00.02	kblockd/0
5	root	15	0	0	0	0	S	0.0	0.0	0:00.03	kapmd
6	root	25	0	0	0	0	S	0.0	0.0	0:00.00	pdflush
7	root	15	0	0	0	0	S	0.0	0.0	0:00.20	pdflush
8	root	15	0	0	0	0	S	0.0	0.0	0:00.04	kswapd0
9	root	10	-10	0	0	0	S	0.0	0.0	0:00.00	aio/0
11	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kseriod
15	root	15	0	0	0	0	S	0.0	0.0	0:00.83	kjournald
121	root	16	0	2036	1204	1588	S	0.0	0.2	0:00.31	devfsd
247	root	15	0	0	0	0	S	0.0	0.0	0:00.00	khubb

Рисунок 10-1. 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.

10.4. Установка приоритетов для процессов: `nice`, `renice`

Все процессы в системе работают с определёнными приоритетами, также называемыми “значениями `nice`”, которые могут изменяться от -20 (наивысший приоритет) до 19 (наименьший приоритет). Если приоритет не определён, 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. Обычные пользователи могут только понижать приоритет своих собственных процессов в диапазоне от 0 до 19. Суперпользователь (`root`) может установить для любого процесса любое значение приоритета.

10.4.1. `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** получают наименьший приоритет и не будут затруднять работу процессов других пользователей.

10.4.2. 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'a с Mandriva Linux:

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

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.

11.1. В начале был `init`

При запуске системы, когда ядро всё 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`'ом². 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.

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 `Mandriva Linux` вы могли увидеть, что существует шесть уровней выполнения:

- 0: полная остановка машины;

1. Вот почему размещение `/sbin` не в корневой файловой системе было бы очень плохой идеей. На этом этапе ядро ещё 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.

- 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 **rc3.d**:

```
$ ls /etc/rc.d/rc3.d/
K09dm@      S12syslog@   S24messagebus@  S40atd@      S91dictd-server@
S01udev@     S13partmon@  S25haldaemon@   S55sshd@     S92lisa@
S03iptables@ S15 cups@    S25netfs@        S56ntpd@     S95kheader@
S05harddrake@ S17alsa@     S29numlock@      S56rawdevices@ S99local@
S10network@   S18sound@    S33nifd@         S75keytable@
S11shorewall@ S20xfs@      S34mDNSResponder@ S90crond@
$
```

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 3-й уровень выполнения, она сначала запускает команду `K09dm` (т.е. `/etc/rc.d/init.d/dm stop`). Далее она запускает все скрипты S: сначала `S01udev`, который в свою очередь вызывает `/etc/rc.d/init.d/udev start`, затем `S03iptables` 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.

11.2.1. Настройка уровней выполнения служб

Вы также можете воспользоваться командой `chkconfig` для добавления, удаления, включения или выключения служб на заданных уровнях выполнения. Используйте команду `chkconfig --add имя_службы` для добавления (включения) службы `имя_службы` на всех поддерживаемых³ уровнях выполнения и `chkconfig --del имя_службы` для удаления (выключения) названной службы на всех уровнях выполнения.



Выполните команду `chkconfig --list`, чтобы увидеть список поддерживаемых служб, их имена и состояние для всех определённых уровней запуска.

Выполнение команды `chkconfig --levels 35 sshd on` включит сервер SSH (`sshd`) на 3-м и 5-м уровнях, а выполнение `chkconfig --levels 3 sound off` уберёт поддержку звука на 3-м уровне. Если вы опустите параметр `--levels список_уровней`, названные службы будут включены или выключены на уровнях выполнения `and 5`. Но **Однако** обратите внимание, что вы можете включить службы на

3. “Поддерживаемые” уровни выполнения означает, что, например, сетевые службы не будут добавлены на 2-й уровень, который не поддерживает работу с сетью.

уровнях выполнения без соответствующей поддержки этих служб, поэтому нужные уровни выполнения лучше указывать явно.

11.2.2. Управление службами в работающей системе

В работающей системе службами можно управлять с помощью команды `service`, независимо от того, настроены ли они на запуск на определённом уровне выполнения или нет. Её синтаксис:

```
service имя_службы действие
```

Где **имя_службы** - это название службы, выводимое командой `chkconfig --list`, а **действие** может быть одним из следующих:

start

Запускает названную службу. Пожалуйста, обратите внимание, что большинство служб предупредят вас, если они уже выполняются, а вы пытаетесь снова запустить их: используйте вместо неё опцию `restart`, описанную ниже.

stop

Останавливает названную службу. Пожалуйста, обратите внимание, что если вы остановите службу, все подключенные к ней пользователи будут автоматически отключены.

restart

Останавливает, а затем запускает названную службу. Это эквивалентно выполнению команды `service имя_службы stop && service имя_службы start`. Пожалуйста, обратите внимание, что если вы перезапустите службу, все подключенные к ней пользователи будут автоматически отключены.

