Mandrake Linux 9.0

Руководство по Командной Строке

(http://www.MandrakeSoft.com)

Mandrake Linux 9.0: ***Руководство по Командной Строке***

Опубликовано 2002-09-19

Camille Bégnis, Christian Roy, Fabian Mandelbaum, Joël Pomerleau, Vincent Danen, Roberto Rosselli del Turco, Stefan Siegel, Marco De Vitis, Alice Lafox, Kevin Lecouvey, Christian Georges, John Rye, Robert Kulagowski, Pascal Rigaux, Frédéric Crozat, Laurent Montel, Damien Chaumette, Till Kamppeter, Guillaume Cottenceau, Jonathan Gotti, Christian Belisle, Sylvestre Taburet, Thierry Vignaud, Juan Quintela, Pascal Lo Re, Kadjo N'Doua, Mark Walker, Roberto Patriarca, Patricia Pichardo Bégnis, Alexis Gilliot, Arnaud Desmons, Wolfgang Bornath, Alessandro Baretta, Aurélien Lemaire

Юридическое замечание

Инструментарий, использованный для создания этого руководства

Это руководство было написано в *DocBook*. *Borges (http://linux-mandrake.com/en/doc/project/Borges/)* был использован для управления набором всех входящих файлов. Исходные файлы в XML обрабатывались `openjade` и `jadetex`, используя таблицы стилей Norman Walsh. Снимки экрана были получены с использованием `xwd` или *GIMP* и конвертированы `convert` (из пакета *ImageMagick*). Все это программное обеспечение входит в поставку **Mandrake Linux** и все его части являются свободным программным обеспечением

Содержание

Список таблиц

Список иллюстраций

Предисловие

1. О Mandrake Linux

Mandrake Linux является дистрибутивом *GNU/Linux*, который поддерживается **MandrakeSoft** S.A. **MandrakeSoft** был рожден в Интернет в 1998 году с главной целью создать простую в использовании и дружественную систему *GNU/Linux*. Два столба **MandrakeSoft** это открытые исходные коды и совместная работа.

1.1. Контакт с Сообществом Mandrake

Следуйте различным Интернет-ссылкам, которые приведут вас к различным ресурсам, посвященным **Mandrake Linux**. Если вы хотите знать больше о компании **MandrakeSoft**, зайдите на ее web-сайт (`http://www.mandrakesoft.com/`). Есть также сайт дистрибутивов **Mandrake Linux** (`http://www.mandrakelinux.com/`) и все что к ним относится.

Прежде всего, **MandrakeSoft** с гордостью представляет свою новую платформу открытой помощи. MandrakeExpert (`http://www.mandrakeexpert.com/`) не просто еще один web-сайт, где люди помогают другим в решении их компьютерных проблем, получая взамен плату, соответствующую качеству предоставленных услуг. Этот сайт предоставляет новый опыт, базирующийся на доверии и удовольствии от полезности для других за их вклад.

В дополнение, MandrakeCampus (`http://mandrakecampus.com/`) предоставляет сообществу *GNU/Linux* возможность свободного обучения и курсы по всем технологиям и проблемам открытого программного обеспечения. Он также дает учителям, преподавателям и обучающимся место, где они могут поделиться своими знаниями.

Существует также сайт для "mandrake-маньяков", который называется Mandrake Forum (`http://www.mandrakeforum.com/`) : первичный сайт по **Mandrake Linux**, на котором размещены советы, предупреждения, хитрости, слухи, пред-объявления, полуофициальные новости, и всякое такое. Кроме того, этот сайт , размещаемый **MandrakeSoft**, является интерактивным, поэтому, если вам есть что сказать нам, или если вы хотели бы поделиться чем-либо с другими пользователями или поискать что-либо: это то самое место, где это можно сделать!

Следуя принципам философии движения Открытого Исходного Кода, **MandrakeSoft** предлагает множество вариантов поддержки (`http://www.mandrakelinux.com/ru/ffreesup.php3`) для дистрибутивов **Mandrake Linux**. Вы приглашаетесь участвовать в различных Списках рассылки (`http://www.mandrakelinux.com/ru/flists.php3`) , где сообщество **Mandrake Linux** демонстрирует свое оживление и увлеченность.

И в заключение, не забывайте заходить на MandrakeSecure (`http://www.mandrakesecure.net/`) . Этот сайт посвящен всем аспектам безопасности дистрибутивов **Mandrake Linux**. Именно здесь вы найдете консультации по вопросам безопасности и багам, а также статьи на тему безопасности и секретности. Эта информация является необходимой для каждого системного администратора или пользователя, заинтересованного в защите.

1.2. Поддерживаем Mandrake

По многочисленным заявкам, **MandrakeSoft** предоставляет возможность своим счастливым клиентам делать взносы (`http://www.mandrakelinux.com/donations/`) для поддержки развития системы **Mandrake Linux**. Ваши пожертвования помогут **MandrakeSoft** поставлять своим пользователям более совершенный чем когда-либо дистрибутив, более безопасный, простой, соответствующий современным требованиям и с большим количеством поддерживаемых языков.

Для многих талантливых: ваши навыки будут очень полезны для выполнения одной из многих задач, требуемых при создании системы **Mandrake Linux**:

- Создание пакетов: система *GNU/Linux* собрана из многих программ, размещенных в сети Интернет. Эти программы нужно собирать в пакеты и дать им возможность успешно работать совместно.
- Программирование: существует множество проектов, напрямую поддерживаемых **MandrakeSoft**: найдите наиболее интересный для вас и предложите свою помощь главному разработчику.
- Интернационализация: перевод web-страниц, программ и документации к ним.

- Документация: последнее, но немаловажное: книга, которую вы сейчас читаете, требует множество усилий, чтобы оставаться на современном уровне с эволюцией системы.

Обратитесь к странице спонсоров (`http://www.mandrakesoft.com/labs/`) чтобы узнать больше о том, как вы можете сделать свой вклад в эволюцию и развитие **Mandrake Linux**.

3-его августа 2001 года, после утверждения себя как мирового лидера в среде Открытых Исходников и программного обеспечения *GNU/Linux*, **MandrakeSoft** стал первой *Linux* компанией, зарегистрированной на Европейской бирже. Если вы уже акционер **MandrakeSoft** или хотели бы им стать, наша страница инвесторов (`http://www.mandrakesoft.com/company/investors`) предоставит вам наилучшую финансовую информацию касательно компании.

1.3. Приобретение продукции Mandrake

Для фанов **Mandrake Linux**, которые хотят получать выгоду от удобного заказа в онлайн, **MandrakeSoft** теперь продает свои продукты по всему миру через MandrakeStore (`http://www.mandrakestore.com/`) , свой web-сайт электронной коммерции. Здесь вы найдете не только операционную систему **Mandrake Linux** и сетевые утилиты (Single Network Firewall), но и особые подписные предложения, поддержку, другое программное обеспечение и лицензии, обучающую документацию, книги, посвященные *GNU/Linux*, а также другие товары от **MandrakeSoft**.

2. Комментарий Редактора

Как вы можете заметить, переходя от одного раздела к другому, эта книга является сборным документом от разных авторов. Даже при том, что много внимания уделялось технической и словарной последовательности, стиль каждого автора очевидно сохранен.

Некоторые из авторов пишут на английском, который не всегда является для них родным. Следовательно, вы можете встретить странные конструкции предложений; не стесняйтесь сообщать нам если что-то вдруг непонятно.

Согласно философии open-source, помощникам всегда рады! Вы можете помочь проекту документации различным способом. Если у вас есть много времени, вы можете написать целый раздел. Если вы говорите на иностранном языке, вы можете поучаствовать в интернационализации этой книги. Если у вас есть идеи как улучшить содержимое, дайте нам знать - принимаются советы даже по заметкам.

За любой информацией о проекте документации **Mandrake Linux** обращайтесь к администратору документации (`mailto:documentation@mandrakesoft.com`).

3. Соглашения, Используемые в Этой Книге

3.1. Соглашения Набора Текста

Для того, чтобы четко дифференцировать специальные слова от текстового потока, команда документации использует различные начертания. В этой таблице приведены примеры каждого специального слова или группы слов с ее начертанием и описания того, что это означает.

Пример	Значение
inode	Используется для обозначения технических терминов
`ls -lta`	Обозначает команду или аргументы команды. Применяется для обозначения команд, опций команд, а также имен файлов. Также смотрите раздел " Разд. 3.2.1 " .
ls(1)	Ссылка на страницы руководства (`man` page). Для получения такого руководства в *`shell`* (или в командной строке), просто наберите `man 1 ls` .
`$ ls *.pid` `imwheel.pid`	Команда документации использует это выделение для отображения текстовых снимков экрана. Сюда входят результаты выполнения команд, распечатки программы, и т.д.

Пример	**Значение**
`localhost`	Это - литеральные данные, которые вообще не входят не в одну из описанных в этой таблице категорий. Например, это могут быть ключевые слова, взятые из конфигурационного файла.
`Apache`	Используется для обозначения названий программ. Так обозначается название программы, а не команды. Название программы и команды могут совпадать, но, в зависимости от контекста, они будут выделяться по-разному.
F iles	Используется для обозначения пунктов меню или текстовых строк в графическом интерфейсе. Подчеркнутый символ указывает быструю клавишу (конечно, если она существует).
`SCSI-Bus`	Так обозначается компьютер или его часть (любое устройство).
Le petit chaperon rouge	Так обозначаются слова на иностранном языке (не русском и не английском).
Warning!	Таким образом выделяется что либо важное. Например предупреждения. Советуем прочитать это вслух **:-)**

Этот значок выделяет примечание. В общем случае, это дополнительная информация для текущего контекста.

Этот значок обозначает совет. Это может быть общий совет, касающийся того, как выполнить определенное действие, или просто полезная хитрость, которая здорово облегчит вам жизнь

Предупреждение. При встрече этого значка будьте очень осторожны. Он всегда обозначает, что сейчас речь пойдет о чем-то очень важном.

3.2. Основные Соглашения

3.2.1. Краткий обзор команд

В примере, приведенном ниже, показано как автор описывает аргументы команды:

```
command <non
      literal argument> [--option={arg1,arg2,arg3}]
      [optional arg. ...]
```

это соглашение является стандартным и используется во многих других местах, например в страницах руководства (man pages).

Между знаками “<” (меньше чем) и “>” (больше чем) помещаются **обязательные** аргументы. Такие параметры должны быть обязательно указаны. Вы не должны копировать то что, написано в руководстве, а вместо этого подставить свое значение. Например, `<filename>` требует чтобы вы указали реальное имя файла. Если это имя `foo.txt` то вы должны ввести `foo.txt`, но не `<foo.txt>` и уж не как не `<filename>` .

Квадратные скобки “[ ]” обозначают необязательные аргументы, которые вы можете указать или не указать для вашей команды.

Троеточие “...” подразумевает, что может быть добавлено произвольное число аргументов.

Фигурные скобки “{ }” указывают на то что один из заключенных в них аргументов должен содержаться в этом месте.

3.2.2. Специальные примечания

Иногда вам будет предложено нажать комбинации клавиш. Например напрямую Ctrl+R , Это значит что вам нужно нажать и удерживать клавишу Ctrl и не отпуская ее нажать клавишу R Аналогичные действия нужно проделать для нажатия клавиш Alt и Shift .

Теперь о меню. Для того чтобы выбрать пункт меню File→Reload user config (**Ctrl+R**) вам нужно: выбрать текст File, находящийся в меню (обычно в горизонтальных меню он находится в верхней части окна). В результате должно появиться подменю. В нем вы должны выбрать пункт Reload user config. Вместо этого вы можете просто использовать комбинацию клавиш (как было описано выше) для быстрого вызова этой функции Ctrl+R , и, в конечном итоге, вы получите такой же результат.

3.2.3. Универсальные пользователи системы

Всякий раз, когда возможно, мы использовали двух универсальных пользователей в наших примерах:

Queen Pingusa	Этот пользователь был создан во время инсталляции системы.
Peter Pingus	Этот пользователь был создан администратором системы уже после установки.

Введение

Добро пожаловать и спасибо вам за использование **Mandrake Linux**! Данное онлайновое руководство предназначено для людей, которые решили погрузиться в глубины своей системы *GNU/Linux*, и для тех, кто хотел бы использовать ее огромные возможности. Руководство состоит из трех частей:

- В части под названием *Система Linux*, мы представляем вам командную строку и различные возможности по ее использованию. Здесь же мы обсуждаем основы редактирования текста, что является весьма важным и существенным под *GNU/Linux*.

  Гл. 1 представляет миры *UNIX* и, более определенно, мир *GNU/Linux*. В ней раскрываются стандартные утилиты работы с файлами и некоторые полезные возможности, предоставляемые *shell*. Очень важно полностью понять концепции, обсуждаемые в этой главе, прежде чем перейти к главе Гл. 3. Затем следует дополнительная глава Гл. 2, в которой обсуждается вопрос работы с жесткими дисками под *GNU/Linux*, а также концепция разделов жестких дисков.

  Затем мы переходим к Гл. 4. Так как большинство конфигурационных файлов *UNIX* являются текстовыми файлами, вы непременно пожелаете редактировать их в ***текстовом редакторе*** . Вы узнаете как использовать два наиболее популярных текстовых редактора в мире *UNIX* и *GNU/Linux*: могущественный *Emacs* и современный (!) *Vi*.

  Теперь вы наверняка сможете выполнять основные задачи в вашей системе. Следующие две главы рассказывают о практическом использовании командной строки (Гл. 5), и контроле процессов (Гл. 6).

- В части *Глубины Linux*, мы затрагиваем вопросы ядра *Linux* и архитектуры файловой системы.

  Гл. 7 освещает принципы организации файлового дерева. Системы *UNIX* имеют тенденцию разрастаться, но при этом каждый файл имеет свое особое место в особом каталоге. После прочтения этой главы вы будете знать о том, где и какие файлы расположены в зависимости от их роли в системе.

  Затем мы расскажем о ***файловых системах*** и ***точках монтирования*** (Гл. 8). Мы дадим определения для обоих этих терминов и поясним их на практических примерах.

  Гл. 9 посвящена файловым системам *GNU/Linux*. После представления некоторых из них, мы обсудим типы файлов и некоторые дополнительные утилиты и концепции типа inode и pipe. Гл. 10 представит специальную файловую систему *GNU/Linux* `/proc`.

  Гл. 11 рассказывает о процедуре загрузки **Mandrake Linux** и как ее эффективно использовать.

- В части *Расширенное Использование* мы завершаем книгу темами, которые предназначены только для отважных и весьма опытных пользователей, которые захотят заняться практикой. Гл. 12 проведет вас через необходимые шаги по сборке и установке свободного программного обеспечения из исходных кодов. Прочтение этой главы должно привлечь вас испытать это, даже если на первый взгляд, возможно, все выглядит пугающе. И наконец, Гл. 13 это один из последних шагов к полной автономии *GNU/Linux*. После прочтения и принятия теории, объясненной в этой главе, начинайте обращать пользователей *Windows* в *GNU/Linux* (если вы еще этого не начали!).

В конце книги приведены две лицензии, используемые для программного обеспечения и документации для *GNU/Linux*, соответственно: Прил. A и Прил. B, а также Прил. C и индекс.

Глава 1. Базовые Концепции Системы UNIX

Название "*UNIX*" может быть некоторым из вас уже знакомо. Возможно вы даже уже используете систему *UNIX* на работе, в таком случае эта глава будет вам вероятно не очень интересна.

А для тех, кто еще никогда не работал с ней, прочтение этой главы является абсолютно необходимым. Знание концепций, которые будут представлены здесь, отвечает на необычайно большое количество вопросов, которые обычно задают начинающие в мире GNU/Linux пользователи. Более того, некоторые из этих концепций станут хорошим ответом на большинство проблем, с которыми вы можете столкнуться в будущем.

1.1. Пользователи и Группы

Понятия пользователей и групп чрезвычайно важны, так как они оказывают непосредственное влияние на все другие концепции, которые представляет эта глава.

Linux является действительно ***многопользовательской (multiuser)*** системой, и, чтобы пользоваться своей *GNU/Linux* машиной, вы должны иметь ***аккаунт*** на ней. Когда вы создали пользователя во время установки, вы на самом деле создали аккаунт пользователя. У вас были запрошены следующие пункты:

- "Реальное имя" пользователя (фактически, что захотите);
- имя ***логина*** (login);
- ***пароль*** (вы ввели его, не правда ли?).

Два важных параметра здесь это название логина (обычно обозначается просто login) и пароль. Все, что вам нужно, что попасть в систему, знать эти два параметра.

Когда вы создаете пользователя, также создается группа по умолчанию. . Как мы увидим позже, группы очень полезны, когда вы разделяете файлы между несколькими людьми. Группа может содержать столько пользователей, сколько пожелаете, и это разделение является обычным делом для больших систем. Например, в университете вы можете иметь одну группу на факультет, другую для преподавателей, и так далее. Обратное тоже верно: пользователь может быть членом одной или нескольких групп, максимум до 32-х. Преподаватель математики, например, может быть членом группы преподавателей и группы студентов, с которыми он работает.

Все это, однако, не сообщает вам о том, как войти в систему. Итак, расскажем об этом.

Если вы выбрали графический вход в систему, ваш стартовый экран будет выглядеть так, как здесь Рис. 1-1.

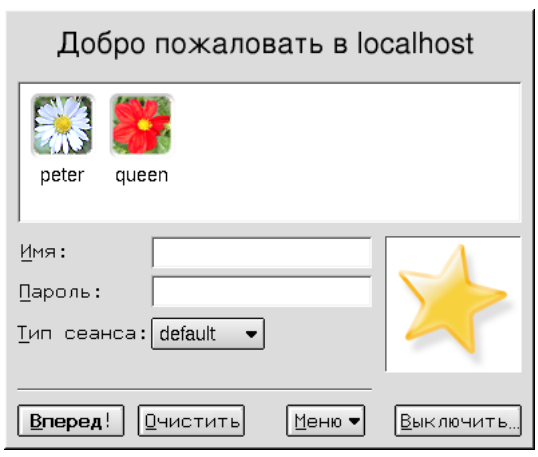

Рисунок 1-1. Сеанс входа в систему в графическом режиме

Чтобы войти, вы должны ввести ваш логин в тестовом поле Login:, а затем ввести пароль в поле для пароля . Имейте в виду, что пароль вам прийдется вводить вслепую; символы не будут отображаться ***echo*** в поле ввода.

Если вы в консольном режиме , ваш экран будет выглядеть похоже на Рис. 1-2.

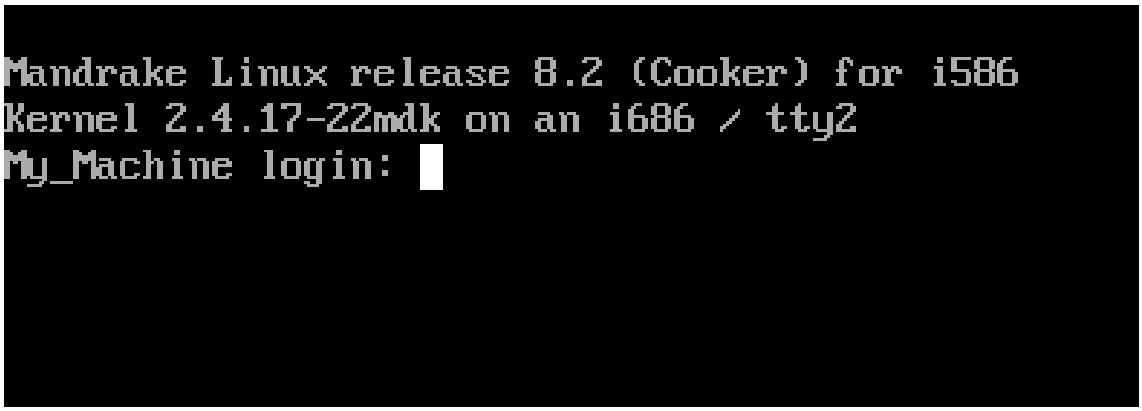

Рисунок 1-2. Сеанс входа в систему в консольном режиме

Вы должны ввести логин в запросе Login: и нажать `Enter`, после чего программа входа (называемая, как ни странно, , *`login`*) отобразит запрос Password:, где вы введете пароль для этого аккаунта. По причине того, что в консольном режиме символы пароля не отображаются, будьте внимательны при вводе своего пароля .... вслепую.

Обратите внимание, что вы можете входить в систему под одним и тем же логином несколько раз в дополнительных ***консолях*** и под *`X`*. Каждый открытый сеанс независим от других, и даже возможно иметь несколько открытых *`X`* сеансов одновременно. По умолчанию, **Mandrake Linux** имеет шесть ***виртуальных консолей*** в дополнении к одной, зарезервированной для графического интерфейса. Вы можете переключаться между ними по клавишам `Alt-F<n>`, где `<n>` это номер консоли, в которую вы хотите переключится. По умолчанию графический интерфейс находится в консоли номер 7. .

Во время инсталляции, *`DrakX`* также спрашивал у вас пароль для очень специального пользователя: `root`. `root` это системный администратор, которым вероятнее всего будете вы сами. Для безопасности вашей системы, аккаунт `root` всегда должен быть защищен хорошим паролем!

Если вы будете постоянно работать под `root`, то очень просто допустить ошибку, которая сделает вашу систему непригодной к использованию; одна единственная ошибка может привести к этому. В особенности, если вы не установили пароль для `root` аккаунта, тогда **любой** пользователь может изменить **любую** часть вашей системы (даже другой операционной системы на вашей машине!). Очевидно, что это не очень хорошая идея.

Стоит обратить внимание, что внутренне система не идентифицирует вас не имени логина, а по уникальному номеру, соответствующему этому логину: *User ID* (UID сокращенно). Подобным образом каждая группа идентифицируется по своему *Group ID* (GID) , но никак не по имени.

1.2. Основы Файла

Файлы это еще одна тема, где *`GNU/Linux`* существенно отличается от *`Windows`* и большинства других ***операционных систем***. Мы рассмотрим здесь большинство явных различий. Для получения большей информации смотрите главу Файловая Система Linux, в которой это все рассмотрено подробно.

Преимущественные различия являются прямым следствием того факта,что *`Linux`* это многопользовательская система: каждый файл это эксклюзивная собственность одного пользователя и одной группы. Еще одна вещь, которую мы не упоминали о пользователях и группах, состоит в том, что каждый из них обладает личным каталогом (называемым ***домашний каталог***). Пользователь является владельцем этого каталога и всех файлов, которые он в нем будет создавать.

Однако, это было бы не очень полезно, если это было бы единственное понятие монопольного использования файла. Но это больше: как владелец файла, пользователь может устанавливать **права** на файлы. Эти права разделяются на три категории пользователей: владелец файла, каждый пользователь, входящий в группу, ассоциированную с файлом (также называемый ***владелец группы (owner group)***), но не являющийся владельцем, и другие, что включает в себя каждого пользователя, отличного от владельца и членов группы владельца.

Существуют три различных варианта прав:

1. *Read* (права на чтение) (`r`): разрешает чтение содержимого файла . Для каталога это обозначает разрешение просмотра его содержимого (то есть файлов в этом каталоге).
2. *Write* (права на запись) (`w`): позволяет изменение содержимого файла. Для каталога права на запись позволяют пользователю добавлять и/или удалять файлы в этом каталоге, даже если пользователь не является владельцем этих файлов.
3. *eXecute* (права на запуск/поиск) (`x`): разрешают запуск файла (как следствие, все выполняемые файлы обычно идут с этими установленными правами). Для каталога, это позволяет пользователю ***пересекать***(traverse) его (что означает вход в каталог или проход через него). Заметьте, что это отделено от прав на чтение: очень может быть, что вы можете проходить через каталог, но не можете читать его содержимое!

Возможны любые комбинации этих прав. Например, вы можете разрешить чтение файла только для себя и запретить для других. Вы даже можете сделать обратное, даже если это не очень логично на первый взгляд... Как владелец файла, вы также можете сменить владельца группы (если и только если вы являетесь членом новой группы), и даже лишить себя прав на файл (собственно поменять его владельца). Естественно, что если вы лишаете себя прав владельца файла, вы потеряете все права на него...

Давайте приведем пример файла и каталога. Ниже находится результат выполнения команды `ls -l` из ***командной строки***:

```
$ 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 команды `ls -l` обозначают (слева направо):

- первые десять символов отображают тип файла и связанные с ним права. Первый символ это тип файла: если это обычный файл, тип будет содержать тире (`-`). Если каталог, вы увидите символ : `d`. Существуют другие типы файлов, о которых мы поговорим в книге *Справочное Руководство*. Девять следующих символов представляют права для данного файла. Здесь вы видите разделение, проведенное между различными пользователями для одного и того же файла: первые три символа обозначают права для владельца файла, следующие три применяются ко всем пользователям, принадлежащим группе, но которые не являются владельцами, и последние три для других. Тире (`-`) обозначает, что права не установлены;
- затем идет номер ссылки на файл. Мы увидим в книге *Справочное Руководство* что уникальный идентификатор файла это не имя, а номер (***номер inode***), и для одного файла на диске возможно иметь несколько названий. Для каталога номер ссылок имеет специальное значение, что мы также обсудим в книге *Справочное Руководство*;
- следующее - это имя владельца файла и имя владельца группы;
- и наконец, показан размер файла (в ***байтах***) и время его последней модификации, идущее за собственно именем файла или каталога.

Давайте теперь поближе посмотрим на права, связанные с каждым из этих файлов: сначала мы отбросим первый символ типа файла и, для файла `a_file`, получим следующие права: `rw-r-----`. Права имеют следующую интерпретацию:

- первые три символа (`rw-`) это права владельца файла, в данном случае queen. Следовательно, queen имеет права читать файл (`r`), изменять его содержимое (`w`) но не может его запускать (`-`);
- следующие три символа (`r--`) применяются к любому пользователю, который не queen, но который является членом группы `users`: такой пользователь сможет читать файл (`r`), но не писать не выполнять его не сможет (`--`);
- последние три символа (`---`) применяются для любого пользователя, который не queen и не член группы `users` : такой пользователь просто не имеет прав на этот файл вообще.

Для каталога `a_directory`, права такие `rwxr-xr--`, и также:

- peter, как владелец каталога, может получит список файлов, лежащих в нем (`r`), добавлять или убивать файлы в этом каталоге (`w`), и он может пересекать его (`x`);
- каждый пользователь, отличный от peter, но член группы `users`, может видеть список файлов в этом каталоге (`r`), но не может ни удалять ни добавлять файлов в нем (`-`), но сможет пересекать его (`x`);
- каждый другой пользователь сможет только получить список содержимого этого каталога (`r`), но это и все. Он не сможет даже войти в каталог.

Есть **одно** исключение из этих правил: `root`. `root` может изменять атрибуты (права, владельца и владельца группы) всех файлов, даже если он не является владельцем. Это означает, что он также может предоставлять себе монопольное использование! Он может читать файлы, на которые у него нет права на чтение , пересекать каталоги, к которым у него обычно нет прав доступа и так далее. И если он испытывает недостаток прав, ему достаточно только добавить их...

И последнее. Нет никакой разницы между именами файлов в *UNIX* и *Windows* мирах. Что касается *UNIX*, то она предоставляет большую гибкость и имеет меньше ограничений:

- имя файла может содержать любой символ (кроме символа ASCII 0, который является концом строки и /, который обозначает разделитель каталога), даже непечатные. Более того, *UNIX* является чувствительной к регистру символов : файлы `readme` и `Readme` являются разными файлами, потому что `r` и `R` читаются как два разных **символа** под системами на базе *UNIX*.
- Как вы могли заметить, имя файла не обязано иметь расширение, если только вам так не нравится больше. Расширения файлов не обозначают содержимого файлов под *GNU/Linux*, и не делают этого на любой операционной системе в этом отношении. Так называемые “расширения файлов” являются очень удобными и только. Точка (`.`) под *UNIX* это просто один из символов. Нужно заметить, что файлы, у которых имена начинаются с точки под *UNIX* являются “скрытыми (hidden) файлами”;

1.3. Процессы

Процесс определяет запрос выполняемой программы и ее ***окружение***. Как и для файлов, мы только укажем наиболее важные различия; вам прийдется обратиться к книге *Справочное Руководство* за более подробным обсуждением предмета.

Наиболее важное различие снова напрямую относится к концепции пользователя: каждый процесс выполняется с правами пользователя, который его запустил. Внутри себя система различает процессы уникальным путем: по номеру. Этот номер называется *process ID*, или PID. . Из этого PID, система знает, среди других вещей, кто (какой пользователь, что это) запустил процесс. Затем, ей только нужно проверить процесс на “достоверность (validity)”. Следовательно, если мы возьмем наш пример для `a_file`, процесс, запущенный пользователем peter будет иметь возможность открывать этот файл в ***режиме read-only (только для чтения)***, но не в ***режиме read-write (чтение-запись)***, так как права файла этого не позволяют. И снова исключением является `root`...

Благодаря этому, *GNU/Linux* является виртуально защищенной против вирусов. Чтобы работать, вирусы должны заражать выполняемые файлы. Как пользователь, вы не имеете никаких прав на запись уязвимых системных файлов, поэтому риск существенно уменьшен. В дополнение к этому факту, вообще говоря, вирусы очень редки в мире *UNIX*. Существует меньше дюжины известных вирусов для *Linux*, и они безобидны когда запускаются обычным пользователем. Только один пользователь может разрушить систему активацией этих вирусов, и снова это `root`.

Достаточно интересно, что антивирусное программное обеспечение существует для *GNU/Linux*, но оно предназначено для файлов *DOS / Windows*... Смысл его в том, что все чаще и чаще вы будете встречать файловые сервера под *GNU/Linux*, обслуживающие машины с *Windows* с помощью программного пакета *Samba*.

Linux делает легким контроль за процессами. Один из путей контроля это сигналы. Сигналами можно, например, приостановить процесс или убить его. Просто пошлите соответствующий сигнал процессу и все. Однако вы ограничены тем, что можете посылать сигналы только своим процессам, а процессам других пользователей нет. Исключение из правила это `root`. Да, опять он! В Гл. 6, вы узнаете как получить PID процесса и посылать ему сигналы.

1.4. Небольшое Введение в Командную Строку

Командная строка это самый прямой путь посылать команды машине. Если вы используете командную строку *GNU/Linux*, то вскоре вы найдете ее наиболее мощной и искусной, чем другие командные запросы, которые вы возможно использовали. Смысл здесь в том, что вы имеете прямой доступ не только ко всем приложениям *X*, но также к тысячам утилит в консольном режиме (в противоположность графическому режиму), которые не имеют графического эквивалента, или многочисленные опции и возможные комбинации которых были бы труднодоступны на форме из кнопок и меню.

Но, общепризнанно, что тут требуется некоторая помощь, чтобы приступить. Первая задача это запустить эмулятор терминала. Вы можете найти его в меню Терминалы вашего *GNOME* или *KDE* . Затем, выберите какой нравится, например Konsole или XTerm. У вас также есть иконка, которая ясно определяет его на панели. (Рис. 1-3).

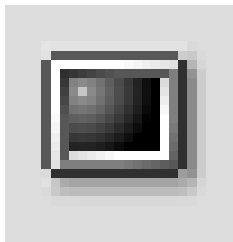

Рисунок 1-3. Иконка терминала на панели KDE

Когда вы запускаете этот эмулятор терминала, вы реально используете *shell*. Это имя программы, с которой вы работаете. Вы окажетесь перед ***запросом***:

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

Здесь предполагается, что имя вашего пользователя queen и ваша машина называется `localhost` (это в случае, если ваша машина не входит в какую-либо сеть). Обычно, после запроса находится пространство для того, чтобы вы набирали то, что вам необходимо. Заметьте, что когда вы заходите как `root`, символ запроса `$` меняется на `#` (это верно для конфигурации по умолчанию, но вы можете изменить все такие детали в *GNU/Linux*). Чтобы перейти в `root`, наберите `su` после запуска *shell*.

```
# Введите пароль root; его не будет видно на экране
[queen@localhost queen]$ su
Password:
# exit вернет вас в нормальны пользовательский аккаунт
[root@localhost queen]# exit
[queen@localhost queen]$
```

Во всех других местах книги, запрос будет символически отображаться знаком `$`, независимо от того, что вы нормальный пользователь или `root`. Вам будет сказано когда нужно быть `root` для выполнения команды, поэтому запомните команду `su`. Кстати, когда символ `#` помещен в начало строки кода, он обозначает комментарий.

Когда вы ***запускаете*** *shell* первый раз, вы обычно попадаете в домашний каталог. Чтобы узнать где вы сейчас находитесь, наберите `pwd` (то означает *Print Working Directory (напечатать рабочий каталог)*):

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

Теперь мы рассмотрим несколько основных команд и скоро вы найдете, что вы не можете жить без них! не сможете

1.4.1. cd: Сменить Каталог

Команда `cd` выглядит как *DOS*', с некоторыми дополнениями. Она делает то, что обозначает ее название - сменяет рабочий каталог. Вы можете использовать `.` и `..`, которые обозначают соответственно текущий и родительский каталоги. Вызвав `cd` без параметров, вы вернетесь в домашний каталог. Набор `cd -` вернет вас в последний каталог, в котором вы находились. И наконец, вы можете выбрать домашний каталог пользователя peter, набрав `cd ~peter` (`~` сама по себе обозначает ваш собственный домашний каталог). Заметьте, что как обычный пользователь, вы не можете перейти в домашний каталог другого пользователя (только если он авторизован для вас или это каталог с конфигурацией систему по умолчанию), если вы не `root` конечно, тогда давайте станем `root` и попрактикуемся:

```
$ 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 и станем снова обычным пользователем.

1.4.2. Некоторые Переменные Окружения и Команда echo

Все процессы имеют свои ***переменные окружения*** и *shell* позволяет вам их посмотреть непосредственно с помощью команды `echo`. Вот некоторые интересные переменные:

1. `HOME`: эта переменная окружения содержит строку, которая представляет ваш домашний каталог.
2. `PATH`: эта переменная держит список всех каталогов, в которых *shell* должна искать запускаемые программы, когда вы пишите 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 держит ваш ID пользователя.
5. `PS1`: эта переменная содержит определения для вашего запроса. Часто это комбинация специальных последовательностей. Вы можете прочитать bash(1) ***страницу руководства (manual page)*** для получения подробной информации.

Чтобы *shell* напечатала значения переменной, вы должны разместить `$` перед ее именем. Здесь вам поможет команда `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 вы можете увидеть, the *shell* подставляет значение переменной прежде чем выполнить команду. Иначе наша `cd $HOME` не работала бы здесь. Фактически, shell сначала заменила `$HOME` ее значением, `/home/queen`. Следовательно, строка становится такой `cd /home/queen`, как мы и хотели. То же самое происходит для `echo $USERNAME` и так далее.

1.4.3. cat: Вывод Содержимого Одного или Более Файлов на Экран

Нечего сказать больше, так как эта команда делает только это: выводит содержимое одного или более файлов на стандартный вывод, обычно на экран:

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

1.4.4. less: Пейджер

Ее название это игра слов по отношению к первому из пейджеров под *UNIX*, который назывался `more`. ***Пейджер*** это программа, которая позволяет пользователю просматривать длинные файлы по страницам (более аккуратно, экран за экраном). Мы говорим здесь о `less` вместо `more` потому, что она гораздо интуитивнее в использовании. Вы можете использовать `less` для просмотра больших файлов, которые не помещаются в экран. Например:

```
less /etc/termcap
```

Для перемещения по этому файлу, используйте клавиши со стрелками вверх и вниз. Для выхода используйте **q**. Однако, `less` умеет гораздо больше, чем это. Напишите `h` для помощи и взгляните. Но, в любом случае, целью этого раздела является научить вас читать длинные файлы, так что нашей цели мы уже достигли **:-)**

1.4.5. ls: Вывод Списка Файлов

Команда `ls` (*LiSt*) это эквивалент команды *DOS* `dir`, но умеет делать гораздо больше. В значительной степени это следствие того, что файлы тоже могут делать больше. Синтаксис команды `ls` такой:

```
ls [options] [file|directory] [file|directory...]
```

Если в командной строке не указан каталог или файл, `ls` выведет список файлов в текущем каталоге. У команды очень много опций, но некоторые из них мы здесь поясним:

- `-a`: составляет список всех файлов, включая ***скрытые файлы*** (помните, что в *UNIX*, скрытые файлы это те, у кого имя начинается с `.`); опция `-A` составляет список "почти" всех файлов, что значит каждый файл, который вывела бы опция `-a` кроме "." и "..";
- `-R`: составляет список рекурсивно, то есть всех файлов и каталогов, упомянутых в командной строке
- `-s`: выводит размер файла в килобайтах после каждого файла;
- `-l`: выводит дополнительную информацию а файлах;
- `-i`: выводит номер inode (уникальный номер файла в файловой системе, см. главу Файловая Система Linux) после каждого файла;
- `-d`: обрабатывает каталоги в командной строке как будто они были нормальными файлами, вместо вывода их содержимого.

Несколько примеров:

- `ls -R`: рекурсивно выводит список текущего каталога;
- `ls -is images/ ..`: выводит список файлов в каталоге `images/` и родительском каталоге для данного. Затем выводит для каждого файла номер inode и размер в килобайтах;
- `ls -al images/*.png`: выводит список всех файлов, чьи имена заканчиваются на `.png` (включая любой скрытый файл) в каталоге `images/`. Заметьте, что это включает также файл `.png`, если таковой имеется.

1.4.6. Полезные Сочетания Клавиш

Есть много вариантов нажатий клавиш и их главное преимущество состоит в том, что они экономят кучу времени при наборе текста. Этот раздел подразумевает, что вы используете *`shell`* по умолчанию, поставляемую с **Mandrake Linux**, *`bash`*, но эти горячие клавиши должны работать и в других shell тоже

Первое: клавиши со стрелками. *`bash`* хранит историю предыдущих команд, которые можно посмотреть по клавишам со стрелками. Вы можете прокручивать максимальное число строк, определенное в переменной окружения `HISTSIZE`. Более того, история постоянна от одного сеанса к другому, таким образом вы не потеряете команд, которые вы набирали в предыдущих сеансах.

Стрелки влево и вправо перемещают курсор влево и вправо текущей строки, и таким образом вы можете редактировать команды. Еще удобно для редактирования: `Ctrl+A` и `Ctrl+E`, например, переместят вас соответственно в начала и конец текущей строки. Клавиши `Backspace` и `Del` работают как обычно. Эквивалент `Backspace` это `Ctrl+H` и эквивалент `Del` это `Ctrl+D`. `Ctrl+K` удалит всю строку от позиции курсора до конца строки, и `Ctrl+W` удалит слово перед курсором.

Нажатие `Ctrl+D` на пустой строке закрывает текущий сеанс, что немного короче, чем набирать `exit`. `Ctrl+C` прервет текущую запущенную программу, если только вы не редактируете вашу командную строку, в этом случае это отменит редактирование и вернет вас в запрос. `Ctrl+L` очищает экран.

И наконец, вариант `Ctrl+S` и `Ctrl+Q`: эти нажатия клавиши соответственно приостанавливают и восстанавливают поток символов на терминале. Они используются очень редко, но может так случиться, что вы по ошибке набрали `Ctrl+S` (ведь **S** и **D** находятся слишком близко на клавиатуре ...). Итак, если вы начали набор, но не увидели ни одного символа на *`Terminal`*, попробуйте `Ctrl+Q` сначала и остерегайтесь : все набранные после `Ctrl+S` символы нежелательны, и по `Ctrl+Q` будет выведено на экран все сразу.

Глава 2. Диски и Разделы

В этой главе находится информация для пользователей, которые выполняют инсталляцию **Mandrake Linux** в режиме эксперт, или для тех, кто хотел бы узнать больше технических деталей относительно своей компьютерной системы.

В этом разделе представлено полное описание схемы разделения дисков *PC*, поэтому он будет наиболее полезен если вы вручную настраиваете разделы вашего жесткого диска. Инсталлятор сделает все за вас автоматически, поэтому нет особой нужды понимать все, что происходит во время инсталляции. Однако, если вы собираетесь проводить инсталляцию в режиме эксперт, то информация данного раздела становится весьма важной.

2.1. Структура Жесткого Диска

Диск физически разделен на небольшие сектора. Последовательность секторов может формировать раздел. Грубо говоря, вы можете создавать столько разделов, сколько пожелаете; каждый из них будет виден как отдельный жесткий диск.

2.1.1. Сектора

Упрощенно жесткий диск является просто последовательностью *секторов*. Сектор - это самый маленький модуль данных на жестком диске и его размер обычно 512 байт. Сектора на жестком диске (n) пронумерованы от (0) до (n-1).

2.1.2. Разделы

Использование множественных разделов позволяет вам создавать много виртуальных жестких дисков на вашем реальном физическом диске. В этом есть много преимуществ:

- Различные операционные системы используют различные структуры диска (называемые файловыми системами): это относится и к *Windows* и к *GNU/Linux*. Наличие множества разделов на жестком диске позволяет устанавливать различные операционные системы на один физический жесткий диск.
- Из соображений производительности, в одной операционной системе могут быть различные диски с разными файловыми системами на них и использоваться для различных вещей. Например, *GNU/Linux* требует второй раздел под названием “swap”. Этот раздел используется виртуальным менеджером памяти под виртуальную память.
- Наконец, весьма полезно разносить различные части вашей операционной системы по разным разделам, даже если они имеют похожую файловую систему. В простейшей конфигурации, вы можете разделить ваши файлы по двум разделам: на одном хранить ваши личные данные, а на другом программы. Это позволит вам обновлять вашу систему, полностью стирая раздел, на котором содержатся программы и сохранять при этом раздел с данными нетронутым.
- Физические ошибки на жестком диске (если они появляются, конечно) обычно располагаются на смежных секторах и не рассеиваются поперек диска. Размещение ваших файлов на различных разделах ограничит потерю данных в случае физического повреждения жесткого диска.

Обычно тип раздела определяет файловую систему, размещаемую на нем. Каждая операционная система признает одни типы, но не признает другие. Почитайте главу о файловых системах *GNU/Linux* в книге *Справочное Руководство*, чтобы узнать больше.

2.1.3. Определение Структуры Диска

2.1.3.1. Самый простой путь

Простейшая конфигурация будет состоять из двух разделов: один для swap-пространства, второй - для файлов [1].

Ключевое правило для размера swap раздела - он должен иметь двойной размер вашей оперативной (RAM) памяти. Однако, в системах с большим количеством памяти (>512 MB), это правило уже не так критично и swap может иметь не такой большой размер.

2.1.3.2. Другая Общая Схема

Отделить данные от программ. Для большей эффективности определяют третий раздел, который называется "root" и имеет метку /. В нем содержатся программы, необходимые для загрузки системы, и основные программы для обслуживания системы.

Итак, мы можем определить четыре раздела:

Swap

Раздел типа `swap`, чей размер занимает примерно два размера физической оперативной памяти.

Root: /

Самый важный раздел. Кроме того, что он содержит самые важные данные и программы, он еще является и точкой монтирования для других разделов.

По размерам root раздел весьма лимитирован: выделить ему 300MB в общем случае будет достаточно. Однако, если вы планируете устанавливать коммерческие приложения, которые обычно размещаются в `/opt`, вам прийдется увеличить размер root раздела. Вместо этого вы можете создать отдельный раздел для `/opt`.

Статические данные: `/usr`

Большинство пакетов устанавливают основную массу своих запускаемых программ и файлов данных в `/usr`. Преимущество создания такого отдельного раздела состоит в том, что вы сможете легко открыть доступ к нему для других машин в сети.

Рекомендуемый размер зависит от количества пакетов, которые вы собираетесь установить, и может варьировать от 100MB для небольшой инсталляции до нескольких GB для полной инсталляции. Компромиссным решением будет выделить один или два GB (в зависимости от размера вашего диска), которых обычно достаточно.

Домашние каталоги: `/home`

Этот каталог содержит персональные каталоги всех пользователей машины. Здесь же иногда размещаются каталоги HTTP сервера для web-браузеров и FTP сервера для передачи файлов. [2]

Размер раздела зависит от количества пользователей (или сервисов), размещенных на нем и их потребностей.

Как вариант, можно не создавать отдельного раздела для файлов `/usr`: `/usr` будет просто каталогом внутри раздела root /.

1. файловая система, которая сейчас используется для файлов *GNU/Linux* называется `ext2`
2. В последнее время эти каталоги размещаются чаще в `/var/www` и `/var/ftp`

2.1.3.3. Экзотические Конфигурации

Когда ваша машина настраивается для использования в специфических целях, например в качестве web-сервера или файервола, требования радикально отличаются от тех, которые подходят для стандартной рабочей станции. Для FTP сервера скорее всего потребуется большой отдельный раздел `/var/ftp`, а `/usr` будет относительно меньше. В таких ситуациях стоит хорошо подумать, прежде чем начинать разбивать диск на разделы и проводить инсталляцию.

Если вы уже используете систему и пришли к тому, что нужно изменить размеры и схему разделов диска, то знайте, что можно изменить размеры большинства разделов без переинсталляции и потери данных. Почитайте об этом в разделе *Управление Разделами Диска* книги *Стартовое Руководство Пользователя*.

При некотором количестве опыта вы даже сможете переносить переполненный раздел на новый жесткий диск.

2.2. Соглашения в Названиях Дисков и Разделов

GNU/Linux использует логический метод названия разделов. Во-первых, при нумерации разделов она игнорирует типы файловых систем любого из разделов, который может у вас быть. Во-вторых, она называет разделы согласно тому, на каком диске они размещаются. Вот как называются диски:

- устройства IDE primary master и primary slave (будь они жесткими дисками, устройствами CD-ROM или чем-то еще) называются `/dev/hda` и `/dev/hdb` соответственно;
- на secondary интерфейсе, master называется `/dev/hdc`, а slave - `/dev/hdd`
- если в вашем компьютере имеются другие интерфейсы IDE (например, IDE интерфейс на некоторых Soundblaster (звуковых картах), диски будут называться `/dev/hde`, `/dev/hdf`, и т.д. У вас также могут быть дополнительные IDE интерфейсы, если у вас есть на материнской плате RAID карты или RAID чипы.
- SCSI диски называются `/dev/sda`, `/dev/sdb`, и т.д., в порядке их появления в цепи SCSI (в зависимости от возрастания *ID*). Устройства SCSI CD-ROM называются `/dev/scd0`, `/dev/scd1`, всегда в порядке их появления в цепи SCSI.

Разделы именуются после диска, на котором они найдены, следующим образом (в нашем примере мы используем случай с разделами на диске primary master IDE):

- primary (или extended) разделы называются `/dev/hda1` затем `/dev/hda4`, если есть;
- логические разделы, если они есть, называются `/dev/hda5`, `/dev/hda6`, и т.д. в порядке их появления в таблице логических разделов.

Итак, *GNU/Linux* будет давать имена разделам так:

Рисунок 2-1. Первый пример названия разделов под GNU/Linux

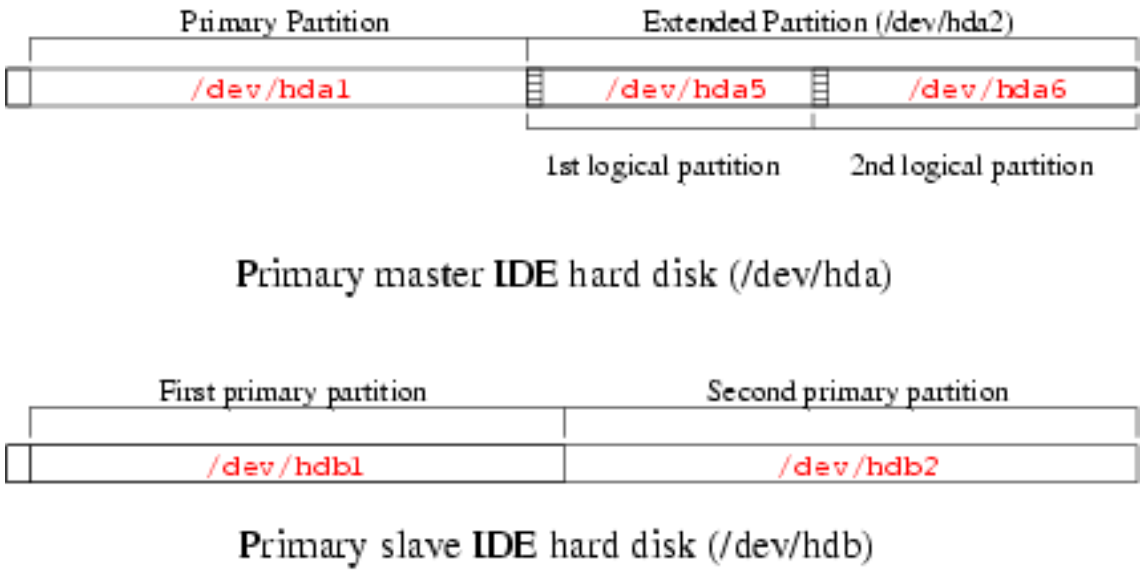

Рисунок 2-2. Второй пример названия разделов под GNU/Linux

Имея все эти сведения под рукой, вы сможете называть различные разделы и жесткие диски, когда вам будет необходимо. Кроме того, вы увидите, что *GNU/Linux* дает имена разделам даже тогда, когда она не знает изначально как ими управлять (она игнорирует тот факт, что они не являются родными *GNU/Linux* разделами).

В текущих 2.4 ядрах, **Mandrake Linux** использует Linux DevFS (Device File System) (`http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html`). Эта система гарантирует полную совместимость с описанной выше схемой, но в будущем это может измениться. Фактически, каждое устройство динамически добавляется в систему, как только оно становится доступным или необходимым.

Например, первый жесткий диск IDE теперь выглядит так:

```
[root@localhost root]# ls -l /dev/hda
lr-xr-xr-x    1 root     root           32 Sep  2 17:14 /dev/hda
              -> ide/host0/bus0/target0/lun0/disc
```

Глава 3. Введение в Командную Строку

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

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

3.1. Утилиты Обработки Файлов

В данном контексте обработка файлов обозначает их копирование, перемещение и удаление. Позже мы рассмотрим способы изменения их атрибутов (владелец, права).

3.1.1. mkdir, touch: Создание Пустых Каталогов и Файлов

`mkdir` (*MaKe DIRectory* (создать каталог)) используется для создания каталогов. Ее синтаксис простой:

```
mkdir [options] <directory> [directory ...]
```

Только один параметр ничего не обозначает: `-p` . Он делает две вещи:

1. с ним команда будет создавать родительские каталоги, если они не существовали ранее. Без этого параметра `mkdir` просто может упасть, пожаловавшись, что упомянутые родительские каталоги не существуют;
2. с ним команда будет возвращаться в командную строку без сообщений, если каталог, который вы хотите создать уже есть. Обратно, если вы не укажете параметра `-p`, `mkdir` вернет ошибку, сообщив что каталог уже имеется.

Вот несколько примеров:

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

Изначально, команда `touch` не предназначалась для создания файлов, а только для обновления доступа к файлу и изменения времени создания файла [1]. Однако, `touch` будет создавать пустые файлы, если их не существовало ранее. Синтаксис такой:

```
touch [options] file [file...]
```

Итак, запуск команды:

```
touch file1 images/file2
```

создаст пустой файл с именем `file1` в текущем каталоге и пустой файл `file2` в каталоге `images`, если файлы еще не существовали до этого.

3.1.2. rm: Удаление Файлов и Каталогов

Команда `rm` (*ReMove*) заменяет команды *DOS* `del` и `deltree`, и имеет много дополнительных параметров . Ее синтаксис следующий:

```
rm [options] <file|directory> [file|directory...]
```

Опции включают:

- `-r`, или `-R`: удаляет рекурсивно. Этот параметр **обязательный** для удаления каталогов, пустых или нет. Однако вы можете также использовать `rmdir` для удаления пустых каталогов.

1. В *UNIX* есть три различных временных метки для каждого файла: время последнего обращения к файлу (`atime`), то есть последний раз когда файл открывался на чтение или запись; последняя дата изменения атрибутов inode (`mtime`); и время, когда изменялось последний раз **содержимое** файла. (`ctime`).

- `-i`: запросить подтверждение перед каждым удалением. Обратите внимание, что по умолчанию в **Mandrake Linux**, `rm` является ***alias (алиасом)*** команды `rm -i`, в целях безопасности (подобные алиасы есть для команд `cp` и `mv`). Вы можете изменить свое мнение относительно полезности этих алиасов. Если вы захотите удалить их, отредактируйте ваш файл `~/.bashrc` и добавьте строку: `unalias rm cp mv`.
- `-f`, обратная опции `-i`, форсированное удаление файлов или каталогов даже если пользователь на имеет права на запись файлов. [2].

Несколько примеров:

- `rm -i images/*.jpg file1`: удаляет все файлы с окончанием `.jpg` в названии в каталоге `images` и удаляет файл `file1` в текущем каталоге с запросом подтверждения. Ответьте **y** чтобы подтвердить удаление, **n** чтобы отменить.
- `rm -Rf images/misc/ file*`: удаляет без вопросов весь каталог `misc/` в каталоге `images/` вместе со всеми файлами в текущем каталоге, чьи имена начинаются с `file`.

`rm` удаляет файлы **безвозвратно**. Не существует способа их восстановить! Не пренебрегайте использовать параметр `-i`, чтобы не удалить что-либо по ошибке.

3.1.3. mv: Перемещение или Переименование Файлов

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

```
mv [options] <file|directory> [file|directory ...] <destination>
```

Некоторые параметры:

- `-f`: форсировать операцию –, никаких предупреждений, если существующие файлы перезаписываются.
- `-i`: напротив, спрашивает подтверждение у пользователя, перед тем как переписать существующий файл.
- `-v`: ***verbose (подробный)*** режим, сообщает о всех изменениях или активности.

Несколько примеров:

- `mv -i /tmp/pics/*.png .`: переносит все файлы из каталога `/tmp/pics/` с окончаниями в именах `.png` в текущий каталог (`.`), но спрашивает подтверждение, перед тем как переписать любой файл.
- `mv foo bar`: переименовывает файл `foo` в `bar`. Если уже существует каталог `bar`, эффектом этой команды будет перенос всего каталога `foo` (самого каталога и всех его файлов и каталогов в нем рекурсивно) в каталог `bar` .
- `mv -vf file* images/ trash/`: переносит без подтверждения все файлы текущего каталога, имена которых начинаются с `file`, вместе со всем каталогом `images/` в каталог `trash/`, и показывает информацию о каждой операции.

2. Для пользователя достаточно иметь права записи в каталоге, чтобы удалять файлы в нем, даже если он не владелец файлов, которые там лежат.