другие действия

Различные службы поддерживают различные действия (предыдущие действия поддерживаются всеми службами). Например, `reload` перезагружает файл конфигурации без перезапуска самой службы; `force-stop` принудительно останавливает службу; `status` сообщает о состоянии службы; и др. Выполнение `service имя_службы` без параметров выводит информацию обо всех действиях, поддерживаемых названной службой.

Глава 12. Безопасный удалённый доступ

Системным администраторам необходимо иметь возможность подключаться к географически удалённым машинам, чтобы редактировать конфигурационные файлы, управлять службами, запускать программы и т.п. Ранее для доступа к удалённым машинам использовался `telnet`, однако он представляет собой небезопасное решение. Т.к. весь обмен информацией ведётся через общедоступные (и по сути незащищённые) каналы связи (Интернет), системные администраторы нуждаются в безопасном удалённом доступе. `ssh` (**Secure SHell**, защищённый шелл) позволяет вам получать доступ к удалённым машинам безопасным методом, шифруя все передаваемые данные.

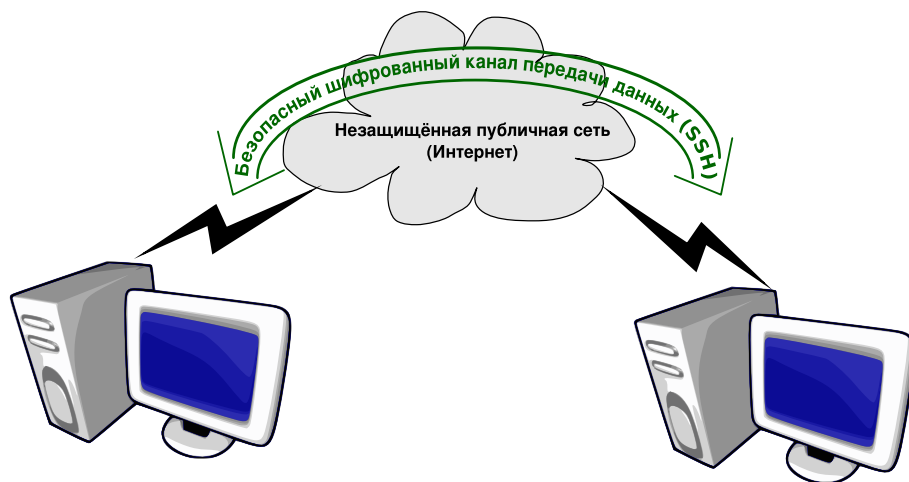


Рисунок 12-1. Схема SSH-соединения

12.1. Настройка сервера SSH

Под “сервером” мы подразумеваем машину, к которой вы будете подключаться. Просто убедитесь, что установлен пакет `openssh-server`, и что запущена служба `sshd`¹.

Базовая настройка сервера SSH позволяет пользователям иметь доступ к машине при условии, что у них на ней есть учётная запись. Если вы хотите ограничить доступ по SSH до списка заданных пользователей, отредактируйте файл `/etc/ssh/sshd_config`, добавив или изменив в нём строку следующего вида:

```
AllowUsers queen peter@192.168.0.*
```

В примере выше доступ к машине по SSH будет разрешён только пользователям `queen` и `peter`; пользователю `peter` будет разрешено подключаться только с машины `192.168.0.` (локальной) сети.

Пользователи должны будут подключиться с использованием своих обычных учётных записей, а затем использовать команду `su` для получения прав `root`’а. Чтобы разрешить пользователям подключаться по SSH непосредственно под `root`’ом, измените строку `PermitRootLogin no` на `PermitRootLogin yes`. Пожалуйста, учтите, что хотя этот параметр и удобен, но является не слишком безопасным.

Пожалуйста, обратитесь к страницам руководства `sshd(8)` и `sshd_config(5)` для получения дополнительной информации о параметрах сервера SSH и его настройке.

12.2. Настройка клиента SSH

Под “клиентом” мы подразумеваем машину, с которой вы будете выполнять подключение. Просто убедитесь, что установлен пакет `openssh-clients`.

Наберите команду `ssh имя_пользователя@удалённая_машина`, чтобы подключиться к удалённой машине под учётной записью `имя_пользователя`. Удалённая система попросит вас ввести свой пароль. Введите его и вы получите доступ так, как если бы сидели за консолью удалённой системы.