3.1.4. cp: Копирование файлов и каталогов

`cp` (*CoPy*) замещает команды *DOS* `copy` и `xcopy` и имеет множество дополнительных параметров. Ее синтаксис:

```
cp [options] <file|directory> [file|directory ...] <destination>
```

`cp` имеет много параметров. Вот основные из них:

- `-R`: рекурсивное копирование; **обязательно** для копирования каталогов, даже пустых.
- `-i`: запрос подтверждения перед перезаписью любого файла, который может быть перезаписан.
- `-f`: обратный `-i`, заменяет любой существующий файл без подтверждения.
- `-v`: подробный (verbose) режим, отображает все действия, производимые `cp`.

Несколько примеров:

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

3.2. Обработка Атрибутов Файлов

Ряд команд, показанных здесь используется для изменения владельца или группы файла или его прав. Мы рассматривали различные права в главе Базовые Концепции Системы `UNIX`.

3.2.1. chown, chgrp: Изменение Владельца или Группы Одного или Нескольких Файлов.

Синтаксис команды `chown` (*CHange OWNer* (изменить владельца)) следующий:

```
chown [options] <user[.group]> <file|directory> [file|directory...]
```

Параметры включают:

- `-R`: рекурсивно. Для изменения владельца всех файлов и каталогов в данном каталоге.
- `-v`: подробный(verbose) режим. Отображает все действия, производимые `chown`; сообщает у каких файлов был изменен или не изменен владелец в результате выполнения команды.
- `-c`: типа `-v`, но сообщает только какие файлы были изменены .

Несколько примеров:

- `chown nobody /shared/book.tex`: изменяет владельца файла `/shared/book.tex` на `nobody`.
- `chown -Rc queen.music *.mid concerts/`: изменяет принадлежность 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 файлах, которые изменены командой.

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

```
chgrp [options] <group> <file|directory> [file|directory...]
```

Параметры этой команды такие-же как и для `chown` и она используется очень похоже. Так, команда:

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

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

3.2.2. chmod: Изменение Прав Файлов и Каталогов

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

```
chmod [options] <change mode> <file|directory> [file|directory...]
```

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

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

   Категорий может быть одна или несколько:

   - `u` (*User*, (Пользователь) права владельца);
   - `g` (*Group*, (Группа) права владельца группы);
   - `o` (*Others*, (Другие) права для "остальных").

   Если категория не определена, изменения будут приняты для всех категорий. `+` устанавливает права, `-` убирает права и `=` устанавливает права. Вообщем, права бывают следующие:

   - `r` (*Read* (Чтение));
   - `w` (*Write* (Запись)) или;
   - `x` (*eXecute* (Выполнение)).

Главные параметры полностью похожи на параметры команд `chown` или `chgrp`:

- `-R`: изменяет права рекурсивно.
- `-v`: подробный (verbose) режим. Отображает действия, производимые с каждым файлом.
- `-c`: также как `-v`, но показывает только файлы, на которые подействовала команда.

Примеры:

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

3.3. Шаблоны Подстановки Shell

Возможно, вы уже использовали символы ***подстановки***, не зная как это называется и что это такое. Когда вы выбираете файл в *Windows* или когда вы ищете файл, вы используете * для соответствия случайной строке. Например, *.txt соответствует всем файлам с окончаниями .txt в именах. Мы тоже интенсивно использовали их в последнем разделе. Но существует гораздо больше подстановок, чем только *.

Когда вы набираете команду типа ls *.txt и нажимаете Return (ввод), задача поиска файлов, совпадающих с критерием *.txt решается не только командой ls но и самой *shell*. Здесь требуется небольшое объяснение о том, как командная строка интерпретируется в *shell*. Когда вы пишете:

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

командная строка сначала разделяет слова (ls и *.txt в этом примере). Когда оболочка видит * в слове, она будет интерпретировать все слово как шаблон постановки и заменять его именами всех совпадающих файлов. Следовательно, команда, перед тем как оболочка выполняет ее, становится такой ls readme.txt recipe.txt, что дает ожидаемый результат. Другие символы заставляют оболочку реагировать следующим образом:

- ?: соответствует одному и только одному символу, независимо от того чем этот символ является;
- [...]: соответствует одному символу, найденному в скобках. Символы можно указать как диапазон символов (то есть 1-9) или ***дискретные значения*** , или даже и то и другое. Пример: [a-zBE5-7] будет соответствовать всем символам между a и z, B, E, 5, 6 или 7;
- [!...]: соответствует любому символу, **не** находящемуся в скобках. Например, [!a-z], будет соответствовать любому символу который не является буквой в нижнем регистре[3];
- {c1,c2}: соответствует c1 или c2, где c1 и c2 также шаблоны подстановки, которые обозначают что вы можете написать {[0-9]*,[acr]} например.

Вот некоторые шаблоны и их значения:

- /etc/*conf: все файлы в каталоге /etc с окончаниями в именах conf. Это может соответствовать /etc/inetd.conf, /etc/conf.linuxconf, **а также** /etc/conf если такой файл существует. Помните, что * может соответствовать пустой строке.
- image/{cars,space[0-9]}/*.jpg: все файлы, заканчивающиеся на .jpg в каталогах image/cars, image/space0, (...), image/space9, если эти каталоги существуют.
- /usr/share/doc/*/README: все файлы с именем README во всех каталогах /usr/share/doc непосредственно. Это будет соответствовать /usr/share/doc/mandrake/README, например, но не соответствовать /usr/share/doc/myprog/doc/README.
- *[!a-z]: все файлы, имена которых **не** заканчиваются буквой в нижнем регистре в текущем каталоге.

3.4. Переназначения и Каналы

3.4.1. Немного Подробнее о Процессах

Для понимания того, что такое переназначения и каналы, нам нужно пояснить понятие о процессах, которое еще не рассматривалось. Каждый процесс *UNIX* (что касается и графических приложений) открывает минимум три файловых дескриптора: стандартный ввод, стандартный вывод и стандартный поток ошибок. Их соответствующие номера это 0, 1 и 2. В общем случае, эти три дескриптора ассоциируются с терминалом, из которого процесс был запущен. Задача перенаправлений и каналов - перенаправить эти дескрипторы. Примеры в данном разделе помогут вам лучше понять эту концепцию.

3. Осторожно! Если это работает *GNU/Linux*, то вполне может не работать под другими *UNIX*-подобными операционными системами. Это зависит от **сопоставления порядка** (collating order). На некоторых системах [a-z] будет соответствовать a, A, b, B, (...), z. Не говоря уже о том, что некоторые языки имеют подчеркнутые символы...

3.4.2. Перенаправления

Представьте, например, что вам нужен список файлов с окончаниями в именах на `.png` [4] в каталоге `images`. Этот список очень длинный, и поэтому вы можете пожелать сохранить его в файл для того, чтобы потом просмотреть его на досуге. Вы можете выполнить следующую команду:

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

Это значит, что стандартный вывод этой команды (`1`) перенаправлен (`>`) в файл с именем `file_list`. Оператор `>` является оператором перенаправления вывода. Если файл для перенаправления не существует, он будет создан, а если существует, то его предыдущее содержимое будет перезаписано. Однако, дескриптор по умолчанию перенаправленный этим оператором, сам является стандартным выводом и его не обязательно указывать в командной строке. Таким образом можно написать проще:

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

и получить тот же результат. Затем вы можете посмотреть файл, используя просмотрщик текстовых файлов, например `less`.

Теперь представьте, что вам нужно узнать, сколько таких файлов существует. Вместо того, чтобы считать их вручную, вы можете использовать утилиту под названием `wc` (*Word Count* (подсчет слов)) с параметром `-l`, которая выведет в стандартный вывод количество строк в файле. Одно из решений следующее:

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

и это даст ожидаемый результат. Оператор `<` является оператором перенаправления ввода, а перенаправляющий дескриптор по умолчанию есть стандартный ввод, то есть `0`, и вы может просто написать:

```
wc -l <file_list
```

Теперь предположим, что мы хотим убрать все “расширения” файлов и положить результат в другой файл. Одним из инструментов для выполнения этого является `sed` (*Stream EDitor* (редактор потока)). Вы просто перенаправляете стандартный ввод `sed` в файл `file_list` и перенаправляете его вывод в результирующий файл `the_list`:

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

и ваш список создан и готов для просмотра на досуге любым просмотрщиком.

Очень полезно перенаправлять стандартный поток ошибок. Например, вы хотите узнать, какие каталоги в `/shared` для вас недоступны: одно из решений это создать список этого каталога и перенаправить ошибки в файл, не отображая при этом стандартного вывода:

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

что обозначает, что стандартный вывод был перенаправлен (`>`) в `/dev/null`, специальный файл, в котором теряется все, что в него пишется, (то есть не отображается стандартный вывод) и стандартный канал ошибок (`2`) перенаправлен (`>`) в файл `errors`.

3.4.3. Каналы

Каналы (pipes, трубы) служат для комбинации перенаправления ввода и вывода. Принцип подобен физической трубе, что следует из названия: один процесс посылает данные в конец трубы, а другой процесс читает данные на другом конце трубы. Оператор канала это `|`. Давайте вернемся к примеру со списком файлов, описанному выше. Допустим, что вы хотите найти без сохранения списка во временный файл, сколько соответствующих файлов находится в нем . Тогда вы можете использовать следующую команду:

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

что значит, что стандартный вывод команды `ls` перенаправлен в стандартный ввод команды `wc`. Это приведет вас к желаемому результату.

4. Возможно, вы подумаете, что безумно говорить так “файлы с окончаниями на `.png`”, вместо того, чтобы сказать “PNG картинки”. Однако, напомним еще раз, что файлы под *UNIX* имеют расширения только по соглашению: расширения ни в коем случае не определяют тип файла. Файл, имеющий окончание `.png`, вполне может быть картинкой JPEG, файлом приложения, текстовым файлом или любым другим. Кстати, то же самое и под *Windows* !

Вы также можете прямо сложить вместе список файлов без расширений, используя следующую команду:

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

или, если вам надо просмотреть список без сохранения в файл:

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

Каналы и перенаправления предназначены не только для текста, который можно читать. Например, следующая команда, посланная из *Terminal*:

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

выдаст скриншот вашего рабочего стола в файл `my_desktop.png` [5] в вашем домашнем каталоге.

3.5. Завершение Командной Строки

Завершение является очень удобной функциональной возможностью и все современные *shells* (включая *bash*) имеют ее. Ее задача облегчить пользователю работу насколько возможно. Чтобы продемонстрировать завершение, приведем пример.

3.5.1. Пример

Предположим, в вашем домашнем каталоге содержится файл `file_with_very_long_name_impossible_to_type` и вы хотели бы его просмотреть. Предположим, в этом же каталоге у вас есть другой файл с названием `file_text`. Вы находитесь в домашнем каталоге, тогда напечатайте следующее:

```
$ less fi<TAB>
```

(то есть напечатайте `less fi`, а затем нажмите клавишу `TAB`). *shell* развернет командную строку для вас:

```
$ less file_
```

а также выдаст список возможных вариантов выбора (из конфигурации shell по умолчанию, что можно изменить). Затем напишите следующее ключевое выражение:

```
less file_w<TAB>
```

и *shell* развернет командную строку до желаемого результата:

```
less file_with_very_long_name_impossible_to_type
```

Все, что вам нужно теперь будет сделать - это нажать клавишу `Enter` для подтверждения и прочитать файл.

3.5.2. Другие Способы Завершения Строки

Клавиша TAB не является единственным способом активировать завершение строки, хотя это наиболее распространено. По общему правилу, слово, которое будет завершено, будет именем команды в первом слове командной строки (`nsl<TAB>` будет давать `nslookup`), и имя файла для всех остальных, если слову не предшествует "magic" (магический) символ типа `~`, `@` или `$`, когда *shell* будет пытаться заполнить имя пользователя, название машины или переменной окружения соответственно[6]. Существует также магический символ для заполнения имени команды (`!`) или имени файла (`/`).

Другие две возможности активизации подстановки это последовательности `Esc-<x>` и `Ctrl+x <x>`, где `<x>` это один из уже упомянутых магических символов. `Esc-<x>` будет пытаться подобрать уникальную подстановку. Если у него не получится, он подставит слово с наиболее возможной подстрокой из списка выбора. ***beep*** обозначает то, что выбор либо не уникальный, либо не существует соответствующего выбора.

5. Да, это будет действительно картинка PNG (однако, пакет *ImageMagick* должен быть инсталлирован в системе...).
6. Помните: *UNIX* различает большие и маленькие символы. Переменные окружения `HOME` и `home` это не одно и то же.

Последовательность `Ctrl+x <x>` отображает список возможных подстановок без попытки подстановки. Нажатие клавиши это то же самое, что последовательное нажатие `Esc-<x>` и `Ctrl+x <x>`, где магический символ зависит от контекста.

Таким образом, единственный путь увидеть все определенные переменные окружения это набрать последовательность `Ctrl+x $` в пустой строке. Другой пример: если вы хотите посмотреть *man* page (страницу руководства) для команды `nslookup`, вы можете просто набрать `man nsl`, затем `Esc-!`, и shell автоматически заполнит команду до `man nslookup`.

3.6. Запуск и Обработка Фоновых Процессов: Контроль Заданий

Вы возможно заметили, что когда вы вводите команду из *Terminal*, вам обычно приходится ждать пока команда не закончит свою работу и *shell* не отдаст вам контроль. Это означает, что вы послали команду в ***foreground*** (приоритетный режим). Однако, бывают случаи когда это нежелательно.

Представьте, например, что вы решили скопировать огромный каталог в другой рекурсивно. Также вы решили игнорировать ошибки и перенаправили канал ошибок в `/dev/null`:

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

Такая команда может выполняться несколько минут до полного завершения. У вас есть два решения: первое решение силовое - остановить (убить) команду и запустить это потом, когда у вас появится время подождать. Чтобы это сделать, нажмите `Ctrl+c`: это прервет процесс и вернет вас в командную строку. Но стоп, не делайте этого! Читайте дальше.

Допустим вы хотели бы, чтобы команда работала в то время, пока вы делаете что-либо другое. Решением является послать процесс в ***background*** (фоновый режим). Чтобы это сделать, наберите `Ctrl+z` чтобы приостановить процесс:

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

и вот вы опять в командной строке. Процесс теперь находится в ожидании, когда вы снова его перезапустите (это видно по ключевому слову `Stopped`). Что, конечно, есть то, что вы хотите сделать, но теперь в фоновом режиме (background). Наберите `bg` (*BackGround*), чтобы получить желаемый результат:

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

Процесс запустится снова как фоновая задача, что будет отображаться знаком `&` (амперсант) в конце строки. Затем вы вернетесь в приглашение командной строки и сможете продолжить работу. Процесс, запущенный как фоновая задача, или в фоновом режиме, называется ***job*** (задание).

Конечно же, вы можете сразу запускать процессы в фоновом режиме, добавляя в конце строки символ **&**. Например, вы можете запустить команду копирования каталога в фоне написав:

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

Если вам нужно, вы можете восстановить этот процесс в приоритетный режим и подождать его окончания, написав `fg` (*ForeGround*). Чтобы вернуть его снова в фоновый режим, наберите последовательность `Ctrl+z`, `bg`.

Вы можете запустить несколько заданий таким образом: тогда каждой команде будет присвоен номер задания. Команда *shell* `jobs` выводит список всех заданий, связанных с текущей *shell*. Знак `+` перед заданием указывает на последний процесс, запущенный как фоновая задача. Чтобы частично восстановить задание в приоритетном режиме, вы можете написать `fg <n>`, где `<n>` номер задания, то есть `fg 5`.

Обратите внимание, что вы можете также приостанавливать или запускать ***полноэкранные*** приложения таким же путем, такие как `less` или текстовый редактор типа *Vi*, и восстанавливать их в приоритетный режим когда нужно.

3.7. Заключительное Слово

Как вы можете видеть, *shell* является всесторонней и ее эффективное использование - вопрос практики. В этой весьма длинной главе мы рассмотрели только несколько доступных команд: **Mandrake Linux** содержит тысячи утилит и даже наиболее опытные пользователи не используют их все.

Есть утилиты на все вкусы и для любых целей: у вас есть утилиты для обработки изображений (типа `convert` упомянутой выше, но также и *GIMP* ***batch (пакетного)*** режима и все утилиты обработки ***pixmap*** (карты пикселей)), звука (Ogg Vorbis кодировщики (encoders), аудио CD проигрыватели (players)), запись CD, *e-mail* клиенты, FTP клиенты и даже веб-браузеры (типа `lynx` или `links`), не говоря уже обо всех административных средствах.

Даже если существуют графические приложения с подобными функциями, они обычно являются графическими интерфейсами, построенными на этих очень похожих утилитах. 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 страницы руководства nohup(1) *man* page).

Глава 4. Редактирование Текстов: Emacs и Vi

Как упоминалось во введении, редактирование текстов [1] это основополагающая возможность в использовании систем *UNIX*. Два редактора, которые мы собираемся рассмотреть здесь, являются слегка сложноватыми для первоначального использования, но, как только вы уясните основы, оба станут для вас мощными инструментами.

4.1. Emacs

Emacs возможно является наиболее мощным редактором из существующих. Он может делать абсолютно все, и его возможности могут расти бесконечно благодаря встроенному языку программирования на основе *lisp*. С помощью *Emacs* вы можете бродить по сети интернет, читать почту, принимать участие в группах новостей usenet, варить кофе и так далее. Однако то, что вы сможете делать в нем к концу прочтения этого раздела, будет ограничено открытием *Emacs*, редактированием одного или нескольких файлов, сохранением их и выходом из *Emacs*. Что ж, неплохо для старта.

4.1.1. Краткое Представление

Вызов *Emacs* относительно прост:

```
emacs [file] [file...]
```

Emacs откроет каждый файл, введенный как аргумент в буфер, (максимум до двух видимых буферов в одно и тоже время), или создаст буфер с именем `*scratch*`, если вы не указали файла. Если вы находитесь в *X*, то вам также будет доступно меню, но в этом разделе мы будем работать только с клавиатурой.

4.1.2. Начало

Пришло время заняться практикой. Как показано в примере, давайте откроем два файла `file1` и `file2`. Если эти файлы не существуют, они будут созданы сразу, как только вы что-нибудь напишите в них:

```
$ emacs file1 file2
```

Вы увидите окно, показанное здесь Рис. 4-1.

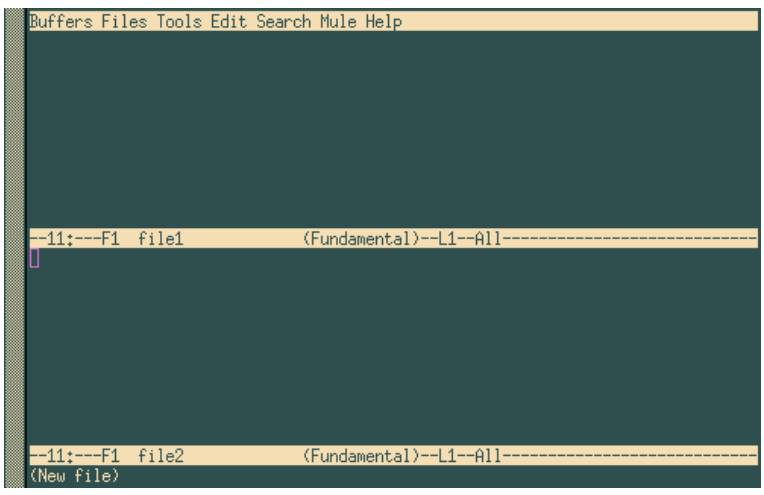

Рисунок 4-1. Emacs, редактирование двух файлов одновременно

Как вы видите, были созданы два буфера: по одному на каждый файл. Внизу экрана (где вы видите `(New file)`), у вас находится третий буфер, который называется мини-буфер. Самостоятельно в него зайти нельзя, только если вы будете приглашены программой *Emacs* во время интерактивных операций. Чтобы

1. “Редактировать текст” означает изменять содержимое файла, оперируя только буквами, цифрами и знаками пунктуации; такими файлами могут быть электронные письма, исходный код, документы или даже конфигурационные файлы.

изменить текущий буфер, наберите `Ctrl+x o`. Вы можете печатать текст как в "нормальном"редакторе, удаляя символы по клавишам `DEL` или `Backspace`.

Для перемещения курсора вы можете использовать клавиши со стрелками, а также такие комбинации клавиш: `Ctrl+a` перейти в начало строки, `Ctrl+e` перейти в конец строки, `Alt+<` перейти к началу буфера и `Alt+>` перейти в конец буфера. Есть много других комбинаций, даже для каждой клавиши со стрелкой [2].

Как только вам понадобится сохранить изменения, сделанные в файле, наберите `Ctrl+x Ctrl+s`, или, если вам нужно сохранить содержимое буфера в другой файл, наберите `Ctrl+x Ctrl+w` и *Emacs* спросит у вас имя файла куда записать содержимое буфера. Вы можете использовать ***заполнение*** чтобы сделать это.

4.1.3. Обработка Буферов

При необходимости вы можете отобразить только один буфер на экране. Вот два способа как это сделать:

- вы находитесь в буфере, который надо спрятать: наберите `Ctrl+x 0`;
- вы находитесь в буфере, который вам нужно оставить на экране, наберите `Ctrl+x 1`.

Теперь два способа как восстановить буфер на экране:

- наберите `Ctrl+x b` и введите имя буфера, который вам нужен,
- наберите `Ctrl+x Ctrl+b`, тогда будет открыт новый буфер, называемый `*Buffer List*`; вы можете передвигаться по этому буферу используя последовательность `Ctrl+x o`, затем выберите необходимый буфер и нажмите клавишу `Enter`, или можно по-другому: просто наберите имя буфера в мини-буфере. Буфер `*Buffer List*` уберется в фон как только вы сделаете выбор.

Если вы уже закончили работу с файлом и хотите избавиться от связанного буфера, наберите `Ctrl+x k`. *Emacs* здесь спросит вас какой буфер следует закрыть. По умолчанию это имя того буфера, в котором вы находитесь. Если вы хотите закрыть другой буфер, введите его имя непосредственно или нажмите `TAB`: *Emacs* в этом случае откроет еще один буфер с именем `*Completions*` (заполнение), где будет выведен список возможных вариантов. Подтвердите выбор клавишей `Enter`.

Также вы можете восстановить два видимых буфера на экране в любое время. Для этого наберите `Ctrl+x 2`. По умолчанию, вновь созданный буфер будет копией текущего буфера (в котором вы, например, сможете редактировать большой файл в разных местах "одновременно"), и вы можете перемещаться между буферами как было описано выше.

Вы можете открывать новые файлы в любое время используя `Ctrl+x Ctrl+f`. В этом случае *Emacs* спросит у вас имя файла, и здесь опять же возможно заполнение.

4.1.4. Копирование, Вырезание, Вставка, Поиск

Представьте, что мы находимся в такой ситуации Рис. 4-2.

2. *Emacs* была разработана для работы на великом множестве машин, некоторые из которых даже не имеют клавиш со стрелками. Это также справедливо и для *Vi*.

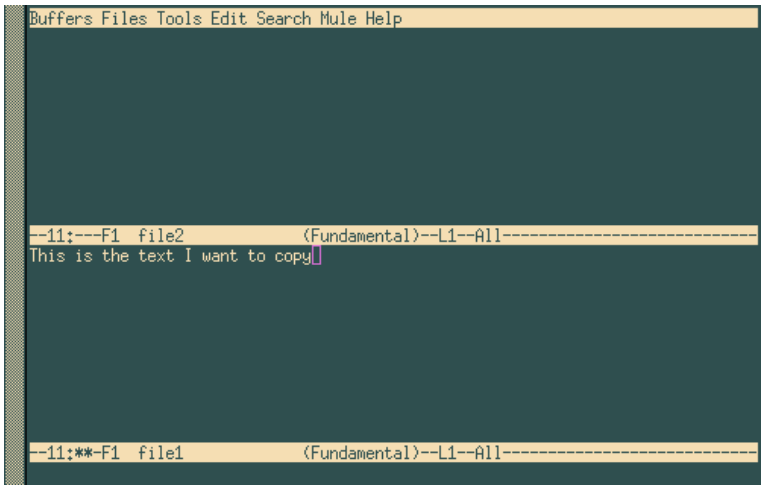

Рисунок 4-2. Emacs, перед тем как копировать текстовый блок

Сначала нужно выделить блок, который вы хотите скопировать. В *X* это можно сделать мышью и выбранная область будет подсвечена. Но мы по-прежнему находимся в текстовом режиме **:-)**. В данном случае мы хотим скопировать целое предложение. Во-первых, нужно сделать отметку в начале области. Предполагая, что курсор находится на позиции как показано на картинке выше, сначала нажмите `Ctrl+SPACE` (`Control` + пробел); после этого *Emacs* сгенерирует сообщение `Mark set` в мини-буфере. Затем переместитесь в начало строки, нажав `Ctrl+a`. Область, выделенная для копирования или вырезания, является всей областью между отметкой и текущей позицией курсора, в нашем случае - целая строка. Затем наберите `Alt+w` (для копирования) или `Ctrl+w` (для вырезания). Если вы копируете, *Emacs* сразу вернет курсор на позицию отметки так, что вы сможете увидеть выделенную область.

Теперь перейдите в буфер, в который вы хотите скопировать текст и наберите `Ctrl+y` чтобы получить то, что отображено на Рис. 4-3.

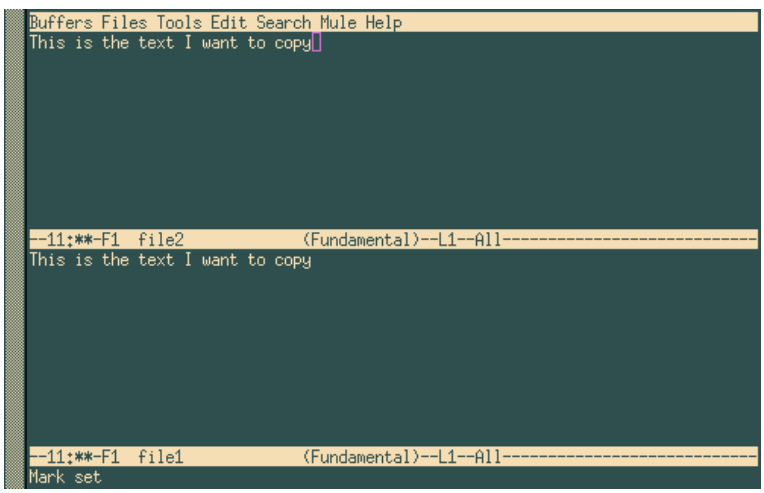

Рисунок 4-3. `Emacs`, после того, как блок скопирован

Фактически все, что вы только что сделали, это скопировали текст в *"kill ring"* программы *Emacs*; этот kill ring содержит все скопированные или вырезанные области с момента запуска *Emacs*. **Любые** свежескопированные или свежевырезанные области находятся вверху kill ring. Последовательность `Ctrl+y` только "вставляет" область в начало . Если вы хотите получить доступ к другим областям, жмите `Ctrl+y`, затем `Alt+y` пока не получите желаемый текст.

Для поиска текста зайдите в необходимый буфер и наберите `Ctrl+s`. В результате *Emacs* спросит что нужно искать. Для начала поиска подобной строки останьтесь в текущем буфере и наберите `Ctrl+s` снова. Когда *Emacs* достигнет конца буфера и больше не найдет совпадений, вы можете набрать `Ctrl+s` снова и перезапустить поиск с начала буфера. Нажатие `Enter` останавливает поиск.

Для поиска и замены наберите `Alt+%`. *Emacs* спросит какую строку искать, на что заменять и будет спрашивать подтверждение для каждого найденного совпадения.

Заключительная и очень полезная вещь: `Ctrl+x u` возвращает предыдущую операцию. Вы можете вернуться назад на любое количество операций.

4.1.5. Выход из Emacs

Горячее сочетание клавиш для этого: `Ctrl+x Ctrl+c`. Если у вас есть несохраненные данные в буферах, *Emacs* спросит хотите ли вы сохранить изменения, произведенные в буферах.

4.2. Vi: предок

Vi был первым из существующих ныне полноэкранных редакторов. *Vi* является одним из главных объектов для нападения со стороны противников *UNIX*, и в то же время одним из главных аргументов ее защитников: с одной стороны он несколько сложноват в изучении, но он становится чрезвычайно мощным инструментом как только человек набирается опыта в его использовании. Несколькими нажатиями клавиш пользователь может "воротить горы" и, кроме *Emacs*, немного есть текстовых редакторов, которые могут похвастаться тем же самым.

В действительности с **Mandrake Linux** поставляется *vim VI iMproved* (VI улучшенный), но в этом разделе мы будем называть его *Vi* .

4.2.1. Режим Вставки, Командный Режим, Режим ex...

Для начала нам нужно запустить *Vi*, что делается также, как и для *Emacs*. Итак, вернемся к двум файлам и наберем:

```
$ vi file1 file2
```

Вы увидите следующее Рис. 4-4.

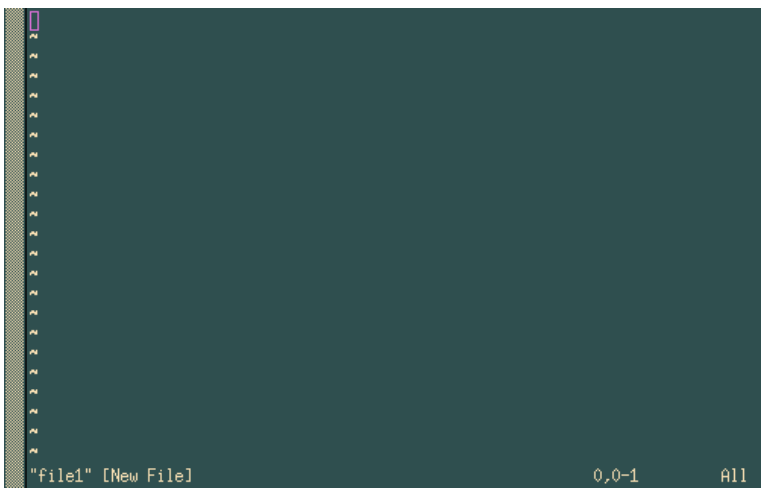

Рисунок 4-4. Начальная позиция в VIM

Теперь вы в ***командном режиме*** перед первым открытым файлом. В командном режиме вы не можете вставлять текст в файл... Чтобы это сделать, нужно перейти в ***режим вставки***, и, следовательно, ввести одну из команд, которые позволят вам это сделать:

- **a** или **i**: для вставки текста до или после курсора соответственно (**A** или **I** для вставки текста в конце или начале текущей линии соответственно);
- **o** или **O**: чтобы вставлять текст соответственно ниже и выше текущей линии.

В режиме вставки вы будете видеть строку `--INSERT--` внизу экрана (таким образом, вы будете точно знать в каком режиме вы находитесь). В этом, и только в этом режиме вы можете набирать текст. Чтобы вернуться в командный режим, наберите `Esc`.

В режиме вставки вы можете использовать `Backspace` и `DEL` для удаления текста по ходу. Для перемещения по тексту в обеих режимах, командном и вставки, используйте клавиши со стрелками. В командном режиме существуют еще и другие комбинации клавиш, которые мы рассмотрим позже.

`ex` режим доступен по нажатию клавиши `:` в командном режиме. Те же `:` появляются внизу экрана и курсор перемещается туда. Все, что вы наберете потом, завершенное нажатием `Enter`, будет рассмотрено *Vi* как команда `ex`. Если вы удалите команду и все напечатанное в `:`, вы вернетесь в командный режим и курсор переместится на свою прежнюю позицию.

Чтобы сохранить изменения в файл, наберите `:w` в командном режиме. Если вам нужно сохранить содержимое буфера в другой файл, наберите `:w <имя_файла>`.

4.2.2. Обработка Буферов

Также как и в *Emacs*, вы можете работать одновременно с несколькими буферами, отображенными на экране. Чтобы так делать, используйте команду `:split`.

Чтобы перемещаться от одного файла к другому в буфере, набирайте `:next` для перемещения к следующему файлу и `:prev` соответственно к предыдущему файлу. Вы можете также использовать `:e <имя_файла>`, что позволит вам перейти к нужному файлу если он уже открыт или открыть другой файл. Здесь снова работает заполнение.

Для смены буфера наберите `Ctrl+w j` и вы перейдете в буфер ниже или `Ctrl+w k` чтобы перейти к буферу выше. Можно также использовать клавиши со стрелками вместо **j** или **k**. Команда `:close` прячет буфер, команда `:q` его закрывает.

Будьте внимательны, потому как *Vi* весьма привередлива: если вы попытаетесь спрятать или закрыть буфер с несохраненными данными, команда не выполнится и вы получите следующее сообщение:

`No write since last change (use! to override)` (Не было записано с предыдущих изменений (используйте `!` для перезаписи))

В таком случае сделайте так, как сказано и наберите `:q!` или `:close!`.

4.2.3. Редактирование Текста и Команды Перемещения

Кроме клавиш `Backspace` и `DEL` в текстовом режиме, *Vi* имеет множество других команд для удаления, копирования, вставки и замены текста в командном режиме. Здесь мы рассмотрим некоторые из них. Все команды, показанные здесь фактически можно разделить на две части: команда для выполнения действия и символ эффекта. Действия могут быть такими:

- **c**: для замены (*Change (изменять)*); редактор удаляет указанный текст и возвращается в режим вставки после этой команды;
- **d**: для удаления (*Delete (удалить)*);
- **y**: для копирования (*Yank*), мы рассмотрим это в следующем разделе.
- **.**: повторить последнее действие.

Эффект определяет то, с какой группой символов работает команда. Вот команды эффекта, соответствующие перемещениям, которые вводятся в командном режиме:

- **h**, **j**, **k**, **l**: один символ влево, вниз, вверх, вправо[3] соответственно;
- **e**, **b**, **w**: к концу (соотв. началу) текущего слова; к началу следующего слова;
- **^**, **0**, **$**: к следующему не пустому символу текущей строки, в начало текущей строки, в конец текущей строки;
- `f<x>`: к следующему появлению символа `<x>`; например, `fe` перенесет курсор к следующему встречающемуся символу **e**;

3. Сочетание клавиш для `dl` (удалить один символ вперед) это **x**; сочетание клавиш для `dh` это **X**; `dd` удаляет текущую строку.

- `/<string>`, `?<string>`: к следующему совпадению строки или регулярного выражения (regexp) `<string>`, и то же самое назад в файле; например, `/foobar` перемещает курсор пока не появится следующее слово `foobar`;
- {, }: в начало, в конец текущего параграфа;
- **G**, **H**: в конец файла, в начало экрана.

К каждому из этих символов эффекта или командам перемещения можно добавлять номер повторения. Для **G** это ссылается на номер в файле. На этой основе вы можете создавать все варианты комбинаций.

Несколько примеров:

- `6b`: перенести назад 6 слов;
- `c8fk`: удалить весь текст до восьмого появления символа **k** затем перейти в режим вставки;
- `91G`: отправляется на строку 91;
- `d3$`: удаляет все до конца текущей строки плюс следующие две строки.

Это правда, что эти команды не очень интуитивны, но здесь дело в практике. Тем не менее, как вы можете видеть, выражение "перемещать горы несколькими клавишами" не сильно далеко ушло от истины **:-)**

4.2.4. Вырезать, Копировать, Вставить

У *`Vi`* есть команда для копирования текста, которую мы уже видели: **y**. Для вырезания текста просто используйте команду **d**. У вас есть 27 блоков памяти для хранения текста: анонимный блок и 26 именованных маленькими буквами алфавита.

Для использования анонимного блока памяти вводите команду как есть. Так, команда `y12w` копирует в анонимную память 12 слов после курсора. [4]. Используйте `d12w`, если нужно вырезать эту область.

Чтобы использовать один из 26 именованных блоков памяти, введите выражение `"<x>` перед командой, где `<x>` задает имя памяти. Таким образом, для копирования тех же 12 слов в память **k**, вы пишите `"ky12w`, и `"kd12w` для того, чтобы вырезать их.

Для помещения содержимого в анонимную память используются команды **p** или **P** (для *Вставки (Paste)*), чтобы вставить текст после или до курсора соответственно. Для вставки содержимого в именованную память, используйте `"<x>p` или `"<x>P` таким же образом (например, `"dp` вставит содержимое памяти `d` после курсора).

Давайте посмотрим на пример Рис. 4-5.

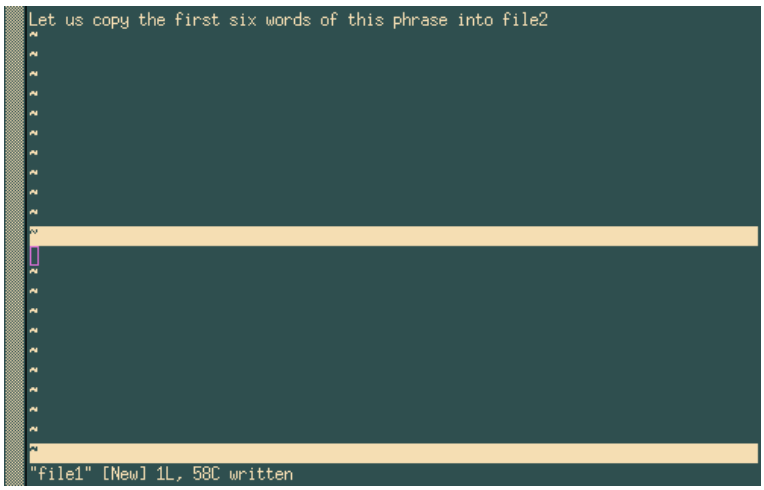

Рисунок 4-5. VIM, перед копированием текстового блока

Чтобы выполнить это действие, мы :

4. ... если курсор стоит в начале первого слова!

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

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

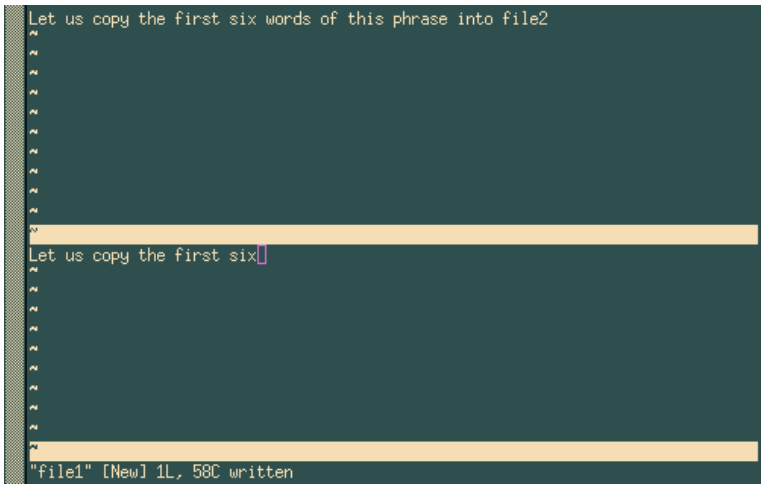

Рисунок 4-6. VIM, после того, как скопирован текстовый блок

Поиск по тексту очень прост: в командном режиме вы можете просто написать **/** перед текстом искомой строки и нажать `Enter`. Например, `/party` будет искать строку `party` от текущей позиции курсора. Нажатие **n** перемещает на следующее найденное совпадение, а если вы достигли конца файла, поиск начнется с начала файла. Чтобы искать в обратном порядке, используйте **?** вместо **/**.

4.2.5. Выход из `Vi`

Для выхода существует команда `:q` (фактически, эта команда закрывает активный буфер, как мы уже видели, и, если это единственный из открытых буферов, вы просто выйдете из *`Vi`*). Это сочетание клавиш годится только тогда, когда вы редактируете только один файл. Итак, чтобы выйти вы будете использовать:

- `:wq` сохранить изменения и выйти (более быстрое решение это `ZZ`), или
- `:q!` выйти без сохранения.

Вы должны знать, что если у вас есть несколько буферов, `:wq` запишет активный буфер и закроет его.

4.3. Последнее слово...

Конечно мы рассказали здесь гораздо больше, чем это было необходимо (в конце концов, была поставлена цель отредактировать текстовый файл), но мы стремились также показать вам некоторые возможности обоих этих текстовых редакторов. Об этих редакторах можно говорить еще много, что мы и видим на примере большого количества книг, посвященных им.

Потратьте время на усвоение всей этой информации, для одного из редакторов, или изучите только то, что считаете нужным. В конечном итоге вы в курсе, что если вы захотите идти вперед, вы сможете **:-)**

5. `y6w` дословно обозначает: “ *Выдернуть (Yank) 6 слов* ”.

Глава 5. Утилиты Командной Строки

Цель этой главы состоит в том, чтобы представить несколько инструментальных средств командной строки, которые могут оказаться полезными для каждодневного использования. Конечно, вы можете пропустить эту главу, если вы намереваетесь использовать только графическую среду, но все же лучше просмотрите - возможно вы кардинально измените свое мнение.

Эта глава не имеет четкой организации. Утилиты представлены в порядке от наиболее часто используемых к реже используемым и менее ясным. Каждая из команд иллюстрирована примером, но эта глава в основном является упражнением для более полного понимания назначения функций и способов их использования.

5.1. grep: Поиск Строк в Файлах

Да, имя команды не очень интуитивно, это аббревиатура от "General Regular Expression Parser"(("общий синтаксический анализатор регулярных выражений"). Использование этой команды просто: поиск образца, заданного как параметр в одном или более файлах. Ее синтаксис:

```
grep [опции] <Образец> [один или более файл(ов)]
```

Если упомянуто несколько файлов, то их названия будут предшествовать каждой строке соответствия, отображенной в результате. Чтобы предотвратить вывод этих имен, используйте опцию `-h`, также, для получения только имен файлов используйте опцию `-l` . Образец - это регулярное выражение, хотя в большинстве случаев это просто слово. Наиболее часто используемые опции:

- `-i`: делает поиск независимым от регистра (то есть игнорируются различия между маленькими и большими буквами);
- `-v`: инвертирует поиск: показывает строки, которые **не** содержат образца;
- `-n`: выводит номера найденных строк;
- `-w`: заставляет *`grep`* воспринимать образец как целое слово.

Вот примеры использования:

```
$ cat my_father
Hello dad
Hi daddy
So long dad

# Найти строки, содержащие "hi", не обращая внимание на регистр
$ grep -i hi my_father
Hi daddy

# Найти "dad" как целое слово и вывести номер строки перед найденным результатом
$ grep -nw dad my_father
1:Hello dad
3:So long dad

# Найти все строки, которые НЕ начинаются на "H"
$ grep -v "^H" my_father
So long dad
$
```

Если вы желаете использовать *`grep`* в канале, вы не должны указывать имя файла, в этом случае данные берутся из ***стандартного ввода***. Результат выводится в ***стандартный вывод***, поэтому вы можете смело передавать выход *`grep`* другим программам. Например:

```
$ cat /usr/share/doc/HOWTO/Parallel-Processing-HOWTO | \
  grep -n thread | less
```

5.2. find: Поиск Файлов по Некоторым Критериям

`find` является одной из старейших утилит *UNIX*. Она предназначена для рекурсивного просмотра одного или более каталогов и нахождения в этих каталогах файлов, которые соответствуют некоторому набору критериев. При всей своей полезности эта утилита имеет не очень понятный синтаксис и требует некоторых усилий для усвоения. Общий синтаксис:

```
find [опции] [каталоги] [критерии] [действия]
```

Если вы не укажете ни одного каталога, `find` будет искать в текущем каталоге. Если вы не укажете никаких критериев - это эквивалентно "true", то есть будут найдены все файлы. Опции, критерии и действия настолько многочисленные, что мы здесь упомянем только некоторые из них. Давайте начнем с опций:

- `-xdev`: не искать в каталогах, находящихся в других файловых системах;
- `-mindepth <n>`: разрешить поиск файла глубже, чем `n` уровней вниз от указанного каталога;
- `-maxdepth <n>`: искать файлы не глубже чем `n` уровней вниз от указанного каталога;
- `-follow`: следовать по символическим ссылкам на каталоги. По умолчанию `find` не ходит по символическим ссылкам;
- `-daystart`: при использовании проверок по времени (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 <type>`: поиск по типу файла; `<type>` может быть одним из: `f` (обычный файл), `d` (каталог), `l` (символическая ссылка), `s` (сокет), `b` (файл блочного типа), `c` (файл символьного типа) или `p` (именованный канал);
- `-name` <образец>: Найти файлы, чьи имена содержат <образец>. При наличии этой опции, <образец> понимается как ***шаблон подстановки*** (см. главу Разд. 3.3);
- `-iname <pattern>`: тоже самое что и `-name`, только регистронезависимо;
- `-atime <n>`, `-amin <n>`: Ищутся файлы доступ к которым был произведен `<n>` дней назад (`-atime`) или `<n>` минут назад (`-amin`). Существует также возможность указать `+<n>` или `-<n>`, в этом случае будут найдены файлы, доступ к которым был произведен соответственно больше или меньше, чем `<n>` дней/минут назад;
- `-anewer <file>`: Ищутся файлы, доступ к которым был раньше, чем к файлу `<file>`;
- `-ctime <n>`, `-cmin <n>`, `-cnewer <file>`: тоже самое, что `-atime`, `-amin` и `-anewer`, но применимо к дате последней модификации, а не последнего доступа;
- `-regex` <образец>: тоже самое, что и `-name`, но образец воспринимается как ***регулярное выражение***;
- `-iregex` <образец>: тоже самое, что и `-regex`, но не зависит от регистра.

Существует масса других атомарных проверок. Подробнее узнать о них можно, почитав страницы руководства `find`(1). Проверки можно комбинировать одним из следующих способов:

- <c1> -a <c2>:(AND) true, если каждое <c1> и <c2> равны true; `-a` является неявным, поэтому вы можете его опускать и писать <c1> <c2> <c3> ... ;
- <c1> -o <c2>:(OR) true, если любой из <c1> или <c2> или оба равны true. Примечание: `-o` имеет меньший ***приоритет***, чем `-a`; для того чтобы выполнялся один из критериев <c1> или <c2> и обязательно критерий <c3>, вы можете использовать такую запись (<c1> -o <c2>) -a <c3>. Вы должны ***защитить (escape)*** (деактивировать) круглые скобки, иначе они будут интерпретироваться *shell*!
- `-not <c1>`: инвертирует проверку <c1>, поэтому `-not <c1>` будет true в случае, когда <c1> равно false.

И наконец, вы можете определить действие для каждого найденного файла. Вот наиболее используемые:

- `-print`: Выводит имена файлов на стандартный вывод. Это действие по умолчанию;
- `-ls`: Выводит найденные файлы в стандартный вывод эквивалентно команде `ls -ilds`;

- `-exec <command>`: выполняет команду `<command>` для найденных файлов. Командная строка `<command>` заканчивается символом `;`, который должен быть защищен (деактивирован) для того, чтобы *`shell`* его не интерпретировала. Позиция в файле отмечается при помощи `{}`. В примерах по использованию (ниже) это хорошо проиллюстрировано;
- `-ok <command>`: тоже самое что и `-exec`, но спрашивает подтверждения перед исполнением каждой команды.

Вы все еще здесь? Замечательно, теперь немного попрактикуемся, потому что это лучший способ разобраться с этим монстром. Допустим, вы хотите найти все каталоги, находящиеся в каталоге `/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`, который в данный момент является текущим каталогом. Вы хотите найти все файлы, содержание которых не изменялось месяц. Поскольку странички на сервере делали разные автора, некоторые файлы имеют расширение `html`, а некоторые `htm`. Вы хотите поместить ссылки на такие файлы в каталог `/var/www/obsolete`. Для этого нужно сделать следующее [1]:

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

Хорошо, это несколько сложновато и требует небольшого объяснения. Вот это есть критерий:

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

который делает то, что мы хотим: он находит все файлы, имена которых заканчиваются на `.htm` или `.html` " `\( -name "*.htm" -o -name "*.html" \)` ", **и** (`-a`) которые не были изменены на протяжении последних 30 дней, то есть приблизительно месяц (`-ctime -30`). Обратите внимание на круглые скобки: они необходимы здесь, потому что `-a` имеет более высокий приоритет. Если бы их не было, то были бы найдены все файлы, заканчивающиеся на `.htm`, а также все файлы, которые не изменялись в течении месяца и заканчиваются на `.html`, а это не то, что нам нужно. Также обратите внимание, что круглые скобки защищены от *`shell`*: если бы мы написали `( .. )` вместо `\( .. \)`, *`shell`* интерпретировала бы их и попробовала выполнить `-name "*.htm" -o -name "*.html"` в другой sub-*`shell`*... Эту проблему можно было бы решить и по-другому - взять круглые скобки в двойные или одинарные кавычки, но обратная наклонная черта предпочтительней, так как в этом случае нужно изолировать только один символ.

И наконец команда, которая будет выполнена для каждого из найденных файлов:

```
-exec ln {} /home/httpd/obsolete \;
```

Здесь также нужно изолировать `;`, иначе *`shell`* отинтерпретирует это как разделитель команд. Если вы этого не сделаете, то `find` начнет жаловаться на то, что у `-exec` пропущен аргумент.

Последний пример: у вас есть огромный каталог (`/shared/images`), содержащий все виды изображений. Вы постоянно пользуетесь командой *`touch`* для того, чтобы изменять время файла с именем `stamp` в этом каталоге, чтобы всегда иметь временную метку. Вы желаете найти все картинки ***JPEG***, которые новее, чем файл `stamp`, ну и, поскольку вы как всегда получаете файлы из разных источников, то файлы имеют расширения `jpg`, `jpeg`, `JPG` или `JPEG`. Кроме того, вы хотите избежать поиска в каталоге `old`. А еще вы хотите, чтобы этот список был отправлен к вам по почте. Ваш имя пользователя - `peter`:

```
find /shared/images -cnewer     \
     /shared/images/stamp       \
     -a -iregex ".*\.jpe?g"     \
     -a -not -regex ".*/old/.*" \
```

1. Примечание : для корректной работы этого примера, каталоги `/var/www` и `/var/www/obsolete` должны находится в одной файловой системе!

```
      | mail peter -s "New images"
```

Вот так вот! Конечно, эта команда не очень удобна, если вы должны набирать её каждый раз, и вы хотели бы, чтобы это выполнялось регулярно... Это можно сделать так:

5.3. crontab: проверка и изменение вашего файла crontab

`crontab` это команда, которая позволяет вам регулярно выполнять команды через заданные промежутки времени, при этом вам нет необходимости входить в систему, а отчет о выполнении команды будет высылаться к вам на email. Вы можете задавать интервалы в минутах, часах, днях и даже месяцах. В зависимости от заданных опций, `crontab` может делать разные действия:

- `-l`: Показать ваш текущий файл `crontab`.
- `-e`: Редактировать ваш файл `crontab`.
- `-r`: Удалить ваш файл `crontab`.
- `-u <user>`: Применить одну из вышеупомянутых опций для пользователя `<user>`. Это может делать только `root` .

Давайте начнем редактировать `crontab`. Если вы наберете командочку `crontab -e`, вы очутитесь в своем любимом текстовом редакторе. В случае, если вы не изменяли переменных окружения `EDITOR` или `VISUAL`, то это будет *`Vi`*. Файл `crontab` состоит из строк, каждая из которых имеет шесть полей. Первые пять полей определяют временной интервал в минутах, часах, днях, месяцах и днях недели. Шестое поле - это команда для выполнения. Строки, начинающиеся с `#` - это комментарии и они будут проигнорированы `crond` (это программа, которая отвечает за выполнение заданий, содержащихся в файле `crontab`). Приведем пример файла `crontab`:

Для того, чтобы привести этот пример удобочитаемым шрифтом, мы были вынуждены сделать разрывы длинных строк. Поэтому, если в примере строка заканчивается символом \, это значит, что следующая строка является продолжением текущей. Это соглашение действительно для файлов `Makefile`, в командах и скриптах *`shell`*, а также в любых других местах, где это следует из контекста.

```
#   Если вы не желаете, чтобы вам отсылалась почта,
# просто раскомментируйте следующую строку
# MAILTO=""
#
#   Каждые 2 дня в 14:00 выполнять
#   пример, приведенный выше. После этого
#   сделать временную метку (файл "stamp")
#   с помощью "retouch"
#   Символ "%" рассматривается как новая строка
#   что дает вам возможность помещать 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
#
# На каждое рождество проиграть мелодию :)
0 0 25 12 * ogg123 $HOME/sounds/merryxmas.ogg
#
# Каждый вторник печатать список покупок
0 17 * * 2 lpr $HOME/shopping-list.txt
```

Существует еще несколько способов задания интервалов, отличных от приведенного примера. Например, вы можете задавать ***дискретные значения***, разделяя их запятыми (`1,14,23`) или диапазон (`1-15`), или

комбинировать оба способа (`1-10,12-20`), или даже с произвольным шагом (`1-12,20-27/2`). Теперь ваша задача найти полезные команды и поместить их сюда!

5.4. at: единоразовое задание команды

Возможно, вы хотели бы выполнить какую-то команду в заданный день в заданное время, но всего один раз (не периодически). Например, вы хотите напомнить себе о встрече, которая состоится в 18:00. Вы работаете в *X*, и желаете получить напоминание в 17:30 о том, что вам пора уходить. В этом случае программа `at` - как раз то, что вам нужно:

```
$ at 17:30
  # в этом месте "at" ждет ввода команды для выполнения
at> gmessage "Пора бежать! Встреча в 18:00"
  # для выхода нажмите CTRL-D
at> <EOT>
$
```

Задавать время можно разными способами:

- `now +<interval>`: Это значит сейчас + интервал (интервал необязателен и если он не определен, то это значит - сейчас). Синтаксис для интервала: `<n>`(minutes | hours | days | weeks | months). Где : minutes-минут, hours-часов, days-дней, weeks-недель, months-месяцев. Например, вы можете задать `now + 1 hour` (один час, начиная от сего момента), `now + 3 days` (трое суток от текущего момента), и так далее.
- `<time> <day>`: Полное указание даты. Обязательный параметр `<time>`(время) и он у программы `at` является очень либеральным. То есть вы можете указывать время самыми загадочными способами. Например, можно написать: `0100`, `04:20`, `2am`, `0530pm`, `1800`, или одно из трех предопределенных значений: `noon` (в полдень 12:00), `teatime` (4pm или 16:00) или `midnight` (полночь 00:00). Параметр `<day>`(день) является необязательным. Его тоже можно указывать различными способами: `12/20/2002` (20-ое декабря 2002 года) или так, как принято в Европе `20.12.2002` Вы можете не указывать год, но в этом случае работает только европейская форма записи даты: `20.12`. Можно указывать месяц буквами: `Dec 20` или `20 Dec`, оба варианта верны.

К программе `at` также применимы следующие опции:

- `-l`: Печатает список текущих заданий; Первое поле - номер задания. Это эквивалентно вызову команды `atq`.
- `-d <n>`: Убирает из очереди задачу с номером `<n>`. Вы можете получить номера задач в очереди, используя команду `atq`. Это эквивалентно использованию команды `atrm <n>`.

Как обычно, для получения более подробной информации, смотрите страницы справочного руководства at(1) `manpage`.