1. Выполните команду `service sshd start`, чтобы запустить её. Служба SSH настраивается на запуск во время загрузки.

Подключаетесь ли вы к одной машине или ко многим (обычная ситуация для системных администраторов), можно избежать ввода пароля с помощью ключей SSH. Воспользуйтесь командой `ssh-keygen`, чтобы сгенерировать свой ключ SSH, а затем `ssh-copy-id имя_пользователя@удалённая_машина`, чтобы скопировать свой ключ на удалённые машины. При вызове команды `ssh-copy-id` вам будет предложено ввести пароль на удалённой машине, но только по одному для каждой из систем. Теперь вы можете использовать SSH для подключения к удалённым машинам без необходимости вводить пароль.



Для реализации этого метода вы должны выполнить команду `ssh-add` и ввести свой пароль (созданный при генерации ваших ключей SSH) каждый раз, когда вы начинаете сеанс на клиентской машине.

Если вы получаете сообщение о том, что не может быть установлено соединение с вашим агентом аутентификации, выполните команду `eval `ssh-agent`` (обратите внимание на обратные кавычки) перед запуском `ssh-add`.

12.3. Обмен файлами между локальной и удалённой системами

Чтобы выгрузить файлы на удалённую систему, на которой работает сервер SSH, воспользуйтесь командой `scp` (от англ. **Secure CoPy**, защищённое копирование). Её синтаксис:

```
scp [опции] локальный/путь [пользователь@]удалённый_хост: [полный/путь/на/удалённом/хосте]
```

Если вы не укажете часть `пользователь@`, тогда будет использован ваш логин на клиентской машине. Если вы опустите путь на удалённой машине, файл будет скопирован в домашний каталог пользователя на удалённой системе. Обратите внимание на двоеточие (:), отделяющее имя пользователя и название машины от пути на удалённой машине.

Чтобы загрузить файлы с удалённой системы на локальную машину, используйте следующий синтаксис:

```
scp [опции] [пользователь@]удалённый_хост:полный/путь/на/удалённом/хосте локальный/путь
```

Если путь источника указывает на каталог, тогда опция `-r` (рекурсивно) является обязательной. Для получения дополнительной информации об опциях `scp`, обратитесь, пожалуйста к `scp(1)`.

Глава 13. Управление пакетами из командной строки

Приложения Rpmrake - это попросту графические интерфейсы к мощным консольным утилитам `urpmi`. Для тех, кто желает управлять своими пакетами из командной строки (полезно, например, если вы работаете удалённо), у нас есть наиболее полезные команды.

13.1. Установка и удаление пакетов

Это выполняется двумя простыми командами:

```
urpmi <имя_пакета>
```

В результате выполнения этой команды будет установлен пакет с указанным именем, если он существует, конечно. Если пакет содержит строку `имя_пакета`, это также даст положительный результат. Если этой строке соответствует несколько пакетов, вы увидите нумерованный список найденных совпадений: просто введите интересующий вас номер и нажмите **Enter**.

Если пакет, который вы пытаетесь установить, имеет зависимости (другие пакеты, необходимые для его нормальной работы), будет выведен их список. Просмотрите его и нажмите клавишу **Y**, чтобы установить пакеты.

```
urpme <имя_пакета>
```

Эта команда удалит пакет с именем `<имя_пакета>`. Если от пакета, который вы пытаетесь удалить, зависят другие пакеты, будет выведен их список с объяснением, почему они должны быть удалены. Просмотрите этот список и нажмите клавишу **Y**, чтобы удалить все пакеты.



И `urpmi`, и `urpme` поддерживают опцию `--auto` для установки или удаления пакетов с автоматической обработкой зависимостей.

Для получения дополнительной информации об опциях этих команд обратитесь к страницам руководства `urpmi(8)` и `urpme(8)`.

13.2. Управление источниками

Источники программного обеспечения отличаются от носителей, с которых вы можете устанавливать пакеты. Чтобы работала команда `urpmi`, должен быть определён как минимум один источник. К предопределённым источникам относятся те, которые вы использовали при установке своей системы (по сети, с CD, DVD и т.п.). Вам следует определить другие источники для установки багфиксов и обновлений по безопасности. Добавлять и удалять источники просто, но должен строго соблюдаться синтаксис.

13.2.1. Добавление нового источника

```
urpmi.addmedia <название> <URL>
```

Эта команда позволяет вам добавить новый источник с локального диска, съёмного устройства (CD/DVD) или из сети по протоколам HTTP, FTP, NFS, ssh или rsync. Синтаксис URL зависит от типа источника, поэтому вам настоятельно рекомендуется обратиться к `urpmi.addmedia(8)` перед тем, как использовать её.