5.5. tar: Tape ARchiver - Архиватор для накопителей на магнитной ленте

Несмотря на то, что мы уже использовали `tar` в главе Гл. 12, мы до сих пор не объяснили как она работает. Именно для этого этот раздел здесь и помещен. Как и `find`, `tar` является одной из старейших стандартных утилит *UNIX*. И, точно также как `tar`, имеет несколько специфичный синтаксис:

```
tar [options] [files...]
```

Теперь приведем список опций. Обратите внимание, что все приведенные опции имеют и длинный вариант написания, но для ознакомления с ними вам прийдется обратиться к страницам руководства, так как здесь они не будут приведены. И конечно же, мы не станем приводить здесь всех опций **:-)**

стартовая черточка (`-`) перед короткой опцией `tar` больше не используется, только после длинной опции.

- `c`: эта опция используется для создания нового архива;
- `x`: эта опция используется для извлечения файлов из существующего архива;
- `t`: предоставляет список файлов в существующем архиве;
- `v`: эта опция просто выводит список файлов, которые добавляются или извлекаются из архива, или, в сочетании с опцией `t` (см. выше), выводит список в длинном формате;
- `f <file>`: создает архив с именем `<file>`, извлекает из архива с именем `<file>` или получает список файлов в архиве с именем `<file>`. Если этот параметр отсутствует, то файлом по умолчанию будет `/dev/rmt0`, который является специальным файлом, связанным с устройством ***streamer***. Если вместо имени файла подставить `-` (минус), ввод или вывод (в зависимости от того, создается или разворачивается архив) будет ассоциирован со стандартным вводом или стандартным выводом;
- `z`: сообщает `tar` что архив, который должен быть создан, нужно сжать с помощью `gzip`, или что архив, который нужно развернуть, сжат `gzip`;
- `j`: тоже самое что и `z`, но программа, используемая для компрессии или декомпрессии это `bzip2`;
- `p`: при извлечении файлов из архива сохраняются все атрибуты, включая владельца, время последнего доступа и так далее. Очень полезно при снятии дампа файловой системы;
- `r`: добавляет в конец существующего архива файлы, список которых получен из командной строки. Обратите внимание на то, что архив, к которому вы хотите добавить файлы, **не** должен быть компрессированным!
- `A`: добавляет архив, имя которого получено из командной строки, к другому архиву (имя которого берется у опции `f`). По аналогии с опцией `r`, для того чтобы это работало - архивы не должны быть компрессированными;

Существует еще много, много, много других опций, поэтому, для получения полного списка, мы вам советуем обратится к справочному руководству tar(1) manual page. Давайте попрактикуемся и рассмотрим для примера опцию `d`. Допустим, мы хотим создать архив всех картинок, находящихся в каталоге `/shared/images`, скомпрессировать их с помощью `bzip2`, затем назвать его `images.tar.bz2`, и разместить в своем домашнем каталоге. Для этого наберите следующее:

```
 #
 # Примечание: вы должны находиться в том каталоге,
 # файлы из которого вы желаете заархивировать!
 #
$ cd /shared
$ tar cjf ~/images.tar.bz2 images/
```

Как вы наверное заметили, в данном примере мы использовали три опции: `c` говорит `tar` что мы желаем создать архив, `j` сообщает о том, что желаем его скомпрессировать при помощи `bzip2`, и `f ~/images.tar.bz2` добавляет, что архив должен быть создан в вашем домашнем каталоге и носить имя `images.tar.bz2`. Наверняка мы захотим проверить, правильно ли создался архив. Для этого, например, можно получить список файлов, входящих в него:

```
 #
 # Вернуться в ваш домашний каталог
 #
$ cd
$ tar tjvf images.tar.bz2
```

Этим мы сказали `tar`-у создать список файлов (`t`) архива `images.tar.bz2` (`f images.tar.bz2`), и предупредили его о том, что этот архив был скомпрессирован при помощи `bzip2` (`j`), а также о том, что нам интересно получить список в длинном формате (`v`). Теперь, предположим, что вы потеряли ваш каталог с картинками... Но, к счастью, у вас осталась копия в архиве, и вполне естественно, мы теперь хотим развернуть это и вернуть файлы на их место в `/shared`. Чтобы в дальнейшем обеспечить корректную работу команды `find`, нужно сделать так, чтобы были сохранены все атрибуты файлов:

```
 #
 # перейдем в каталог, в который нам необходимо развернуть архив
 #
```

```
$ cd /shared
$ tar jxpf ~/images.tar.bz2
```

Вот и все!

Теперь, к примеру, вы желаете извлечь из этого архива только подкаталог с автомобилями `images/cars` и больше ничего. Для этого вы должны набрать:

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

Если вы испытываете опасения относительно использования этого, то не бойтесь. При попытке сделать резервную копию специальных файлов, программа `tar` поместит в архив только названия таких файлов, и не будет помещать содержимого. Так произойдет в случае, если вы захотите поместить в архив файл `/dev/mem`. Также `tar` ведет себя корректно при работе с ссылками, так что не волнуйтесь относительно этого. Для работы с символическими ссылками обратите внимание на опцию `h` в `manpage`.

5.6. bzip2 и gzip: Программы для компрессирования (сжатия) данных

Как вы наверно заметили, мы уже говорили об этих двух программах когда рассматривали `tar`. В отличии от `winzip` для *`Windows`*, архивирование и компрессия осуществляется двумя различными утилитами –: `tar` для архивации, и двумя программами для сжатия `bzip2` и `gzip`. Рассмотрением этих двух программ мы сейчас и займемся. Кстати, возможно использование и других утилит для сжатия данных, таких как `zip`, `arj`, `rar` и прочих. Они также существуют для *`GNU/Linux`*, но весьма редко используются.

Программа `bzip2` была написана для замены программы `gzip`. Она действительно показывает лучшие характеристики по степени сжатия данных, но, с другой стороны она потребляет значительно больше памяти при работе. По этой причине программа `gzip` все еще более широко используется, чем `bzip2`.

Обе эти команды имеют подобный синтаксис:

```
gzip [options] [file(s)]
```

В случае, когда не указывается filename, обе программы `gzip` и `bzip2` будут ожидать данных со стандартного потока ввода или посылать результаты в стандартный поток вывода. Поэтому вы можете использовать обе эти программы в каналах. Они также имеют общий набор опций:

- `-1`, ..., `-9`: устанавливает степень сжатия. Большее число - большая степень сжатия, но, соответственно и медленнее.
- `-d`: разжать файл(ы). Это эквивалентно использованию программ `gunzip` или `bunzip2`;
- `-c`: выбросить результаты компрессии/декомпрессии файлов, имена которых получены как параметры, в стандартный вывод.

По умолчанию, обе программы `gzip` и `bzip2` удаляют файлы, которые были компрессированы/декомпрессированы, если вы не используете опцию `-c`. В программе `bzip2` вы можете избежать этого, используя опцию `-k`, но программа `gzip` не имеет подобной опции!

Теперь приведем несколько примеров. Допустим, мы желаем скомпрессировать все файлы в текущем каталоге, названия которых заканчиваются на `.txt`, используя программу `bzip2`. Вы можете сделать это так:

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

Допустим, вы хотите дать попользоваться кому-то своими картинками, но у этого человека есть только `gzip`, а `bzip2` напрочь отсутствует. Вам не прийдется декомпрессировать архив, а потом компрессировать

его снова. Вы можете декомпрессировать его в стандартный вывод, использовать канал, скомпрессировать со стандартного ввода и перенаправить вывод в новый архив:

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

Вот все и получилось. Кстати, вы можете вызвать `bzcat` вместо `bzip2 -dc`. Это работает и для программы `gzip`, но её имя `zcat`, но не **gzcat**. Также вы располагаете `bzless` (и по аналогии `zless`), которая предназначена для просмотра компрессированных файлов без предварительной декомпрессии. В качестве упражнения попробуйте найти команды, которые нужно ввести для просмотра сжатого файла без использования утилит `bzless` или `zless`.

5.7. Значительно больше...

Существует еще так много команд, что книга, которая бы всесторонне их описывала, имела бы размер огромной энциклопедии. Эта глава не охватывает даже десятой части рассматриваемого предмета. Тем не менее, воспользовавшись знаниями, полученными в этой главе, вы уже можете сделать многое. При желании вы можете прочитать следующие `manual` pages: sort(1), sed(1) and zip(1) (да, это то, о чем вы подумали: вы можете создавать и использовать `.zip` под *GNU/Linux*), convert(1), и так далее. Лучшим способом получить знания по этим утилитам является большое количество практики и экспериментов с ними. Вы найдете множество способов использовать полученные знания, порой даже самых неожиданных. Получайте удовольствие от процесса!

Глава 6. Контроль Процессов

6.1. Подробнее о Процессах

Если вообще возможно мониторить процессы и “просить” их прерваться, остановиться, продолжить и т.д., то эта глава как раз и охватывает эту тему. Чтобы понимать операции, которые мы собираемся производить здесь, полезно будет знать немного больше о процессах.

6.1.1. Дерево Процессов

Как и файлы, все процессы, которые запускаются в системе *GNU/Linux*, организованы в виде дерева. Корень этого дерева команда `init`. Каждый процесс имеет номер (свой PID, *Process ID*), а также номер своего родительского процесса (PPID, *Parent Process ID*). PID `init` равен 1, и такой же его PPID: `init` сам себе отец.

6.1.2. Сигналы

Каждый процесс в *UNIX* может реагировать на сигналы, посланные ему. Существует 64 различных сигнала, которые идентифицируются по своим номерам (начиная с 1) или по их символическим именам (`SIGx`, где `x` имя сигнала). 32 “старших” сигнала (от 33 до 64) являются сигналами реального времени и в этой главе мы их рассматривать не будем. Для каждого из этих сигналов у процесса может быть свой собственный behavior, исключая два сигнала: сигнал номер 9 (`KILL`), и сигнал номер 19 (`STOP`).

Сигнал `9` прерывает процесс безоговорочно, не давая времени на нормальное прекращение. Этот сигнал вы посылаете процессам, которые залипли или создают проблемы другим. Полный список доступных сигналов можно получить по вызову команды `kill -l`.

6.2. Информация о Процессах: ps и pstree

Эти две команды выводят список процессов, запущенных в данный момент в системе, согласно установленным вами критериям.

6.2.1. ps

Запустив эту команду без аргументов, вы увидите только процессы, запущенные вами и только с данного терминала, который вы сейчас используете:

```
$ 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` имеет массу опций, наиболее распространенные из них это:has a handful of

- `a`: также показывает процессы, запущенные другими пользователями;
- `x`: также показывает процессы, не имеющие контролирующего терминала или запущенные с другого терминала;
- `u`: выводит имя пользователя, который запустил процесс, и время запуска для каждого из процессов.

Существует еще множество опций. За полной информацией рекомендуем обратится к соответстыущим страничкам руководства. В данном случае это : ps(1).

Вывод команды разделен на несколько полей: одно из них, которое будет вас интересовать чаще, это поле `PID`, которое содержит информацию об идентификаторе процесса. Поле `CMD` содержит имя выполняемой. Чаще всего команда `ps` используется так:

```
$ ps ax | less
```

Этим вы получите список всех работающих в настоящий момент процессов, что даст вам возможность определить, какие процессы создают проблемы и прервать их.

6.2.2. pstree

Команда `pstree` выводит процессы в форме дерева. Одно из преимуществ этого это то, что вы можете сразу видеть, какой процесс является родителем какого: когда вам нужно будет убить целый ряд процессов, в случае когда интересующие вас процессы являютя дочерними для одного родительского, то вы можете просто убить родителя. Вы можете использовать опцию `-p`, которая отображает PID каждого процесса и опцию `-u`, которая отображает имя пользователя, который запустил этот процесс. Так как дерево в общем случае весьма длинное, вы можете вызвать `pstree` так:

```
$ pstree -up | less
```

Таким образом вы получите обзор всего дерева процессов.

6.3. Посылка Сигналов Процессам: kill, killall и top

6.3.1. kill, killall

Эти две команды используются для посылки процессам сигналов. Команда `kill` требует в качестве аргумента номер процесса, а `killall` требует имя процесса.

Обе эти команды опционально могут принимать номер сигнала, который должен быть послан процессу в качестве аргумента. По умолчанию, они обе посылают сигнал номер 15 (`TERM`) для соответствующего процесса (или нескольких процессов). Например, если вам надо убить процесс с номером PID 785, вы вводите команду:

```
$ kill 785
```

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

```
$ kill -19 785
```

Допустим, вам нужно убить процесс, для которого вы знаете имя команды. Чтобы не искать номер процесса командой `ps`, вы можете убить процесс по его имени:

```
$ killall -9 netscape
```

Чтобы не случилось, вы будете убивать только свои собственные процессы (если конечно вы не `root`), так что не беспокойтесь о процессах “соседей” с такими же именами, с ними ничего не произойдет.

6.3.2. Объединение ps и kill: top

`top` это программа, которая соединяет в себе функции `ps` и `kill`. Она часто используется в качестве монитора процессов в реальном режиме времени, предоставляя также информацию о загрузке CPU и памяти, времени выполнения и т.д., как показано здесь Рис. 6-1.

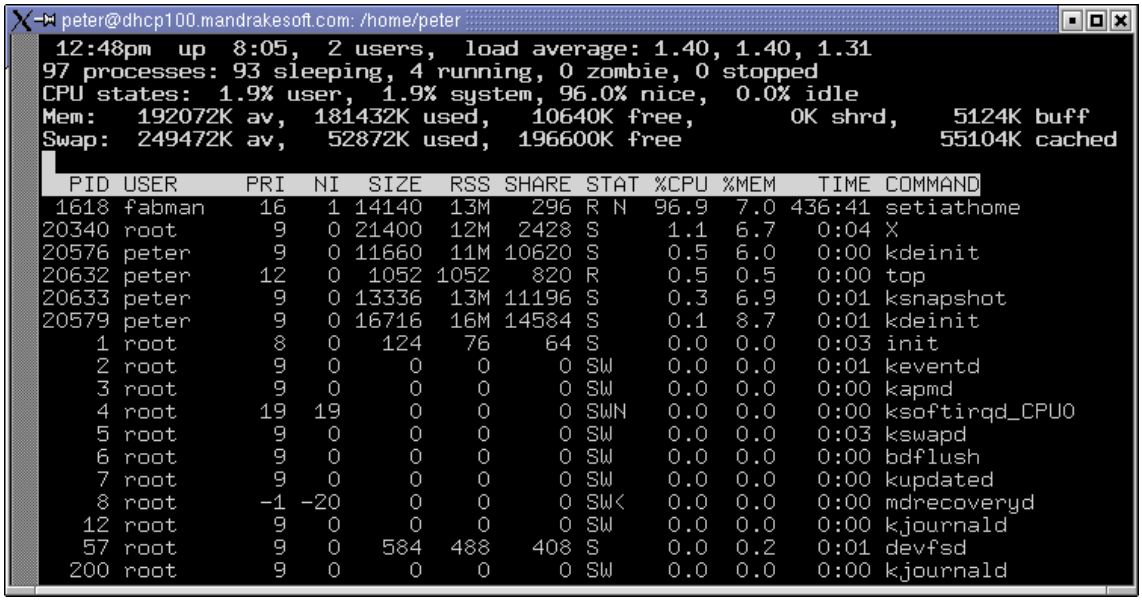

Рисунок 6-1. Мониторинг процессов с помощью top

Программа полностью контролируется с клавиатуры. Справку можно получить по нажатию **h**. Наиболее часто используемые команды это:

- **k**: эта команда используется для посылки сигнала процессу `top` спросит вас PID процесса и номер сигнала, который нужно послать (`TERM` — или `15` — по умолчанию);
- **M**: эта команда используется для сортировки процессов по количеству используемой ими памяти (поле `%MEM`);
- **P**: эта команда используется для сортировки процессов по занятому ими времени CPU (поле `%CPU`;, метод сортировки по умолчанию);
- **u**: эта команда используется для отображения процессов данного пользователя. `top` спросит какого именно. Вводите **имя** пользователя, но не его UID. Если вы не введете имени, будут показаны все процессы;
- **i**: по умолчанию отображаются все процессы, даже спящие; эта команда предоставит информацию только о работающих в данный момент процессах (процессах, у которых поле `STAT` имеет значение `R`, *Running*). Повторное использование этой команды, снова вернет вас в предыдущее состояние отображения.

Глава 7. Организация Дерева Файлов

В настоящее время, *UNIX* система большая, очень большая. Это особенно касается *GNU/Linux*: количество программного обеспечения сделало бы ее неуправляемой, если бы не существовало определенных правил расположения файлов в дереве.

Общепринятым в этом отношении является стандарт FHS (*Filesystem Hierarchy Standard* (стандарт иерархии файловой системы)) версии 2.2 на момент написания данного руководства. Документ, который описывает стандарт, доступен в Интернете в различных форматах по адресу The Pathname web site (`http://www.pathname.com/fhs/`). Этот раздел дает только краткую информацию, которой будет достаточно, чтобы знать в каком каталоге искать (или разместить) нужный файл.

7.1. Разделяемые/Неразделяемые, Статические/Переменные Данные

Данные в *UNIX* системе можно классифицировать согласно двум критериям. Они обозначают следующее: разделяемые данные могут быть одинаковыми для нескольких компьютеров в сети, в то время как неразделяемые не могут. Статические данные не могут изменяться при обычном использовании, переменные же данные могут. Во время исследования древовидной структуры мы будем рассматривать различные каталоги с позиции этих категорий.

Обратите внимание, что данные классификации только рекомендованы. Им не обязательно следовать, но принятие этих рекомендаций очень поможет вам управлять вашей системой. Также, имейте в виду, что различие статический/переменный применимо только к системному использованию. Если вы инсталлируете программу, вам очевидно прийдется изменить "нормальные" статические каталоги, то есть: `/usr`

7.2. Корневой Каталог: /

Каталог `root` содержит в себе всю иерархию системы. Его нельзя классифицировать, так как его подкаталоги могут быть как статическими или разделяемыми, так и нет. Вот список главных каталогов и подкаталогов с их классификацией:

- `/bin`: Основные бинарные файлы. Этот каталог содержит базовые команды, который могут использоваться всеми пользователями и являются необходимыми для работы системы: `ls`, `cp`, *login*, и т.д.. Статический, неразделяемый;
- `/boot`: содержит файлы, необходимые для загрузчика *GNU/Linux* (*grub* или *LILO* для **Intel**). В нем может находится kernel: если его здесь нет, он должен быть в каталоге `root`. Статический, неразделяемый;
- `/dev`: файлы системных устройств (`dev` для *DEVices*). Статический, неразделяемый;
- `/etc`: в этом каталоге находятся все конфигурационные файлы данного компьютера. Статический, неразделяемый;
- `/home`: содержит все персональные каталоги пользователей системы. Этот каталог может быть или не быть разделяемым (в некоторых больших сетях он может быть открыт на доступ по NFS). Переменный, разделяемый;
- `/lib`: этот каталог содержит основные библиотеки системы. В нем также хранятся модули ядра в `/lib/modules`. Все библиотеки, необходимые для работы бинарников из `/bin` и `/sbin` каталогов должны размещаться здесь, вместе с линковщиком `ld.so`. Статический, неразделяемый;
- `/mnt`: каталог, содержащий точки монтирования для временных файловых систем. Переменный, неразделяемый.
- `/opt`: хранит не требуемые для системных операций пакеты. Рекомендуется размещать здесь файлы (бинарники, библиотеки, страницы руководств и т.д.) для пакетов типа `/opt/8<O_?0:5B0` и их специфических конфигурационных файлов из `/etc/opt`;
- `/root`: домашний каталог для `root`. Переменный, неразделяемый;
- `/usr`: см. следующий раздел. Статический, разделяемый;
- `/sbin`: содержит системные бинарники для запуска системы, доступные только для `root`. Обычный пользователь тоже может запускать их но не будет иметь к ним быстрого доступа. Статический, неразделяемый;

- `/tmp`: каталог предназначен для того, чтобы содержать временные файлы, создаваемые программами. Переменный, неразделяемый;
- `/var`: место расположения данных, которые могут изменяться программами в реальном режиме времени (то есть: почтовыми серверами, программами наблюдения, серверами печати и т.д.) Весь каталог `/var` является переменным, но его некоторые подкаталоги могут быть разделяемыми или неразделяемыми.

7.3. /usr: Большой

Каталог `/usr` является главным каталогом для хранения приложений. Все бинарные файлы в этом каталоге не должны быть необходимыми для загрузки или обслуживания системы потому что иерархия `/usr` часто располагается на отдельной файловой системе. Учитывая его большой размер, `/usr` имеет свою собственную иерархию подкаталогов. Мы упомянем только несколько:

- `/usr/X11R6`: полная иерархия *X Window System* . Все бинарники, необходимые для работы *X* (включая *X* сервера) и все необходимые библиотеки должны находиться здесь. Каталог `/usr/X11R6/lib/X11` содержит все аспекты конфигурации *X*, которые не отличаются для разных компьютеров. Специфические конфигурации для каждого компьютера должны размещаться в `/etc/X11`;
- `/usr/bin`: этот каталог содержит большинство системных бинарников. **Любая** бинарная программа, которая не является необходимой для обслуживания системы и не для системного администрирования, должна размещаться здесь. Исключением являются программы, которые вы инсталлировали самостоятельно. Они должны размещаться в `/usr/local`;
- `/usr/lib`: содержит все необходимые библиотеки для запуска программ из каталогов `/usr/bin` и `/usr/sbin`. Есть также символическая ссылка `/usr/lib/X11`, указывающая на каталог, содержащий *X Window System* библиотеки, `/usr/X11R6/lib` (если *X* конечно инсталлированы);
- `/usr/local`: это место, куда вы должны инсталлировать собственные приложения. Программа инсталляции будет сама создавать необходимую иерархию: `lib/`, `bin/`, и т.д.;
- `/usr/share`: этот каталог содержит все аппаратно-независимые данные, необходимые для приложений из `/usr`. Среди других вещей вы найдете здесь информацию о зоне и расположении. (`zoneinfo` и `locale`).

Необходимо также упомянуть каталоги `/usr/share/doc` и `/usr/share/man`, которые соответственно содержат документацию приложений и страницы руководства системы.

7.4. /var: Изменяемые при Использовании Данные

Каталог `/var`содержит все оперативные данные программ, запущенных в системе. В отличие от рабочих данных в каталоге `/tmp`, эти данные должны сохраниться неповрежденными в случае перезагрузки. Есть много подкаталогов, и некоторые очень полезны:

- `/var/log`: содержит файлы системных журналов (log);
- `/var/spool`: хранит рабочие файлы системных демонов. Например, `/var/spool/lpd` содержит рабочие файлы сервера печати, а `/var/spool/mail` хранит рабочие файлы почтового сервера (то есть всю почту приходящую и уходящую из вашей системы).
- `/var/run`: используется для слежения за всеми процессами, используемыми системой и позволяет вам производить над ними действия в случае изменений в системе ***runlevel*** (см. раздел Гл. 11).

7.5. /etc: Конфигурационные Файлы

`/etc` один из основных каталогов систем *UNIX*. Он содержит все базовые конфигурационные файлы системы. **Никогда** не удаляйте его чтобы очистить дисковое пространство! Более того, если вы желаете разбить вашу древовидную структуру на несколько разделов, помните, что `/etc` не может быть размещен на отдельном разделе: он необходим для инициализации системы.

Вот некоторые важные файлы:

- `passwd` и `shadow`: это два текстовых файла, которые содержат всех пользователей системы и их зашифрованные пароли. `shadow` нужен если вы используете теневые пароли, что является опцией инсталляции по умолчанию;
- `inittab`: конфигурационный файл для команды `init`, который играет основную роль в загрузке системы, как мы это увидим позже.
- `services`: файл содержит список существующих сервисов сети;
- `profile`: это конфигурационный файл *`shell`*, хотя некоторые *`shells`* используют другие файлы. Например, *`bash`* использует файл `bashrc`;
- `crontab`: конфигурационный файл команды `cron`, программы, ответственной за периодическое выполнение программ.

Кроме того, некоторые подкаталоги существуют для программ, которые требуют большое количество конфигурационных файлов. Это относится к *`X Window System`*, например, который хранит все свои файлы в каталоге `/etc/X11`.

Глава 8. Файловая Система и Точки Монтирования

Наилучший путь понять “как это работает” это рассматривать на практическом примере, что мы и собираемся здесь сделать. Предположим вы только что приобрели новый жесткий диск и на нем нет никаких разделов. Ваш раздел **Mandrake Linux** полон до отказа и вместо того, чтобы начинать все сначала, вы решаете перенести целый раздел древовидной структуры на новый диск. Так как новый диск очень большой, вы решаете перенести самый большой каталог на него: `/usr`. Но сначала немного теории.

8.1. Принципы

Как мы уже отмечали в *Руководство по Установке*, каждый жесткий диск разделен на несколько разделов, каждый из которых содержит файловую систему. В то время как *Windows* назначает букву каждой из своих файловых систем (конечно только для тех, которые распознает), *GNU/Linux* имеет уникальную древовидную структуру файлов, и каждая файловая система ***примонтирована*** в одно местоположение в древовидной структуре.

Также как *Windows* необходим “C: drive”, *GNU/Linux* должна иметь возможность где-нибудь примонтировать корень (root) своей файловой системы (/), фактически в разделе, который содержит ***корневую файловую систему*** . Как только корень примонтирован, вы можете монтировать другие файловые системы древовидной структуры в различные ***точки монтирования***. Любой каталог ниже корневой структуры может работать как точка монтирования. Обратите внимание, что вы можете монтировать одну и ту же файловую систему несколько раз.

Это дает большую гибкость конфигурации. В случае веб-сервера, например, обычное дело выделить целый раздел под каталог, в котором размещаются данные веб-сервера. В общем случае каталог, содержащий эти данные это `/var/www`. Следовательно, это может работать как точка монтирования раздела. Вы можете посмотреть в Рис. 8-1 и Рис. 8-2 ситуацию в системе до и после монтирования файловой системы.

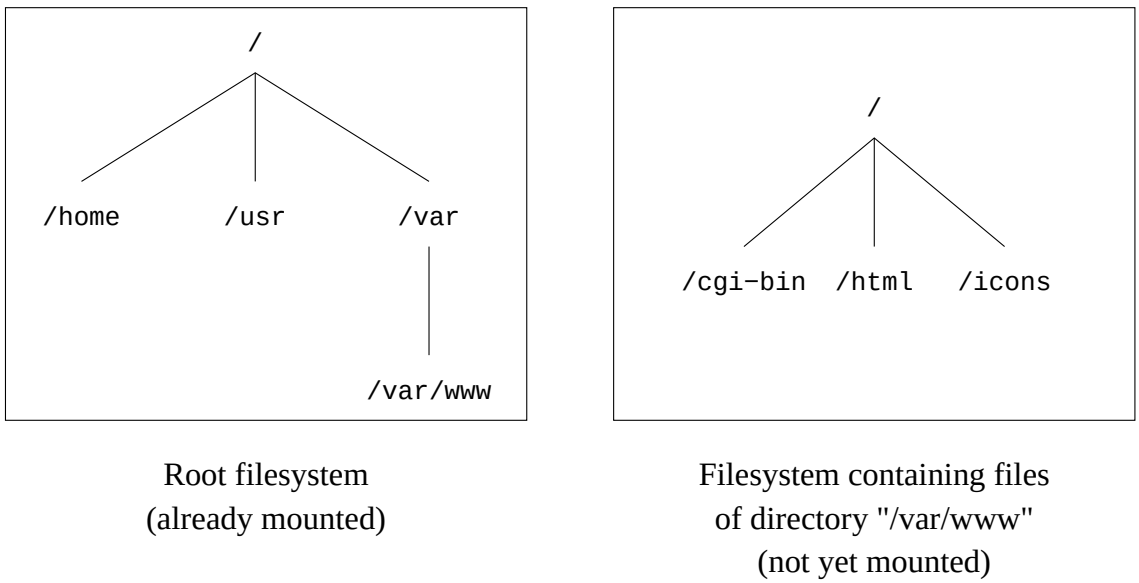

Рисунок 8-1. Еще не примонтированная файловая система

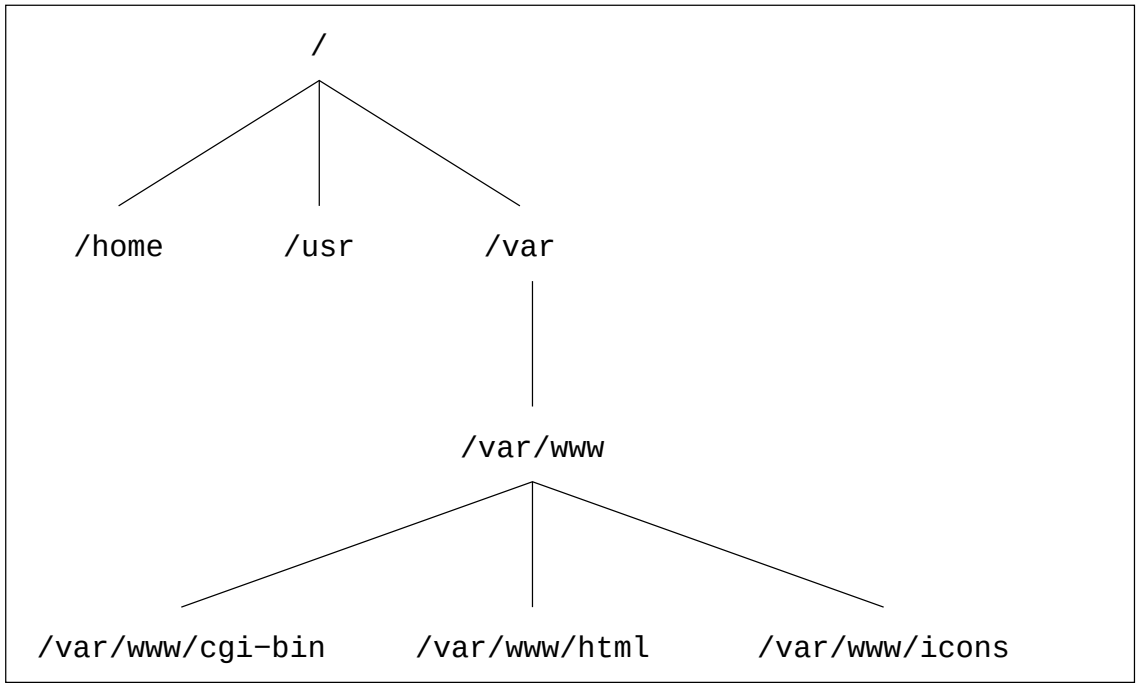

Рисунок 8-2. Файловая система теперь примонтирована

Как вы себе можете представить, это дает множество преимуществ: древовидная структура всегда будет одинаковой, и это распространяется как на одну так и на множество файловых систем. [1]. Всегда можно физически перенести ключевую часть древовидной структуры на другой раздел, когда кончается дисковое пространство, что мы и собираемся сделать здесь.

Есть две вещи, которые вам надо знать о точках монтирования:

1. каталог, который работает как точка монтирования должен существовать;
2. и этот каталог **предпочтительно должен быть пустым**: если в каталоге, который выбран как точка монтирования, будут находиться файлы и подкаталоги, они просто будут "скрыты" вновь примонтированной файловой системой и не будут доступны до тех пор, пока вы не отмонтируете файловую систему.

8.2. Разделение Жесткого Диска, Форматирование Разделов

Исходя из принципов, упомянутых выше, рассмотрим задачу нашего раздела: жесткий диск должен быть разбит на разделы и в каждом из них будет размещена файловая система. Ваш новый жесткий диск еще не имеет разделов, поэтому начнем с разбиения диска на разделы. Чтобы это сделать, вы должны иметь права `root`.

Первое, что вы должны знать, это "имя" вашего жесткого диска (то есть какой файл определяет его). Предположим вы установили жесткий диск как slave на primary IDE интерфейсе, тогда его имя будет `/dev/hdb`.[2]. Пожалуйста прочитайте раздел *Управление Разделами Диска* в *Стартовое Руководство Пользователя*, где объясняется как разбивать диски. Обратите внимание, что *`DiskDrake`* также может создавать для вас файловые системы.

8.3. Команды mount и umount

Теперь, когда файловая система создана, вы можете примонтировать раздел. Изначально он будет пустым. Команда монтирования это `mount` и ее синтаксис следующий:

```
mount [options] <-t type> [-o mount options] <device> <mounting point>
```

1. *`GNU/Linux`* может управлять 64 файловыми системами одновременно.
2. Как определить имя диска объяснялось в *Руководство по Установке*.

В данном случае, мы хотим временно примонтировать наш раздел в `/mnt` (или любой другой выбранный вами каталог –; помните, что он должен существовать); комманда для монтирования свежесозданного раздела:

```
$ mount -t ext2 /dev/hdb1 /mnt
```

Опция `-t` используется для определения типа файловой системы раздела, предполагаемого для размещения. Среди файловых систем, с которыми вы столкнетесь, наиболее часто встречающиеся это ext2FS (файловая система *GNU/Linux*), VFAT (для всех разделов *DOS/Windows*:FAT 12, 16 или 32) и ISO9660 (CD-ROM файловая система). Если вы не укажете тип, `mount` попытается самостоятельно определить тип монтируемой файловой системы путем чтения superblock. Она редко ошибается при этом.

Опция `-o` используется для указания одной или нескольких параметров монтирования. Обратитесь к страницам руководства mount(8) за подробной информацией.

Теперь, когда вы примонтировали ваш новый раздел, вам нужно скопировать в него весь каталог `/usr`:

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

Когда все файлы уже скопированы, мы можем отмонтировать наш раздел. Используйте команду `umount` Синтаксис простой:

```
umount <mount point|device>
```

Таким образом, чтобы отмонтировать наш новый раздел, мы должны написать:

```
$ umount /mnt
```

или так:

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

Так как этот раздел должен “стать” каталогом `/usr`, мы должны объяснить это системе. Чтобы это сделать, мы редактируем:

8.4. Файл /etc/fstab

Файл `/etc/fstab` делает возможным автоматическое монтирование определенных файловых систем, что особенно нужно при загрузке системы. Он содержит ряд строк, описывающих файловые системы, их точки монтирования и другие параметры. Вот пример файла `/etc/fstab`:

```
/dev/hda1   /           ext2    defaults        1 1
/dev/hda5   /home       ext2    defaults        1 2
/dev/hda6   swap        swap    defaults        0 0
/dev/fd0    /mnt/floppy auto    sync,user,noauto,nosuid,nodev,unhide 0 0
/dev/cdrom  /mnt/cdrom  auto    user,noauto,nosuid,exec,nodev,ro 0 0
none        /proc       proc    defaults        0 0
none        /dev/pts    devpts  mode=0622       0 0
```

Строки содержат, по порядку:

- устройство монтируемой файловой системы;
- точку монтирования;
- тип файловой системы;
- параметры монтирования;
- ***флаг*** для `dump`, утилиты создания резервных копий;
- Порядок проверки для `fsck` (*FileSystem CheckK*).

Здесь **всегда** есть запись о корневой файловой системе. Раздел ***swap*** является специальным, поэтому его не видно в древовидной структуре и в поле точки монтирования для таких разделов всегда содержится ключевое слово `swap`. Подробную информацию о файловой системе `/proc` читайте в разделе Гл. 10. Другая особая файловая система это `/dev/pts`.

Давайте теперь вернемся к теме. Вы перенесли всю иерархию `/usr` на `/dev/hdb1` и теперь нам нужно, чтобы этот раздел монтировался как `/usr` во время загрузки системы. Для этого нужно добавить запись в файл:

```
/dev/hdb1       /usr         ext2    defaults  1 2
```

Теперь раздел будет монтироваться при каждой загрузке. Кроме того, он будет проверяться если необходимо.

Есть две специальные опции: `noauto` и `user`. Опция `noauto` указывает, что файловая система не должна монтироваться при загрузке, но ее можно монтировать тогда, когда вы укажете это делать. Опция `user` указывает, что любой пользователь может монтировать или размонтировать эту файловую систему. Эти две опции обычно используются для CD-ROM и floppy устройств. Файл `/etc/fstab` содержит еще множество опций, с которыми можно ознакомиться на страницах руководства `(man)` (fstab(5)).

Одно из его главных преимуществ заключается в том, что он упрощает синтаксис команды `mount`. Чтобы примонтировать файловую систему, описанную в нем, достаточно обратиться к точке монтирования или устройству. Чтобы примонтировать floppy диск, вы можете написать:

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

или:

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

Закончим с нашим примером переноса раздела: мы скопировали иерархию `/usr` и прописали в `/etc/fstab` чтобы новый раздел монтировался при загрузке системы. Но файлы старого раздела `/usr` еще на диске! Следовательно мы должны удалить их чтобы очистить дисковое пространство (что изначально и было нашей целью). Чтобы это сделать, вам сначала нужно загрузиться в single user mode (запустить команду `telinit 1` в командной строке) , а затем:

- удалить все файлы из каталога `/usr` (то есть из “старого”, пока “новый” еще не примонтирован): `rm -Rf /usr/*`;
- примонтировать “новый” `/usr`: `mount /usr/`.

Вот и все. Теперь возвращаемся в multiuser режим (`telinit 3` или `telinit 5`), и, если административной работы больше не осталось, выходим из аккаунта `root`

8.5. Заметка о Возможности Supermount

Новые ядра, поставляемые с **Mandrake Linux** предоставляют интересную возможность для пользователей, который часто используют floppy и CD диски. Инсталлированная (или нет) в зависимости от выбранного уровня безопасности, она автоматически монтирует и отмонтирует носители по мере того, как они вставляются или изымаются. Это весьма удобно и избавляет от необходимости запускать каждый раз `mount` или `umount`

Глава 9. Файловая Система Linux

Вполне естественно, что ваша *GNU/Linux* система размещена на вашем жестком диске в файловой системе. Здесь мы осветим различные аспекты, связанные с файловыми системами, а также рассмотрим возможности, которые они предоставляют.

9.1. Сравнение Нескольких Файловых Систем

В время инсталляции вы можете выбирать различные **файловые системы** для ваших разделов жесткого диска. Это означает, что вы можете форматировать ваши разделы согласно различным алгоритмам.

Если вы не специалист, то выбор файловой системы не очевиден. Мы предлагаем краткое описание трех из самых современных файловых систем, любая из которых является доступной под **Mandrake Linux**.

9.1.1. Различные Используемые Файловые Системы

9.1.1.1. Ext2FS

Ext2FS это аббревиатура от **Second Extended Filesystem** (по-русски - **расширенная файловая система номер два**) или проще **ext2**. Многие годы **ext2** была файловой системой по умолчанию в *GNU/Linux*. Ext2 заменила собой **Extended File System** (вот откуда появилось "Second" в названии). В "новой" файловой системе были исправлены некоторые проблемы, а также убраны ограничения.

Ext2FS уважает обычные стандарты для файловых систем Unix-типа. По своей концепции она была предназначена для развития, обеспечивая при этом большую ошибкоустойчивость и хорошую производительность.

9.1.1.2. Ext3

Как видно из названия, **Third Extended File System** (**Расширенная файловая система номер три**) - является наследником файловой системы Ext2FS. Ext3 совместима с Ext2, но обладает одной новой и очень интересной особенностью - **журналирование**.

Одним из главных недостатков "традиционных" файловых систем, подобных Ext2FS, является низкая сопротивляемость к резким системным сбоям (сбой питания или авария программного обеспечения). Вообще говоря, такие события являются серьезным экзаменом для структуры файловой системы. Попытки исправить ошибки иногда приводят к увеличению искажений и появлению более серьезных ошибок в файловой системе. Результатом этого могут быть частичные потери данных в файловой системе.

Журналирование призвано решить эту проблему. Упрощенно можно сказать, что процесс сохранения объекта (например, сохранение файла) происходит **прежде** чем запись в журнал. Этот процес можно сравнить с тем, как капитан корабля пишет ежедневные события в бортовой журнал только после того, как они действительно произошли. В результате мы получаем всегда последовательную (когерентную) файловую систему. Это приводит к тому, что при появлении проблем, проверка и восстановление происходят очень быстро. Время, потраченное на то, чтобы проверить файловую систему таким образом, пропорционально его фактическому использованию и не больше его размера.

Ext3FS предлагает технологию журналирования файловой системы, сохраняя при этом структуру Ext2FS, что обеспечивает превосходную совместимость.

9.1.1.3. ReiserFS

В отличии от Ext3FS, **ReiserFS** создан на пустом месте. Это тоже журналируемая файловая система подобно Ext3FS, но их внутренняя структура радикально отличается. В **ReiserFS** используется концепция бинарных деревьев (binary-tree), позаимствованная из программного обеспечения баз данных.

9.1.1.4. JFS

JFS - сокращение от journalized filesystem (журналируемая файловая система). JFS была разработана и использовалась IBM. Вначале JFS была закрытой системой, но недавно IBM решила открыть доступ для движения свободного программного обеспечения. Внутренняя структура JFS близка к ReiserFS.

9.1.2. Различия Между Этими Файловыми Системами

	Ext2FS	Ext3FS	ReiserFS	JFS
Стабильность	Отличная	Хорошая	Хорошая	Средняя
Инструментальные средства для спасения удаленных файлов	Есть(комплексные)	Есть(комплексные)	Нет	Нет
Время перезагрузки после аварии	Долго (даже очень долго)	быстро	Очень быстро	Очень быстро
Восстанавливаемость данных в случае аварийного отказа	Хорошо, НО большой риск частичной или полной потери данных	Неизвестно	Очень хорошо. Полная потеря данных очень редка	Очень хорошо

Таблица 9-1. Характеристики файловой системы

Максимальный размер файлов зависит от большого количества параметров (например таких, как размер блока для ext2/ext3), а также от версии ядра и архитектуры. Тем не менее, доступный минимум, согласно ограничениям файловой системы, в настоящее время равен 2Tb (1Tb=1024 Gb) и может увеличен до 4Pb (1Pb=1024 Tb) для JFS. К сожалению, эти значения также ограничены максимальным размером блочного устройства, который в текущих версия ядер `2.4.X` равен (только для архитектуры `X86`) 2TB [1] даже в RAID режиме. Для получения дополнительной информации, проконсультируйтесь здесь Adding Support for Arbitrary File Sizes to the Single UNIX Specification (`http://ftp.sas.com/standards/large.file/x_open.20Mar96.html`).

9.1.3. Размышления о Производительности

Всегда очень сложно сравнивать производительность. Каждый тест имеет ограничения и его результаты должны быть интерпретированы внимательно и с осторожностью. В настоящее время, Ext2FS - очень зрелая файловая система, но её развитие недостаточно. С другой стороны, журналируемые файловые системы, подобные Ext3FS и ReiserFS развивается очень быстро. Проверки, сделанные пару месяцев или недель назад, уже слишком стары. Давайте не забывать, что сегодняшний материал тестирования (особенно это касается жестких диков) очень быстро устаревает. Однако JFS в настоящее время показывает лучшие результаты по производительности.

Каждая из систем предоставляет преимущества и имеет свои недостатки. Фактически, все зависит, от того как вы используете вашу машину. Для простой настольной машины (рабочей станции) вполне подойдет Ext2FS. Для сервера предпочтительно использовать журналируемую файловую систему Ext3FS. ReiserFS больше подходит для серверов баз данных. JFS предпочтителен в случаях, когда главная задача файловой системы - это производительность.

При “нормальном” использовании, все четыре файловые системы дают приблизительно одинаковые результаты. ReiserFS позволяет предоставляет быстрый доступ к маленьким файлам и сравнительно медленный к большими файлами(много мегабайт). В большинстве случаев, преимущества от журналирующих способностей ReiserFS сводят на нет его недостатки.

1. Вы можете задаться вопросом, как достигнуть таких мощностей с жесткими дисками, что едва достигают 180Gb. Фактически, используя 3 RAID платы, поддерживающие 8*128Gb дисков, вы получите 3Tb ...

9.2. Всё Является Файлом

Стартовое Руководство Пользователя описывает концепцию прав доступа к файлам, а также понятия владельца файла, но в действительности для ***файловых систем*** *UNIX* (это относится и к *GNU/Linux' ext2fs*) требуется, чтобы мы ввели определение файла.

В данном случае, "всё" **действительно** означает всё. Жесткий диск, разделы на жестком диске, параллельные порты, подключение к web-сайтам, *Ethernet* карточка, все они являются файлами. Даже каталоги - это файлы. В *GNU/Linux* существует много типов файлов в дополнение к стандартным файлам и каталогам. Обратите внимание, здесь под типом файла, мы не подразумеваем **содержимое** файла: в *GNU/Linux* и любой другой *UNIX* системах, файл, будь это текст, или PNG картинка, или двоичный файл, или что-то подобное, является только потоком байтов. Разделение файлов согласно их содержанию оставлено приложениям.

9.2.1. Различные Типы Файлов

Если вы помните, при вводе команды `ls -l`, символ перед правами доступа идентифицирует тип файла. Мы уже видели два типа файлов: обычный файл (`-`) и каталог (`d`). Блуждая по диску, вы также можете встретить и другие типы файлов:

1. ***Символьные файлы***: это любые специальные системные файлы (типа `/dev/null`, который мы уже обсуждали), или периферийные устройства (последовательные или параллельные порты), которые разрешают совместное использование своего содержания без ***буферизации*** (их значение не сохраняется в памяти). Такие файлы идентифицированы символом `c`.
2. ***Файлы блочного доступа***: Эти файлы - являются периферийными устройствами, и, в отличие от символьных файлов, их содержание - **буферизируется**. В эту категорию входят такие файлы, как например: жесткие диски, разделы на жестком диске, дисководы для гибких дискет, CD-ROM, и так далее. Примерами файлов блочного доступа могут служить файлы `/dev/hda`, `/dev/sda5`. В результате выполнения команды `ls -l`, они идентифицированы символом`b`.
3. ***Символические ссылки***: эти файлы очень широко используется в **Mandrake Linux**, например в процедуре запуска системы (см. главу Гл. 11). Название этих файлов отражает цель. Символические ссылки связаны с файлом символическим способом, что означает, что такие файлы могут указывать на существующий файл. Подробнее об этом позже в этой главе. Символические ссылки очень часто (и неправильно, что будет показано позже) называют ***"мягкими (soft) ссылками"***. Такие файлы идентифицируются символом `'l'`.
4. ***Именованные каналы (pipes)***: очень похожи на каналы, используемые в *shell*, фактически разница только в том, что такие каналы имеют название. Именованные каналы очень редки. Маловероятно, что вы встретите хотя-бы один во время исследования файлового дерева. На всякий случай сообщаем что символ, идентифицирующий их - `'p'`. Чтобы узнать больше об этом, взгляните сюда : Разд. 9.4.
5. ***Сокеты*** (socket: розетка, разъем): Этот тип файла для всех сетевых подключений. Только некоторые из них имеют названия. Нужно заметить, что сокеты бывают нескольких типов, но эта тема выходит за рамки данной книги. Такие файлы идентифицированы символом `'s'`.

Вот примеры каждого типа файлов:

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

9.2.2. Inode

Inode является, вместе с принципом "Все есть файл", фундаментальной частью файловой системы *UNIX*. Слово ***"inode"*** это сокращение от *Information NODE* (Информационный УЗЕЛ).

Inodes хранятся на диске в **inode table** (таблице информационных узлов). Они существуют для всех типов файлов, которые могут храниться в файловой системе, и это включает каталоги, именованные каналы, файлы символьного режима и так далее. Что приводит к другому известному выражению: "inode есть файл". При помощи inode *UNIX* идентифицирует файл уникальным способом.

Иначе говоря *UNIX* **не идентифицирует файл по его имени**. Вместо этого используется номер [2] из таблицы inode. Причина для этого заключается в том, что файл может иметь несколько названий, или вообще не иметь никакого названия. Имя файла в *UNIX*, является только указателем на inode. Такой указатель называется **link** (линк или ссылка). Давайте рассмотрим ссылки более подробно.

9.3. Ссылки

Лучший способ понять что такое ссылка - рассмотреть пример. Давайте создадим обычный файл:

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

Параметр `-i` для команды `ls` отображает номер inode первым значением в выводимой строке. Как вы можете видеть, до создания файла `a`, в каталоге не было никаких файлов. Для нас представляет интерес также и третье поле, которое показывает количество ссылок на данный файл (inode всегда ссылается - поэтому одна ссылка существует всегда).

Команда `touch a` выполняет два независимых действия:

- Создает inode, которому операционная система дала номер 32555, и чей тип - обычный файл;
- А также создает ссылку на этот inode, названную `a`, в текущем каталоге `/home/queen/example`. Поэтому файл `/home/queen/example/a` это ссылка на inode, с номером 32555, и поэтому верно количество ссылок 1 (одна ссылка).

а теперь, если мы напечатаем следующее:

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

мы создадим другую ссылку на тот же самый inode. Как вы можете видеть, мы не создавали файл с названием `b`, но вместо этого мы только добавили другую ссылку на inode с номером 32555 в том же самом каталоге и присвоили этой новой ссылке имя `b`. Рассмотрим результаты работы команды `ls -l`, Теперь счетчик ссылок на inode с номером 32555 равен 2, так как на этот самый inode ссылаются два файла `a` и `b`.

Теперь, если мы сделаем следующее:

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

мы увидим, что даже когда мы удалили "оригинальный файл", inode все еще существует. Но теперь единственная ссылка на этот inode - это файл с именем `/home/queen/example/b`.

2. Важно: обратите внимание, что номер inode уникален **в пределах файловой системы**, что означает, что inode с тем же самым номером может существовать на другой файловой системе. Это приводит к различию между дисковым inode и inode в оперативной памяти. Два дисковых inode могут иметь один и тот же номер, если они находятся на двух различных файловых системах. В оперативной памяти inode имеет уникальный номер на всю систему.

Следовательно, файл под *UNIX* не имеет никакого названия; вместо этого, он имеет одну или более ***ссылок*** на себя в одном или более каталогах.

Сами каталоги также размещены в inodes, но количество ссылок на них, в отличие от всех других типов файлов, совпадает с числом подкаталогов. На каталог всегда есть по крайней мере две ссылки: каталог непосредственно (`.`) и его каталог предыдущего уровня (`..`).

Типичным примером файла, на который никто не ссылается, (то есть он не имеет имени) есть сетевое подключение: вы никогда не сможете увидеть на своем диске файл, соответствующий вашему сетевому подключению к web-сайту Linux Mandrake (`www.mandrakelinux.com`). Подобная ситуация возникает в случае, когда вы используете ***канал*** в *shell*. В этом случае существует inode, соответствующий этому каналу, но на этот inode никто не ссылается. То есть он не имеет имени.

9.4. "Анонимные" Каналы и Именованные Каналы

Давайте вернемся к примеру с каналами, поскольку это весьма интересно, а также является хорошей иллюстрацией понятия ссылок. Когда вы используете канал в командной строке, *shell* создает для вас канал и работает таким образом, что команда перед каналом пишет в него, а команда после трубы читает из канала. В данном случае мы наблюдаем неименованный канал (например такой как используется в *shell*) или именованный (смотрите выше) действительно похож на FIFO (*First In, First Out (Кто первым вошел, тот первым вышел)*). Мы уже приводили пример того, как используются каналы в *shell*, но давайте в для более глубокого понимания выполним еще одну демонстрацию:

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

В приведенном примере есть одна вещь, которую вы наверняка не заметили (потому что это происходит слишком быстро для того чтобы быть заметным). состоящая в том, что записи на каналах блокируются. Это означает, что во время выполнения команды `ls`, запись в канал блокируется до тех пор, пока процесс с другой стороны канала (в данном случае head) не прочитает канал. Чтобы визуализировать этот эффект, можно создавать именованные каналы, которые, в отличии от используемых *shell*, имеют имена (то есть на них существуют ссылки). [3] Для создания именованного канала используется команда `mkfifo`:

```
$ mkfifo a_pipe
$ ls -il
total 0
    169 prw-rw-r--    1 queen     queen        0 Dec 10 14:12 a_pipe|
  #
  # вы видите что счетчик ссылок показывает 1, результат ls
  # показывает что файл является каналом ('p').
  #
  # Также можно создать ссылку с помощью ln
  #
$ ln a_pipe the_same_pipe
$ ls -il
total 0
    169 prw-rw-r--    2 queen     queen        0 Dec 10 15:37 a_pipe|
    169 prw-rw-r--    2 queen     queen        0 Dec 10 15:37 the_same_pipe|
$ ls -d /proc/[0-9] >a_pipe
  #
  # Процесс заблокировался, так как никто не читает канал с другой стороны.
  # Чтобы приостановить процесс нажмите Ctrl+z
  #
zsh: 3452 suspended  ls -d /proc/[0-9] > a_pipe
  #
  # ...теперь отошлем процес в фоновый режим (background):
  #
$ bg
[1]  + continued  ls -d /proc/[0-9] > a_pipe
  #
  # а теперь почитаем из канала...
  #
```

3. На самом деле существуют и другие различия между этими типами каналов. Но они выходят за рамки этой книги.

```
$ head -5 <the_same_pipe
  #
  # ...процесс записи прервался....
  #
[1]  + 3452 done       ls -d /proc/[0-9] > a_pipe
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
#
```

Кстати, точно также можно заблокировать и чтение. Если выполнить приведенные выше команды в обратном порядке, мы увидим что команда `head` блокируется, в ожидании пока какой-нибудь процесс не выполнит чтение:

```
$ head -5 <a_pipe
  #
  # Программа заблокировалась, Для того чтобы приостановить : Ctrl+z
  #
zsh: 741 suspended  head -5 < a_pipe
  #
  # отправляем в фоновое выполнение...
  #
$ bg
[1]  + continued  head -5 < a_pipe
  #
  # ...ну и теперь покормите немножко этот канал :)
  #
$ ls -d /proc/[0-9] >the_same_pipe
$ /proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
[1]  + 741 done       head -5 < a_pipe
$
```

В предыдущем примере присутствует один нежелательный эффект: команда `ls` прерывается до того как команда `head` получила данные. Поэтому получается, что это действие нельзя выполнить без выполнения промежуточных действий в командной строке.

9.5. "Специальные" Файлы: Символьный режим и Блочный режим

Как уже говорилось, такими файлами могут быть любые файлы, созданные системой или периферийными устройствами на вашей машине. Мы также упомянули, что содержимое файлов блочного режима буферизируется, а файлы символьного режима не буферизируются. Для демонстрации этого вставьте дискету в дисковод и введите дважды следующую команду:

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

Как вы видите, происходит следующее: при первом исполнении команды содержимое дискеты было прочитано полностью, при вторичном выполнении команды физического обращения к дисководу не произошло. Такой эффект наблюдается потому, что содержание дискеты было буферизировано при первом выполнении команды – и вы не сменили дискету в дисководе.

А теперь, таким же образом попробуем отпечатать большой файл (это должно работать):

```
$ cat /a/big/printable/file/somewhere >/dev/lp0
```

Команда будет выполняться медленно, независимо от того запустили вы ее один раз, два раза или пятьдесят раз. Это происходит потому, что `/dev/lp0` это файл символьного режима, и его содержимое не буферизируется.

Тот факт, что файлы блочного режима буферизированы, имеет хороший побочный эффект: кроме того, что буферизируется чтение, также буферизируется и запись. Это приводит к тому, что запись на диск происходит асинхронно: в момент когда вы записываете файл, немедленной записи на диск не происходит. Реально запись на диск произойдет только тогда, когда *GNU/Linux* примет решение о необходимости этого.

Каждый из специальных файлов имеет ***главный*** и ***вторичный*** номер. При выполнении команды `ls -l` эти номера отображаются на месте размера файла, так как размер для таких файлов не может быть вычислен:

```
ls -l /dev/hda /dev/lp0
brw-rw----    1 root     disk       3,   0 May  5  1998 /dev/hda
crw-rw----    1 root     daemon     6,   0 May  5  1998 /dev/lp0
```

Здесь главный и вторичный номер для файла `/dev/hda` будут 3 и 0 соответственно, а для файла `/dev/lp0`, это будут числа 6 и 0. Обратите внимание, что эти номера уникальны в категории файла, что означает, что может существовать файл символьного режима с главным номером 3, и младшим 0 (этот файл фактически существует: `/dev/ttyp0`), и точно так же может быть файл блочного режима с главный номером 6 и младшим 0. Эти номера существуют для того чтобы ***GNU/Linux*** могла правильно связывать операции с этими файлам (то есть это указатели на периферийные устройства): вы обычно не работаете с дисководом таким же образом, как и (например) с жесткими дисками SCSI

9.6. Символические Ссылки. Ограничения "Жестких" Ссылок

Здесь мы должны обратить внимание на очень распространенное заблуждение (даже среди пользователей ***UNIX***), которое появилось главным образом вследствие того, что ссылки, как мы видели раньше, (неправильно называемые "жесткими (hard)" ссылками), связаны только с обычными файлами (и мы видели, что это не имеет место для – тем более что даже символические ссылки связаны). Но это требует, чтобы мы сначала объяснили, что такое символические ссылки ("soft" ссылки или чаще "symlinks").

Символические ссылки - это файлы специфического типа. Единственное, что содержится в символических ссылках - это произвольная строка, которая может указывать (или не указывать) на существующий файл. Когда вы обращаетесь к символической ссылке из командной строки или из программы, фактически, вы обращаетесь к файлу, на который она указывает (разумеется, если символическая ссылка указывает на существующий файл). 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
```

вы можете видеть, что файл `mylink` имеет тип `'l'` что обозначает что он является символической *ссылкой*. Права доступа для символической ссылки не имеют значения: они будут всегда `rwxrwxrwx`. вы можете также видеть, что `myfile` и `mylink` - это два различных файла, поскольку их inode имеют различный номер. Фактически, при выполнении команды `cat mylink`, вы получите содержимое файла `myfile`. Для демонстрации того, что символическая ссылка может содержать произвольную строку, давайте проделаем следующее:

```
$ ln -s "I'm no existing file" anotherlink
$ ls -il anotherlink
    418 lrwxrwxrwx    1 queen     queen           20 Dec 10 21:43 anotherlink
-> I'm no existing file
$ cat anotherlink
cat: anotherlink: No such file or directory
$
```

Символические ссылки существуют потому, что они преодолевают некоторые ограничения, накладываемые при использовании жестких ссылок:

- вы не можете создать жесткую ссылку к inode в каталоге, который находится на другой файловой системе. Причина проста: счетчик ссылки сохраняется непосредственно в inode и не может совместно использоваться в различных файловых системах. Мягкие ссылки (symlinks) позволяют это.

- вы не можете сделать жесткую ссылку на каталог, поскольку мы видели, что счетчик ссылок для каталога используется для других целей. Но вы запросто можете сделать symlink на каталог и пользоваться этим симлинком точно также, как вы пользуетесь обыкновенным каталогом.

Поэтому символические ссылки (symlink) незаменимы во многих случаях. Часто люди используют симлинки даже тогда, когда можно обойтись использованием обыкновенных ссылок. Одно из преимуществ обыкновенных линков состоит в том, что при удалении "оригинального файла" вы не теряете содержимое файла.

Если вы внимательно следили за сказанным выше, то вы уже знаете что размер файла симлинка равен длине строки, содержащейся в симлинке, то есть равен длине названия

9.7. Атрибуты Файлов

В файловой системе FAT существуют атрибуты файла (arhive, system, hidden). Точно также, файловая система *ext2fs* тоже имеет атрибуты, правда эти атрибуты отличаются от атрибутов, принятых в FAT. Атрибуты в *ext2fs* используются очень редко, поэтому мы упоминаем о них только из соображений законченности повествования. Однако, если вас действительно интересует безопасность вашей системы, рекомендуем ознакомиться с этим разделом.

Существует две команды для управления атрибутами файла: `lsattr(1)` и `chattr(1)`. Как вы наверное догадались, команда `lsattr` служит для просмотра атрибутов, ну а команда `chattr` служит для изменения атрибутов. Атрибуты могут быть установлены только для каталогов и обыкновенных файлов. Доступны следующие атрибуты:

1. `A` (*no Access time*): если для файла или каталога установлен такой атрибут, то, при обращении к этому файлу (для чтения или записи), у файла не будет модифицироваться время последнего обращения. Это может быть полезно, например, при работе с файлами и каталогами, к которым очень часто обращаются, открывая их для чтения. Это может несколько разгрузить систему, так как время последнего доступа - это единственный параметр в inode, который изменяется при открытии файла в режиме read-only (только для чтения).
2. `a` (*append only*): если для файла или каталога установлен такой атрибут, а также этот файл (каталог) допускает запись, то единственная доступная операция записи - это добавление в конец файла (append). В случае каталога это означает, что вы можете только добавлять файлы, но не можете переименовывать или удалять любой существующий файл внутри этого каталога. Только `root` может устанавливать или очищать этот атрибут.
3. `d` (*no dump*): `dump (8)` это стандартная для *UNIX* утилита резервного копирования. Она создает резервную копию любой файловой системы, у которой в файле `/etc/fstab` значение dump counter установлено в 1 (подробнее смотрите Гл. 8). Если файл или каталог имеет этот атрибут, то он будет игнорирован при создании резервной копии файловой системы. Обратите внимание, что в случае установки атрибута для каталога, под действие атрибута рекурсивно попадают все нижележащие файлы и каталоги.
4. `i` (*immutable*): файл или каталог с таким атрибутом не может изменяться вообще: то есть, он не может быть переименован, на него не может быть создана ссылка. [4] Также такой файл не может быть удален. Только `root` может устанавливать или очищать этот атрибут. Обратите внимание, что этот атрибут также предотвращает изменения времени доступа, поэтому вы не должны совместно использовать атрибуты `A` и `i`.
5. `s` (*secure deletion*): при удалении файла, помеченного таким атрибутом, место на диске, которое занимал файл, будет заполнено нулями.
6. `S` (*Synchronous mode*): если установлен такой атрибут, то все изменения в файл будут записаны немедленно. Попросту говоря, этот атрибут снимает буферизацию записи для этого файла

Возможно имеет смысл установить атрибут `'i'` на некоторые жизненно важные системные файлы, чтобы избежать "неожиданностей". Также, можно проставить атрибут `'A'`, например, на файлах помощи (man pages): Это предотвратит лишние дисковые операции и даже слегка сэкономит батареи на портативных компьютерах.

4. Убедитесь, что вы поняли что обозначает "добавить ссылку" на файл или каталог **:-)**

Глава 10. Файловая Система /proc

Файловая система `/proc` является особой для *GNU/Linux*. Это виртуальная файловая система и она не занимает места на вашем диске. Это очень удобный способ получить информацию о системе, тем более, что большинство файлов в этом каталоге читаемы для человека (ну хорошо, с небольшой помощью). Большинство программ реально получают информацию из файлов в `/proc`, форматируют их своим способом, а затем отображают. Так делают все программы, которые отображают информацию о процессах, и некоторые из них мы уже видели (`top`, `ps` и сотоварищи). `/proc` это также хороший источник информации о ваших аппаратных средствах, и таким же образом некоторые программы являются только интерфейсами к информации, содержащейся в `/proc`.

Существует также специальный подкаталог `/proc/sys`. Он позволяет изменять некоторые параметры ядра в реальном режиме времени или отображать их.

10.1. Информация о Процессах

Если вы пролистаете содержимое каталога `/proc`, вы увидите много каталогов, названиями которых являются номера. Эти каталоги содержат информацию о все запущенных в данный момент процессах в системе:

```
$ 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/
```

Имейте в виду, что как пользователь, вы можете увидеть информацию только о своих собственных процессах. Для других пользователей вы такой информации не получите. Поэтому перейдем в `root` и посмотрим какая информация доступна для процесса 127:

```
$ su
Password:
$ cd /proc/127
$ ls -l
total 0
-r--r--r--    1 root     root            0 Dec 14 19:53 cmdline
lrwx------    1 root     root            0 Dec 14 19:53 cwd -> //
-r--------    1 root     root            0 Dec 14 19:53 environ
lrwx------    1 root     root            0 Dec 14 19:53 exe -> /usr/sbin/apmd*
dr-x------    2 root     root            0 Dec 14 19:53 fd/
pr--r--r--    1 root     root            0 Dec 14 19:53 maps|
-rw-------    1 root     root            0 Dec 14 19:53 mem
lrwx------    1 root     root            0 Dec 14 19:53 root -> //
-r--r--r--    1 root     root            0 Dec 14 19:53 stat
-r--r--r--    1 root     root            0 Dec 14 19:53 statm
-r--r--r--    1 root     root            0 Dec 14 19:53 status
$
```

Каждый каталог содержит одинаковые входы. Вот краткое описание некоторых из них:

1. `cmdline`: этот (псевдо-) файл содержит целую командную строку, использованную для вызова процесса. Он не отформатирован: нет пробелов между программой и ее аргументами, и нет разделителя в конце строки. Чтобы посмотреть его, вы можете использовать: `perl -ple 's,\00, ,g' cmdline`.
2. `cwd`: это символическая ссылка на текущий рабочий каталог (следует из имени) процесса.
3. `environ` Этот файл содержит все переменные окружения, определенные для процесса, в форме `VARIABLE=value`. Подобно `cmdline`, вывод не форматирован вообще: нет разделителей между различными переменными, и нет разделителя в конце. Единственное решение как его посмотреть: `perl -pl -e 's,\00,\n,g' environ`.
4. `exe`: это символическая ссылка на соответствующий запущенному процессу выполняемый файл.
5. `fd`: этот подкаталог содержит список файловых дескрипторов, открытых процессом в данный момент. Смотрите ниже.

6. `maps`: когда вы выводите содержимое именованной трубы (с помощью `cat`, например), вы можете видеть части адресного пространства процесса, которые в текущий момент отображаются в файле. Слева направо это поля: адресное пространство, связанное с этим отображением, права отображения, смещение с начала файла, где начинаются отображения, страший и младший номер (в шестнадцатиричном формате) устройства, где хранится файл отображения, номер узла файла и, последнее, имя файла. Когда устройство обозначено как 0 и нет номера узла и имени файла - это анонимное отображение. Смотрите mmap(2).
7. `root`: Это символическая ссылка на корневой каталог, используемый процессом. Обычно это будет `/`, но посмотрите chroot(2).
8. `status`: этот файл содержит различную информацию о процессе: название выполняемой программы, его текущее состояние, его PID и PPID, его реальные и эффективные UID и GID, использование им памяти, и другое.

Если мы выведем список каталога `fd` для нашего процесса 127, мы получим следующее:

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

Фактически, это список файловых дескрипторов, открытых процессом. Каждый открытый дескриптор обозначен символической ссылкой, именем каждого номера дескриптора, и указателем на файл, открытый этим дескриптором. [1]. Обратите также внимание на права symlinks: это - единственное место, где они имеют смысл, поскольку они представляют права, с которыми файл, соответствующий дескриптору, был открыт.

10.2. Информация об Аппаратном Обеспечении

Кроме каталогов, связанных с различными процессами, `/proc` также содержит несметное количество информации об аппаратном обеспечении вашей машины. Список файлов каталога `/proc` показывает следующее:

```
$ ls -d [a-z]*
apm      dma          interrupts  loadavg  mounts      rtc       swaps
bus/     fb           ioports     locks    mtrr        scsi/     sys/
cmdline  filesystems  kcore       meminfo  net/        self/     tty/
cpuinfo  fs/          kmsg        misc     partitions  slabinfo  uptime
devices  ide/         ksyms       modules  pci         stat      version
$
```

Если мы посмотрим, например, содержимое `/proc/interrupts`, мы можем увидеть, что тут есть список прерываний, используемых системой на данный момент, а также периферийных устройств, которые держат их. Точно так же, каталог `ioports` содержит список адресных интервалов ввода-вывода, занятых в настоящее время , и наконец, `dma` делает то же самое для каналов DMA. Поэтому, чтобы найти конфликт, нужно смотреть содержание этих трех файлов:

```
$ cat interrupts
           CPU0
  0:     127648          XT-PIC  timer
  1:       5191          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  5:       1402          XT-PIC  xirc2ps_cs
  8:          1          XT-PIC  rtc
 10:          0          XT-PIC  ESS Solo1
 12:       2631          XT-PIC  PS/2 Mouse
 13:          1          XT-PIC  fpu
 14:      73434          XT-PIC  ide0
```

1. Если вы помните, что рассказывалось в разделе Разд. 3.4, вы знаете, что обозначают дескрипторы 0, 1 и 2.

```
 15:      80234          XT-PIC  ide1
NMI:          0
$ cat ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
0300-030f : xirc2ps_cs
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
1050-1057 : ide0
1058-105f : ide1
1080-108f : ESS Solo1
10c0-10cf : ESS Solo1
10d4-10df : ESS Solo1
10ec-10ef : ESS Solo1
$ cat dma
 4: cascade
$
```

Или проще, использовать команду `lsdev`, которая получает информацию из этих трех файлов и сортирует ее по периферийным устройствам, что, несомненно, более удобно. [2]:

```
$ lsdev
Device            DMA   IRQ  I/O Ports
------------------------------------------------
cascade             4     2
dma                           0080-008f
dma1                          0000-001f
dma2                          00c0-00df
ESS                           1080-108f 10c0-10cf 10d4-10df 10ec-10ef
fpu                          13  00f0-00ff
ide0                         14  01f0-01f7 03f6-03f6 1050-1057
ide1                         15  0170-0177 0376-0376 1058-105f
keyboard                      1  0060-006f
Mouse                        12
pic1                          0020-003f
pic2                          00a0-00bf
rtc                           8  0070-007f
serial                        03f8-03ff
Solo1                        10
timer                         0  0040-005f
vga+                          03c0-03df
xirc2ps_cs                    5  0300-030f
$
```

Полная распечатка файлов была бы слишком длинной, тем не менее приведем описание некоторых из них:

- `cpuinfo`: этот файл содержит, соответственно своему названию, информацию о процессоре(рах) в вашей машине.
- `modules`: этот файл содержит список модулей, которые используются ядром в данный момент, а также процент использования для каждого модуля. Фактически, это та же самая информация, которую предоставляет команда `lsmod`.
- `meminfo`: этот файл предоставляет информацию о загрузке памяти на момент запроса. Более аккуратно форматированный вид этой же информации можно получить командой `free`.
- `apm`: если у вас портативный компьютер, то вывод этого файла предоставит вам информацию о состоянии батарей. Вы сможете увидеть куда подключен AC, текущую загрузку батарей, и, если APM *BIOS* вашего ноутбука поддерживает это (к сожалению это не всегда так), то вы сможете увидеть

2. `lsdev` - это часть пакета `procinfo`.

еще и сколько времени в минутах батареи смогут проработать. Файл не очень читаем, поэтому лучше использовать вместо него команду `apm`, которая предоставит информацию в "человекочитаемом" виде.

- `bus`: этот подкаталог предоставит вам информацию о всех периферийных устройствах, найденных на различных шинах вашего компьютера. Информация внутри него редко вообще читаема, и главным образом с ней имеют дело и умеют форматировать внешние утилиты, такие как: `lspcidrake`, `lspnp`, и др..

10.3. Подкаталог `/proc/sys`

Задача этого каталога - сообщать о различных параметрах ядра и позволять замену некоторых из них в реальном режиме времени. В противоположность всем другим файлам каталога `/proc`, некоторые файлы этого каталога могут быть записаны, но только под `root`

Список каталогов и файлов был бы слишком большим, тем более что их наличие преимущественно зависит от конкретно вашей системы, а большинство файлов будет использоваться только для очень специализированных приложений. Тем не менее, приведем три обычных случая использования этого подкаталога:

1. Разрешение роутинга: Даже если заданное по умолчанию ядро от **Mandrake Linux** может роутить, вы должны явно позволить ему это делать. Для этого нужно под `root` напечатать следующее:

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

   Замените 1 на 0, если вы хотите запретить роутинг.

2. Предотвращение подмены IP: имитация IP состоит в том, чтобы заставить интерфейс поверить в то, что пакет, пришедший из мира, является его собственным, вышедшим из него пакетом. Эта техника очень часто используется ***кракерами*** [3], но вы можете заставить ядро предотвращать такие вторжения. Вам только нужно написать:

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

   и все типы таких атак становятся невозможными.

3. Увеличение размера таблицы открытых файлов и таблицы inode: Размер таблицы открытых файлов и таблицы inode является динамическим под *`GNU/Linux`*. Значений по умолчанию обычно достаточно, но они могут быть недостаточными если ваша машина - нагруженный сервер (например, сервер баз данных). Реально, первым препятствием может стать тот факт, что процессы больше не смогут открывать файлы по той причине, что таблица полна, поэтому вам нужно увеличить ее размер. В то же время вам нужно будет увеличить и размер таблицы inode. Вот две строчки, которые решают эту проблему:

   ```
   $ echo 8192 >/proc/sys/fs/file-max
   $ echo 16384 >/proc/sys/fs/inode-max
   ```

Чтобы это выполнялось при каждой загрузке системы, вы можете добавить все эти строки в `/etc/rc.d/rc.local` и таким образом избежать их набора каждый раз заново, но есть и другое решение, состоящее в том, чтобы заполнить файл `/etc/sysctl.conf`, смотрите sysctl.conf(5).

3. Но не ***хакерами***!

Глава 11. Файлы Загрузки: init sysv

Традиционно для *UNIX* есть две схемы загрузки системы: схема *BSD* и схема "*System V*", обе получившие название после того, как были представлены в *UNIX* (соотв. *Berkeley Software Distribution* и **AT&T** *UNIX System V*). Схема *BSD* более простая, но схема *System V*, хотя и менее понятная сразу (после прочтения этой главы вы измените свое мнение), определенно более гибкая в использовании.

11.1. В Начале Был init

Когда система стартует, а после этого ядро конфигурирует все и монтирует корневую файловую систему, она выполняет команду `/sbin/init` [1]. `init` это отец всех процессов в системе, и он также отвечает за перевод системы в нужный ***runlevel***. Мы рассмотрим runlevel в следующем разделе.

Конфигурационнный файл `init` это `/etc/inittab`. У этого файла есть собственная страница руководства (inittab(5)), но мы все-таки опишем здесь несколько конфигурационных настроек.

Первая строка, на которую нужно обратить внимание это:

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

Эта инструкция сообщает `init` что `/etc/rc.sysinit` должен быть выполнен при инициализации системы (`si` (*System Init*) стоит раньше всего другого. Для того, чтобы определить runlevel по умолчанию, `init` ищет строку с ключевым словом `initdefault`:

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

Для этого случая `init` знает, что по умолчанию runlevel 5. Кроме того он знает, что для входа в runlevel 5, он должен запустить команду:

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

Как вы могли заметить, синтаксис каждого runlevel одинаков.

`init` также отвечает за перезагрузку (`respawn`) некоторых программ, которые являются единственным процессом, способным к перезапуску. Это случай, например, для всех login программ , которые запускаются в каждой из 6-ти виртуальных консолей. [2]. Для второй виртуальной консоли это выглядит так:

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

11.2. Режимы Выполнения (runlevels)

Все файлы, имеющие отношение к загрузке находятся в каталоге `/etc/rc.d`. Вот список файлов:

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

В начале, как мы видим, запускается файл `rc.sysinit`. Этот файл отвечает за установки базовой конфигурации машины: тип клавиатуры, конфигурация некоторых устройств, проверка файловой системы и т.д.

Затем запускается скрипт `rc`, с желаемым номером runlevel в качестве аргумента. Как мы видим, runlevel это простое целое число, и для каждого runlevel `<x>` имеется соответствующий `rc<x>.d` каталог. В обычной инсталляции **Mandrake Linux** вы можете увидеть 6 определенных runlevel:

- 0: полная остановка машины;
- 1: ***single-user*** (однопользовательский) режим; используется в случае серьезных проблем или для восстановления системы;

1. Теперь вы видите, что размещение `/sbin` на отличной от корневой файловой системе - очень плохая идея **:-)**
2. Таким образом, вы можете, если нужно, добавлять или удалять виртуальные консоли, максимум до 64, изменяя этот файл и следуя синтаксису. Но не забудьте, что *X* тоже запускается в виртуальной консоли! Так что оставьте хотя бы одну для него.

- 2: ***multi-user*** (многопользовательский) режим, без поддержки сети;
- 3: Multi-user (многопользовательский) режим с поддержкой сети;
- 4: неиспользуемый;
- 5: такой же как и 3, но, кроме всего, запускает графический интерфейс для входа в систему (login);
- 6: перезагрузка.

Давайте посмотрим, например, на содержимое каталога `rc5.d`:

```
$ ls rc5.d
K15postgresql@  K60atd@      S15netfs@   S60lpd@        S90xfs@
K20nfs@         K96pcmcia@   S20random@  S60nfs@        S99linuxconf@
K20rstatd@      S05apmd@     S30syslog@  S66yppasswdd@  S99local@
K20rusersd@     S10network@  S40crond@   S75keytable@
K20rwhod@       S11portmap@  S50inet@    S85gpm@
K30sendmail@    S12ypserv@   S55named@   S85httpd@
K35smb@         S13ypbind@   S55routed@  S85sound@
```

Как вы видите, все файлы в этом каталоге это символические ссылки, и все они имеют специфический вид. Их общий вид такой:

```
<S|K><order><service_name>
```

`S` обозначает *Start* сервиса, а `K` обозначает *Kill* (остановку) сервиса. Скрипты запускаются в возрастающем порядке номеров, а если два скрипта имеют одинаковый номер, тогда действует алфавитный порядок. Мы также видим, что каждая символическая ссылка указывает на соответствующий скрипт в каталоге `/etc/rc.d/init.d` (кроме `local`), скрипт, который отвечает за контроль особого сервиса.

Когда система отправляется в заданный runlevel, все начинается с запуска `K` ссылок в таком порядке: `rc` ищет куда указывает ссылка, затем вызывает соответствующий скрипт с одним аргументом `stop`. Затем запускаются `S` скрипты, используя тот же метод, не учитывая того факта, что скрипт вызван с аргументом `start`.

Итак, не учитывая всех остальных скриптов, мы можем видеть, что когда система переходит в runlevel 5, сначала запускается `K15postgresql`, то есть `/etc/rc.d/init.d/postgresql stop`. Затем `K20nfs`, затем `K20rstatd`, и так до последнего; потом, запускаются все `S` скрипты: первым `S05apmd`, который вызывает `/etc/rc.d/init.d/apmd start`, и так далее.

Вооружившись всем этим, вы можете создать свой собственный целый runlevel за несколько минут, или предотвратить запуск или остановку сервиса путем удаления соответствующей символической ссылки (есть также программы с интерфейсом для выполнения этого, в особенности *`drakxservices`* и *`chkconfig`*; бывшая графическая программа)

Глава 12. Сборка и инсталляция свободного программного обеспечения

Меня часто спрашивают о том, как установить свободное программное обеспечение из исходников [1]. Сам процесс компилирования программного обеспечения действительно прост, так как действия, необходимые для этого, обычно одинаковы, независимо от того, какое программное обеспечение вы устанавливаете. Цель этого документа состоит в том, чтобы провести новичка шаг за шагом и объяснять ему значение каждого перемещения. Я предполагаю, что читатель имеет минимальные знания по использованию системы *UNIX* (например о том, как использовать команды `ls` или `mkdir`).

Это руководство является просто руководством, но ни как не справочником. Именно поэтому в конце приведено несколько ссылок, по которым вы можете найти ответы на вопросы, которые у вас могут возникнуть по ходу чтения этого раздела. Разумеется, это руководство может быть улучшено, поэтому любые замечания и исправления по его содержанию принимаются.

12.1. Введение

Различие между свободным программным обеспечением и коммерческим программным обеспечением состоит в доступности исходников [2]. Это значит, что свободное программное обеспечение поставляется как архив файлов исходного кода. Это может смутить новичков, так как перед тем, как использовать свободное программное обеспечение, его прийдется откомпилировать.

Большинство свободного программного обеспечения существует в откомпилированном виде. Вы можете просто установить пре-компилированные бинарники [3] Некоторое свободное программное обеспечение не распространяется в виде бинарников. Кроме того, если вы используете экзотическую операционную систему или экзотическое "железо" - многие программы будут доступны только в виде исходников. Более того, компилируя программное обеспечение самостоятельно, вы можете отключить ненужные вам опции или наоборот, включить только интересующие вас, или даже расширить функциональность компилируемой программы в соответствии со своими нуждами.

12.1.1. Требования

Для сборки программного обеспечения вам потребуются:

- компьютер с работающей операционной системой,
- базовые знания по использованию вашей операционной системы,
- некоторое количество свободного места на вашем диске,
- компилятор (обычно для языка программирования *C*) и архиватор (`tar`),
- некоторое количество провизии (в особо тяжелых случаях дело может затянуться). Настоящие хакеры едят пиццу, а не quiches (что-то типа пирога с овощами и мясом).
- чего нибудь попить (из тех же соображений). Настоящие хакеры пьют содовую – и кофеин.
- Телефонный номер вашего друга, который занимается перекомпиляцией ядра каждую неделю,
- много терпения для того, чтобы это сделать!

Компиляция программ из исходников вообщем не представляет проблем, но в случае, если вы к этому еще не привыкли, то вас может отпугнуть любое незначительное препятствие.

---

1. Словом "исходник" обозначается понятие "исходный код программы". Мы вводим этот термин в переводе для того, чтобы не строить сложных предложений.
2. Это не совсем верно, так как некоторое коммерческое программное обеспечение тоже доступно в исходных кодах. Но в этом случае пользователю запрещается использовать или изменять код коммерческого програмного обеспечения.
3. Словом "бинарник" обозначается понятие "бинарный файл", по аналогии с исходником. В данном случае под бинарником подразумевается бинарный файл, полученный из исходного кода программы путем компиляции. Это понятие мы применяем для упрощения структуры предложений в этой главе, хотя отдаем себе отчет в том, что это "жаргонное" выражение.

12.1.2. Компиляция

12.1.2.1. Принципы

Для того, чтобы транслировать исходный код в бинарный файл, необходимо выполнить ***компиляцию*** (compilation) (обычно это делают из исходных кодов на языках *C* или *C++*, которые являются самыми распространенными языками программирования (под *UNIX*) в сообществе свободного программного обеспечения). Некоторые свободные программы написаны на языках, которые не требуют компиляции (например, *perl* или *shell*), но им все-таки обычно необходима некоторая настройка.