Если вы добавляете новый источник с обновлениями, используйте в командной строке `urpmi.addmedia` с опцией `--update`.

Вы можете использовать онлайн-ресурсы наподобие Easy Urpmi Page (<http://easyurpmi.zarb.org/>), если вы не знаете, где искать источники с полезными приложениями, упакованными специально для вашей системы Mandriva Linux. Также сайт Mandriva Club (<http://club.mandriva.com/>) предоставляет

модуль `Urpmi media` (<http://club.mandriva.com/modules.php?name=Mirrors-list>) для тестирования дополнительных пакетов.



Список источников Mandriva Club доступен только его членам.

13.2.2. Удаление источников

```
urpmi.removemedias <название>
```

Эта команда просто удалит источник с этим **названием**. Если вы не запомнили название источника, выполните команду `urpmi.removemedias` без параметров. При этом будет выведен перечень всех источников.

13.2.3. Обновление источников

```
urpmi.update <название>
```

Эта команда сканирует названный источник и обновляет список его пакетов. Это полезно для часто изменяющихся источников с обновлениями по безопасности и багфиксами. Используйте опцию `-a`, чтобы пересканировать все определённые источники.

13.2.4. Порядок источников

Порядок, в котором источники определены в файле `/etc/urpmi/urpmi.cfg`, имеет большое значение, поскольку он определяет источник, с которого будут устанавливаться пакеты, если они доступны на нескольких источниках: пакеты будут установлены с первого в списке источника.



При добавлении сетевых источников они могут быть добавлены перед съёмными и локальными источниками. Это связано с тем, что зачастую на сетевых источниках находятся более новые пакеты, чем на локальных или съёмных.

13.3. Советы и подсказки

13.3.1. Синтезированные и полные списки

При добавлении источника доступны на выбор два списка пакетов: синтезированный и полный. Используйте опцию `--probe-synthesis`, чтобы попытаться найти синтезированный список пакетов, или опцию `--probe-hdlist`, чтобы попытаться найти полный список. Синтезированные списки меньше по размеру, что больше подходит для пользователей с более медленным сетевым подключением. Однако они более ограничены в поиске информации о пакетах.

13.3.2. Поиск пакетов, содержащих определённый файл

Допустим, что для вашей системы нужен определённый файл, но вы не знаете в каком из пакетов он находится. Выполните команду `urpmf <имя_файла>` и на экран будет выведен перечень всех пакетов, содержащих или не содержащих этот файл.



Если вы используете синтезированные списки `urpmf` сможет выполнить поиск файлов только в уже установленных пакетах.

Вы даже можете указать часть имени. Например, команда `urpmf salsa` выведет список всех пакетов, имена файлов в которых содержат слово `salsa`.

```
[root@test queen]# urpmf salsa
kaffe:/usr/lib/kaffe/lib/i386/libtritonusalsa-1.1.2.so
kaffe:/usr/lib/kaffe/lib/i386/libtritonusalsa.la
kaffe:/usr/lib/kaffe/lib/i386/libtritonusalsa.so
```

13.3.3. Обновление пакетов

Эта команда обновит названные пакеты:

```
urpmi.update -a && urpmi --update <имя_пакета>
```

Эта команда автоматически обновит все необходимые пакеты, как если бы это выполнено через **Mandriva Update**:

```
urpmi.update -a && urpmi --update --auto-select --auto
```

Если у вас нет ни одного источника, отмеченного как содержащий обновления, вы можете опустить опцию `--update` в командах `urpmi` из примеров выше.

Приложение А. Глоссарий

account

учетная запись, аккаунт; в системе UNIX® это комбинация, состоящая из имени, личного каталога, пароля и shell'a, 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. Также это может использоваться для фундаментальных или очень простых операций, наподобие сложения двух целых чисел.

background

фоновый режим; в контексте shell’a, 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, предупреждающий 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 (дискета, CD, DVD или любое другое устройство), 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 кэш.

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, 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 права доступа 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 несвободное (коммерческое) 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 компокуемых 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

эскапирование, 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 прав доступа и владельцев файлов и другие возможности.

FAQ

(*Frequently Asked Questions, часто задаваемые вопросы, “ЧаВО”*) документ, содержащий ряд вопросов с ответами по определенной теме. Согласно истории, **FAQ**'и появились в тематических конференциях (**newsgroup**). 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 **Mandriva Linux** 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 особым 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 видеопамати 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 Mandriva Linux.

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 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; 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-I**).

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 комплексными 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 французский. **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'a**, 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 инопод 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. Она является 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 ограничена 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 - Mandriva Linux (**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**, свободен 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; директивы 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’a, 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 закешировано, быстро получит ответ, так как 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 комментарий*) 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 Mandriva Linux.

run level

уровень выполнения; конфигурация системного программного обеспечения, позволяющая существовать только выбранным процессам. Разрешенные файлы для каждого уровня выполнения определены в файле /etc/inittab. Обычно существует семь предопределенных уровней выполнения: 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 отличие 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 Mandriva Linux, 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

тенивые 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 права 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 выдает 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 умолчанию 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

- .bashrc, 48
- атрибут
 - файл, 49
- вирус, 11
- владелец, 49
 - изменение, 49
- временные метки
 - ctime, 47
- временные метки
 - atime, 47
 - mtime, 47
- группа, 7
 - изменение, 49
- диски, 17
- документация
 - Mandriva Linux, 3
- домашний каталог
 - раздел, 19
- значения
 - дискретные, 51
- интернационализация, 2
- канал, 52
 - анонимный, 30
 - именованный, 30
 - файл, 28
- каталог
 - копирование, 49
 - переименование, 48
 - перемещение, 48
 - создание, 47
 - удаление, 47
- командная строка
 - завершение, 53
- командная строка
 - введение, 47
 - утилиты, 65
- командный процессор
 - шаблоны подстановки, 51
- командный процессор, 47
- команды
 - at, 73
 - bzip2, 75
 - cat, 13
 - cd, 12
 - chgrp, 49
 - chmod, 50
 - chown, 49
 - cp, 49
 - crontab, 72
 - grep, 66
 - gzip, 75
 - init, 81
 - kill, killall, 78
 - less, 14, 52
 - ls, 14
 - mkdir, 47
 - mount, 43
 - mv, 48
 - ps, 77
 - pwd, 12
 - rm, 47
 - rmdir, 48
 - scp, 86
 - sed, 52
 - ssh, 85
 - ssh-add, 86
 - ssh-keygen, 85
 - tar, 74
 - touch, 47
 - umount, 43
 - urpmi, 87
 - wc, 52
 - поиск, 70
- консоль, 8
- корневой
 - каталог, 21, 36
 - раздел, 18
- краткий обзор
 - команда, 5
- модули, 39
- окружение
 - переменная, 13
 - процесс, 36
- основной
 - младший, 19
 - старший, 19
- пакеты
 - управление, 87
- пароль, 8
- перенаправление, 52
- подстановка
 - символ, 51
- пользователи, 7
 - обычные, 5
- порядок сортировки, 51
- права доступа, 50
- приглашение, 8, 12
- приложения
 - ImageMagick, 53
 - терминалы, 53
- программирование, 2
- процесс, 11, 35, 54
- процессы, 77
- разделы, 17, 41
 - логический, 19
 - основной, 19
 - расширенный, 19
- разработка, 2
- сборка пакетов, 2
- своп, 17
 - размер, 18
- сектор, 17
- символы
 - подстановка, 51
 - специальные, 54
- ссылка
 - жесткая, 32
 - символическая, 32

- стандартный
 - ввод, 51
 - вывод, 51
 - поток ошибок, 51
- текстовые редакторы
 - Emacs, 57
 - vi, 60
- удалённый доступ
 - автоматизация, 85
- удалённый доступ, 85
- уровень выполнения, 81
- утилиты
 - обработка файлов, 47
- учетная запись, 7
- файл
 - атрибут, 33, 49
 - блочный режим, 28
 - блочный режим, 31
 - копирование, 49
 - переименование, 48
 - перемещение, 48
 - поиск, 70
 - символьный режим, 28
 - символьный режим, 31
 - создание, 47
 - сокет, 28
 - ссылка, 28, 29
 - удаление, 47
- Borges, ??
- DocBook, ??
- FHS, 21
- GID, 9
- IDE
 - устройства, 19
- inode, 28
 - таблица, 28
- Mandriva Club, 1
- Mandriva Expert, 1
- Mandriva Linux
 - безопасность, 1
 - списки рассылок, 1
- Mandriva Store, 2
- Peter Pingus, 5
- PID, 11
- Queen Pingusa, 5
- RAM memory, 18
- root
 - пользователь, 8
- SCSI
 - диски, 19
- shell, 12
- Soundblaster, 19
- ssh
 - клиент, 85
 - ключ, 85
 - сервер, 85
- swap
 - раздел, 18
- udev, 20
- UID, 9

- UNIX®, 7
- usr
 - раздел, 18