Компиляция кода на *C* выполняется с помощью *C* компилятора, чаще всего `gcc`, это свободный компилятор, разработанный в рамках проекта GNU (`http://www.gnu.org/`). Полное компилирование программного продукта - это комплексная задача, которая включает в себя последовательную компиляцию множества различных исходных файлов (так проще для программистов - они могут помещать различные части своей работы в отдельных файлах). Для того чтобы было проще делать такие повторяющиеся операции, существует утилита с названием `make`.

12.1.2.2. Четыре шага компиляции

Чтобы понять как происходит компиляция (между прочим, это поможет в решении возникающих проблем), вы должны знать четыре шага компиляции. Объект постепенно конвертируется из текстового файла, написанного на языке, который понятен для квалифицированного человека (то есть язык *C*), в язык, который понятен для машины (или в некоторых случаях, частично, для **очень** квалифицированного человека). `gcc` выполняет последовательно четыре программы, каждая из которых делает один из упомянутых шагов:

1. `cpp`: На первом шаге происходит замена директив условной компиляции на чистые инструкции *C*. Это называется препроцессором (***preprocessor***). Как правило, это вставка заголовков (`#include`) или определение макрокоманд (`#define`). В конце этой стадии генерируется чистый код на *C*.
2. *cc1*: На этом шаге код на *C* конвертируется в код на ***языке ассеблера (assembly language)***. Сгенерированный код зависит от конечной архитектуры.
3. `as`: На этом шаге происходит генерация кода из языка ассемблера в ***объектный код (object code)*** (или бинарный код (binary code)). В конце этого шага генерируются файлы с расширением `.o`.
4. *ld*: На последней стадии происходит ***линковка (linkage)*** всех объектных файлов (`.o`) и необходимых библиотек, и, в результате получается выполняемый файл.

12.1.3. Структура распространения программ

Правильно сформированный набор исходных кодов программы обязательно имеет следующую организацию:

- В файле `INSTALL` описана процедура инсталляции.
- В файле `README` содержится общая информация, относящаяся к программе (короткое описание, автор, URL где можно найти эту программу, документы имеющие отношение к программе, относящиеся к делу ссылки, и т.д.). В случае отсутствия файла `INSTALL` в файле `README` должна находиться краткая информация по инсталляции.
- В файле `COPYING` должна содержаться лицензия или условия распространения программы. В некоторых случаях для этих целей используется файл `LICENSE`, который имеет тоже самое содержимое.
- В файле `CONTRIB` или в файле `CREDITS` содержится список людей, имеющих отношение к этому программному продукту (активные участники, их вклад в продукт и т.д.).
- Файл `CHANGES` (реже `NEWS`), содержит последние изменения и устраненные ошибки.
- Файл `Makefile` (смотри раздел Разд. 12.4.1) позволяет произвести компиляцию программного обеспечения (этот файл необходим для утилиты `make`). Этого файла часто может не существовать сразу, тогда он создается динамически в процессе настройки.
- Весьма часто файлы `configure` или `Imakefile` позволяют сгенерировать новый файл `Makefile`.

- Каталог, в котором содержатся исходные коды, и где после компиляции будут лежать бинарники обычно называется `src`.
- Каталог, в котором содержится документация, относящаяся к этому программному продукту (обычно в форматах `man` или *`Texinfo`*) называется `doc`.
- Иногда может поставляться каталог, в котором содержатся специфические для программы данные (обычно это файлы настройки, примеры получаемых данных или файлы ресурсов).

12.2. Распаковка

12.2.1. Архив `tar.gz`

Стандартным [4] форматом сжатия для систем *UNIX* является формат программы `gzip`, разработанный в рамках проекта GNU и соединяющий в себе все лучшее из основных утилит компрессии.

Утилита `gzip` неразрывно связана с утилитой `tar`. `tar` это старая программа, оставшаяся в живых с тех времен, когда компьютерщики сохраняли свои данные на магнитных лентах. В настоящее время дискеты и CD-ROM вытеснили магнитные ленты, но утилита `tar` все еще используется для создания архивов. Все файлы и каталоги могут быть слеплены в один файл и после этого легко сжаты с помощью утилиты `gzip`.

Это является причиной, по которой свободное программное обеспечение обычно доступно в виде архива `tar`, сжатого с помощью программы `gzip`. Поэтому расширение файла выглядит так: `.tar.gz` (иногда для краткости `.tgz`).

12.2.2. Использование `GNU Tar`

Для распаковки этого архива могут быть использованы `gzip` и `tar`. Но GNU версия программы `tar` (`gtar`) позволяет использовать `gzip` ***"на лету" (on-the-fly)*** , и распаковать и разархивировать файлы из архива одним действием (кстати, это экономит дополнительное место на диске по причине отсутствия промежуточного файла архива).

Формат использования `tar` выглядит так:

```
tar <file options> <.tar.gz file> [files]
```

Опция `<files>` необязательна. Если она опущена, то действие будет применено ко всему архиву. Этот аргумент не нужен для извлечения содержимого архива `.tar.gz`

Для примера:

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

Некоторые опции `tar`:

- `v` verbose. В этом случае имена всех файло, найденных в архиве, будут показаны на экране. Если эта опция опущена, процесс будет происходить без вывода какой-либо информации на экран.
- `f` Это обязательная опция. Без нее `tar` будет использовать вместо архива стример (то есть устройство `/dev/rmt0`).

4. Все чаще и чаще в новых программах начинают использовать `bzip2` - более эффективный компрессор для текстовых файлов (и требующий больших мощностей от компьютера). Смотрите раздел Разд. 12.2.3, где описаны особенности этой программы.

- `z` эту опцию нужно применять к архиву, сжатому gzip (у которых расширение `.gz`). Если упустить эту опцию, `tar` сгенерирует ошибку. И наоборот - эта опция не может быть использована с несжатыми архивами.

Программа `tar` позволяет производить разные действия с архивом (извлекать, читать, создавать, добавлять...). Для этих действий применяются следующие опции:

- `x`: позволяет извлечь файлы из архива.
- `t`: получить список содержимого архива.
- `c`: позволяет создать новый архив. Между прочим, вы можете использовать это для создания резервных копий ваших личных файлов.
- `r`: позволяет вам добавить файлы в конец архива. Этого нельзя делать в сжатых архивах.

12.2.3. `bzip2`

Формат сжатия `bzip2` был разработан для замены `gzip`. `bzip2` делает архивы меньшего размера, чем `gzip`, но это еще пока не стандарт. Совсем недавно стали попадаться архивы с расширением `.tar.bz2`.

`bzip2` используется также, как `gzip` посредством вызова команды `tar`. Разве что нужно заменить опцию `z` на `j`. Например:

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

В некоторых дистрибутивах программ вместо этого могут предлагать использовать опцию `I`:

```
$ tar xvfI foo.tar.bz2
```

Это же можно записать и длинным способом:

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

Перед тем как использовать `tar`, вы должны убедится, что программа `bzip2` установлена в каталог, который содержится в переменной окружения `PATH`.

12.2.4. Просто сделайте это!

12.2.4.1. Самый простой способ

Теперь, когда вы готовы к распаковке архива, не забудьте получить права `root`. Вам прийдется делать вещи, которые обычному пользователю делать не положено. Некоторые этапы все же можно делать не имея привилегированных прав, но проще сразу перейти в `root` на весь процесс установки программы.

Во-первых, зайдите в каталог `/usr/local/src` и скопируйте туда архив. Вы всегда сможете найти тут архив в случае потери программного обеспечения. Если у вас мало места на жестком диске, после компиляции сохраните исходные коды на дискетах. Вы также можете впоследствии просто удалить исходники, если вы уверены, что всегда сможете получить их в *Web*.

Правильно отработавшая программа `tar`, развернув архив, должна создать новый каталог. Зайдите в него. Теперь мы можем продолжить.

12.2.4.2. Самый безопасный путь

UNIX-подобные системы (такие как *GNU/Linux* и *FreeBSD*) являются безопасными системами. Это значит, что обычные пользователи не могут произвести действия, угрожающие работоспособности системы (например, отформатировать диск). Также обычный пользователь не имеет доступа к файлам других пользователей. Кстати, это придает системе иммунитет от вирусов.

С другой стороны, `root` может делать все. Наличие исходных кодов делает возможным проверку программы на наличие злонамеренного кода (вирусы и трояны). Проявите бдительность [5].

5. В мире BSD существует пословица: “Никогда не доверяйте пакету, к которому у вас нет исходных кодов (Never trust a package you don’t have the sources for).”

Идея состоит в том, чтобы создать пользователя специально для администрирования (например `free` или `admin`) путем использования командочки `adduser`. Этот пользователь должен иметь права на запись в следующих каталогах: `/usr/local/src`, `/usr/local/bin` и `/usr/local/lib`, а также все подкаталоги в `/usr/share/man`. Мы рекомендуем сделать этого пользователя владельцем этих каталогов или создать группу для него и разрешить этим каталогам запись для этой группы.

Как только все эти предосторожности приняты, вы можете следовать указаниям в разделе Разд. 12.2.4.1.

12.3. Конфигурирование

Большой технический интерес представляет тот факт, что авторы, создавая исходные коды, делают их ***портируемыми***. Свободное программное обеспечение, разработанное для систем *UNIX*, может быть использовано во всех существующих системах *UNIX* (свободных или коммерческих) с минимальными изменениями. Для этого необходим процесс конфигурирования перед компиляцией.

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

- Использовать *AutoConf* (см. раздел Разд. 12.3.1), если в поставке существует файл с названием `configure`.
- Использовать *imake* (см раздел Разд. 12.3.2), если в поставке существует файл с названием `Imakefile`.
- Выполнить ***скрипт*** (например `install.sh`) согласно инструкциям, содержащимся в файлах `INSTALL` или `README`

12.3.1. AutoConf

12.3.1.1. Принципы

Программа *AutoConf* используется для правильной настройки программного обеспечения. Она создает файлы, необходимые для компиляции (такие как `Makefile`) и кое что изменяет прямо в исходных кодах (например `config.h.in`).

Принципы работы *AutoConf* просты:

- Программист, разрабатывающий данную программу, знает какие проверки необходимо сделать для настройки этой программы (например: “какая версия ***библиотеки*** у вас установлена?”). Программист записывает эти проверки (в специальном формате) в файл с именем `configure.in`.
- После этого программист запускает программу *AutoConf*, которая из файла `configure.in` создает конфигурационный скрипт с названием `configure`. Этот скрипт делает все необходимые проверки, какие запланировал программист.
- Конечный пользователь запускает этот скрипт и *AutoConf* настраивает все что нужно для компиляции.

12.3.1.2. Пример

Пример использования *AutoConf*:

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

```
checking for X... libraries /usr/X11R6/lib, headers /usr/X11R6/include
checking for ANSI C header files... yes
checking for unistd.h... yes
checking for working const... yes
updating cache ./config.cache
creating ./config.status
creating lib/Makefile
creating src/Makefile
creating Makefile
```

Есть возможность внести кое-какие изменения в работу скрипта `configure` путем внесения некоторых опций в командную строку скрипта или в переменные окружения. Пример:

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

или (при помощи *bash*):

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

или так:

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

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

Обычно это ошибка такого типа: `configure: error: Cannot find library guile` (большинство ошибок конфигурации, выдаваемые `configure` напоминают эту).

Это обозначает, что скрипт `configure` не смог найти библиотеку (в данном примере это библиотека `guile`). Принцип проверки заключается в том, что скрипт `configure` для проверки компилирует маленькую программу, которая использует искомую библиотеку. Если компиляция этой тестовой программы не завершилась успешно, то ошибка произойдет и при компиляции нашего программного обеспечения.

- Причину ошибки можно найти в конце файла `config.log`, в котором содержится журнал всех действий, предпринятых при настройке. Компилятор *C* обычно выдает довольно понятные сообщения об ошибках. Этот факт должен помочь вам в решении проблемы.
- Производится проверка того, что нужная библиотека установлена надлежащим образом. Если она не установлена, то ее необходимо установить (из исходников или бинарников) и запустить `configure` повторно. Эффективным способом проверки является поиск файлов, содержащихся в библиотеке; это практически всегда `lib<name>.so`. Например так:

  ```
  $ find / -name 'libguile*'
  ```

  или вот так (в этом случае поиск проходит быстрее):

  ```
  $ locate libguile
  ```

- Проверить, доступна ли библиотека для вашего компилятора. То есть библиотека должна находиться в каталогах: `/usr/lib` , `/lib` , `/usr/X11R6/lib` (или в других каталогах, список которых содержится в переменной окружения `LD_LIBRARY_PATH`, которые описываются в Разд. 12.4.5 номер 5.b. Проверьте, является ли этот файл библиотекой. Для этого выполните команду `file libguile.so`.
- Проверьте, что заголовочные файлы библиотеки (хедеры, headers) установлены в правильном месте (обычно `/usr/include` или `/usr/local/include` или `/usr/X11R6/include`). Если вы не знаете, какой из хедеров вам нужен, проверьте что установлена версия для разработки (development) данной библиотеки (например для `libgtk+2.0` должен быть установлен пакет `libgtk+2.0-devel`). Версия пакета библиотек для разработки содержит в себе файлы “include”, которые необходимы для компиляции программ, использующих эту библиотеку.
- Проверьте наличие свободного места на диске (скрипт `configure` требует некоторого количества свободного места на диске для размещения временных файлов). Используйте команду `df -h` для отображения разделов вашего диска и степени их заполнения.

Если для вас непонятны сообщения, находящиеся в файле `config.log` , не стесняйтесь спрашивать помощи у сообщества свободного программного обеспечения (см. раздел Разд. 12.6.2).

Кроме того, проверьте не отвечает ли `configure` в 100% `No` и разберитесь, не отвечает ли этот скрипт `No`, когда вы точно знаете, что библиотека установлена. Например, было бы очень странно, если в вашей системе отсутствовала бы библиотека `curses`. В этом случае, у вас вероятно неправильно установлена переменная окружения `LD_LIBRARY_PATH`!

12.3.2. `imake`

`imake` позволяет вам производить настройку свободного програмного обеспечения путем создания файла `Makefile` по простым правилам. Эти правила определяют, какие файлы необходимы компилятору для создания бинарного файла и, пользуясь этими данными, программа *`imake`* генерирует соответствующий `Makefile`. Эти правила находятся в файле с названием `Imakefile`.

Интересно то, что программа *`imake`* использует архитектурно-зависимую информацию (***site***) Это весьма удобно для приложений, которые используют *`X Window System`*. Тем не менее, программа *`imake`* используется для многих других приложений.

Использовать *`imake`* очень просто. Для этого нужно просто зайти в каталог, в котором находится развернутый архив и выполнить скрипт `xmkmf`, который вызовет программу *`imake`*:

```
$ xmkmf -a
imake -DUseInstalled -I/usr/X11R6/lib/X11/config
make Makefiles
```

Если site неправильно установлен, перекомпилируйте и установите *`X11R6`*!

12.3.3. Вариант со скриптом `shell`

Для получения более подробной информации читайте файлы `INSTALL` или `README`. Обычно нужно выполнить файл с названием `install.sh` или `configure.sh`. Инсталляционный скрипт может сам определять все что ему нужно или быть интерактивным и спрашивать у вас разные данные (например, путь).

Если вы не можете определить какой файл вам нужно запустить, вы можете набрать `./` (находясь в *`bash`*) и нажать дважды кнопочку **TAB** (клавиша горизонтальной табуляции). *`bash`* автоматически найдет в текущем каталоге все файлы, которые можно выполнить (в том числе и выполняемые скрипты). Если существует несколько файлов, которые можно выполнить, вы получите список таких файлов и вам останется только выбрать нужный.

В особых случаях нужно установить модули *`perl`*. Установка таких модулей выполняется путем выполнения скрипта, написанного на *`perl`*. Стандартная команда для выполнения такого скрипта выглядит так:

```
$ perl Makefile.PL
```

12.3.4. Другие варианты

Некоторые дистрибутивы свободного программного обеспечения на первых стадиях развития имеют ужасную организацию (мы вас предупреждаем!). Они требуют внесения изменений “ручками” в некоторые конфигурационные файлы. Обычно это файлы `Makefile` (см. раздел Разд. 12.4.1) и `config.h` (эти названия чаще встречаются. но могут быть и другие).

Мы не советуем делать таких манипуляций, если (конечно) вы не понимаете, что вы делаете. Эти действия требуют некоторых знаний, опыта и желания, но не забывайте пословицу : “practice makes perfect” (практика делает совершенным).

12.4. Компиляция

Теперь, когда программное обеспечение правильно сконфигурировано, остается только его откомпилировать. Этот этап прост и не должен вызывать каких либо серьезных проблем.

12.4.1. make

Общество свободного программного обеспечения считает утилиту `make` излюбленным инструментом для компиляции исходных кодов. Это дает следующие преимущества:

- Разработчик экономит время, потому что у него есть возможность эффективно управлять процессом компиляции своего проекта.
- Конечный пользователь может откомпилировать и установить программное обеспечения, введя всего несколько командных строк, даже в том случае, если он ничего не понимает в программировании.

Действия, которые необходимо выполнить для получения откомпилированной версии исходных кодов, обычно хранятся в файле с названием `Makefile` или (реже) в файле `GNUMakefile`. На самом деле, при вызове команды `make` читается этот файл из текущего каталога. Возможно явное указание этого файла для команды `make` с помощью опции `-f`.

12.4.2. Правила

Программа `make` действует согласно системным ***зависимостям (dependencies)***, так для компиляции бинарного файла (" ***цели (target)*** ") требуется прохождение через несколько стадий ("зависимостей (dependencies)"). Например, для того чтобы создать (мнимый) бинарный файл `glloq`, сначала нужно откомпилировать объектные файлы `main.o` и `init.o` (это промежуточные файлы в процессе компиляции), а потом их нужно слинковать в готовый бинарный файл. Эти объектные файлы также являются целями, а зависимостями для них будут файлы исходных текстов.

Этот текст является только маленьким введением, инструкцией по выживанию в сложном мире `make`. Для того, чтобы узнать больше, мы советуем вам отправится на сайт **APRIL** (`http://www.april.org/groupes/doc/`), где размещена более подробная документация о `make`. [6] Для получения исчерпывающей информации обратитесь ко второму изданию **O'Reilly Managing Projects with Make** Andrew Oram и Steve Talbott.

12.4.3. Поехали!

Обычно при использовании программы `make` придерживаются некоторых соглашений. Например, таких как:

- `make` без аргументов производит просто компиляцию и не выполняет установки программы.
- `make install` компилирует программу (правда не всегда), и инсталлирует необходимые файлы в нужное место в файловой системе. Некоторые файлы не всегда корректно инсталлируются (`man`, `info`), но их можно скопировать вручную. Иногда команду `make install` необходимо запускать несколько раз в подкаталогах. Обычно это случается с модулями, разработанными третьими лицами.
- `make clean` удаляет все временные файлы, создающиеся при компиляции, а также, в большинстве случаев, исполняемые файлы.

На первой стадии необходимо скомпилировать программу, для этого нужно набрать: (нереальный пример):

```
$ make
gcc -c glloq.c -o glloq.o
gcc -c init.c -o init.o
gcc -c main.c -o main.o
gcc -lgtk -lgdk -lglib -lXext -lX11 -lm glloq.o init.o main.o -o glloq
```

Замечательно, допустим что все бинарники правильно откомпилировались. Мы готовы для перехода на следующую стадию, на которой будут инсталлироваться файлы пакета (бинарные файлы, файлы с данными и т.д). Смотрите раздел Разд. 12.5.

---

6. На русском языке информацию о `make` можно получить тут: "Эффективное использование GNU Make" (`http://doc.lafox.com.ua/unix/make/gmake/gmake.html`) , "Применение GNU make" (`http://doc.lafox.com.ua/unix/make/make/make.html`) , "GNU Automake" (`http://doc.lafox.com.ua/unix/make/automake/automake-ru_toc.html`)

12.4.4. Объяснения

Если вы достаточно любопытны, чтобы заглянуть в файл `Makefile`, то вы найдете там известные команды (`rm`, `mv`, `cp`, и т.д.) и, кроме того, странные строки вроде этой `$(CFLAGS)`.

Это ***переменные*** , которые обычно расположены в начале файла `Makefile` и связанные с ними значения. Это удобно в случае, когда вам нужно использовать несколько раз одно и тоже значение в нескольких местах.

Например, для того чтобы напечатать строку " `foo` " при выполнении цели `make all`, можно сделать следующее:

```
TEST = foo
all:
        echo $(TEST)
```

Обычно определены следующие переменные:

1. `CC`: Это компилятор. Обычно это `cc`, который присутствует в большинстве свободных систем, также это может быть его аналог `gcc`. Если у вас возникают сомнения, ставьте `gcc`.
2. `LD`: это программа, которая используется на конечной стадии компилирования (см. раздел Разд. 12.1.2.2). По умолчанию это программа *`ld`*.
3. `CFLAGS`: это дополнительные опции компилятору, которые используются компилятором на первой стадии компиляции. Среди них:
   - `-I<path>`: указывает компилятору где искать дополнительные заголовочные файлы (к примеру: `-I/usr/X11R6/include` разрешает компилятору использовать файлы header, расположенные в allows `/usr/X11R6/include`).
   - `-D<symbol>`: определяет дополнительные символы, которые могут быть необходимы для программ, компиляция которых зависит от определения таких символов (к примеру: использовать заголовочный файл `string.h` в случае, если определено `HAVE_STRING_H`).

   Строка для компиляции обычно выглядит следующим образом:

   ```
   $(CC) $(CFLAGS) -c foo.c -o foo.o
   ```

4. `LDFLAGS` (или `LFLAGS`): Этот аргумент используется во время конечной стадии компиляции. Среди них:
   - `-L<path>`: Определяет дополнительные пути поиска библиотек (например: `-L/usr/X11R6/lib`).
   - `-l<library>`: Определяет дополнительные библиотеки, которые будут использоваться в конечном этапе компиляции.

12.4.5. А что если... все это не работает?

Не паникуйте, это может случится с любым. Чаще всего это могут быть следующие ошибки:

1. `glloq.c:16: decl.h: No such file or directory`

   Компилятор не может найти соответствующий заголовочный файл. Вообще-то эту ошибку должна была предвидеть программа конфигурации. Но эта проблема решаема:

   - Проверьте, действительно ли существует данный файл в следующих каталогах: `/usr/include`, `/usr/local/include`, `/usr/X11R6/include` или в каком-нибудь из подкаталогов. Если там нет, поищите по всему диску (с помощью утилит `find` или `locate`) и, если вы все же не можете найти этот файл, проверьте, действительно ли вы установили библиотеку, в которую входит этот заголовочный файл. Примеры использования утилит `find` и `locate` вы можете найти в соответствующих страничках руководства (man find; man locate).
   - Проверьте, действительно ли этот файл доступен для чтения (для проверки этого введите `less <path>/<file>.h`)

- Если это каталог типа `/usr/local/include` или `/usr/X11R6/include`, вам прийдется добавить новый аргумент компилятору. Откройте соответствующий `Makefile` (будьте внимательны: нужный файл находится в каталоге, в котором произошла ошибка компиляции [7]) в вашем любимом текстовом редакторе (*Emacs*, *Vi*, и т.д.). Перейдите на строку, в которой содержится ошибка, и добавьте строку `-I<path>` - где `<path>` это путь к каталогу, в котором вы отыскали недостающий заголовочный файл. Эти опции нужно добавить в конце строки, в которой вызывается компилятор (`gcc`, или `$(CC)`). Если вы не знаете куда добавить эти опции, допишите их в конце строк `CFLAGS=<something>` или `CC=<something>`, которые расположены в начале файла.
- Запустите снова `make` и, если это опять не работает, перепроверьте снова то, что эти опции (смотрите предыдущий пункт) действительно были добавлены и получены компилятором.
- Если это опять не работает, обращаетесь за помощью к вашему местному гуру, или к сообществу свободного программного обеспечения (смотрите раздел Разд. 12.6.2).

2. `glloq.c:28: ‘struct foo’ undeclared (first use this function)`

   Структуры - это такие специальные формы представления данных, которые используются при написании любых программ. Многие из них определяются системой в заголовочных файлах. Обычно эта проблема вызвана тем, что нет какого-то заголовочного файла или он неверен. Правильной процедурой для решения этой проблемы будут следующие действия:

   - Попробуйте найти где определяется эта самая структура (в программе или ее определяет система). Для этого используется утилита *grep*, с помощью которой выясняется определена ли эта структура в каком либо заголовочном файле.

     Станьте root-ом и выполните следующую команду:

     ```
     $ find . -name '*.h'| xargs grep 'struct foo' | less
     ```

     В результате может получиться очень много строк (потому что вы найдете все случаи, когда эта структура используется). Если структура все-таки существует, найдите заголовочный файл, в котором она определяется.

     Определение структуры выглядит так:

     ```
     struct foo {
             <содержимое структуры>
     };
     ```

     Проверьте, соответствует ли это тому, что имеется у вас. Если да, то это значит, что заголовочный файл не включен в `.c` файл, содержащий ошибку. Для устранения этого дефекта есть два способа:

     - в начало `.c` файла, содержащего ошибку, добавьте строку `#include "<filename>.h"`.
     - скопируйте определение этой структуры в начало `.c` файла (на самом деле это не очень правильно, зато обычно помогает).

   - Если файл с определением находится в одном из системных каталогов `/usr/include`, `/usr/X11R6/include`, или `/usr/local/include`) , то сделайте тоже самое, только в этом случае вставьте строку такого вида `#include <<filename>.h>`.
   - Если эта структура не находится, попробуйте выяснить в какой библиотеке (то есть это набор функций, структур и т.д., содержащихся в отдельном пакете) оно должно содержатся (просмотрите файлы `INSTALL` или `README` на предмет того, какие библиотеки использует данная программа и какие версии библиотек необходимы). Если требуемые программой версии библиотек не соответствуют тем, что установлены в вашей системе - вам прийдется установить требуемые версии этих библиотек.
   - Если все же это не работает, уточните, может ли работать эта программа в Linux (некоторые программы могут не работать корректно во всех *UNIX*). Проверьте также, правильно ли переданы все

7. Проанализируйте сообщения об ошибках, выдаваемые программой `make`. Обычно в последних строках содержится название каталога, в котором произошла ошибка (сообщение имеет примерно такой вид `make[1]: Leaving directory ‘/home/benj/Project/foo’`). Выберите сообщение с самым маленьким номером. Для проверки того, что это именно тот каталог, зайдите в него и повторно выполните `make`. При этом вы должны получить точно такую-же ошибку.

опции команде `configure`. Особенно, нет ли каких-то дополнительных опций для вашей конкретной архитектуры.

3. Ошибки синтаксического анализа `(parse error)`

   Такие проблемы довольно сложно решать, так как компилятор может выдать сообщение об ошибке в строке, которая находится ниже той строки, в которой реально присутствует ошибка. Иногда это просто неопределенный тип данных. Если вы встречаете сообщение об ошибке такого вида:

   ```
   main.c:1: parse error before ‘glloq_t
   main.c:1: warning: data definition has no type or storage class
   ```

   это значит, что тип данных `glloq_t` не определен. Для решения этой проблемы нужно предпринять действия, аналогичные тем, что были описаны для решения предыдущей проблемы.

еще может быть `parse error` в старых библиотеках `curses`, если мне не изменяет память.

4. `no space left on device` (на диске кончилось место)

   Эту проблему легко решить: недостаточно места на диске для того, чтобы создать бинарник из исходника. Решение состоит в расчистке места на том разделе, на котором находится каталог инсталляции (удалите временные файлы или исходники, деинсталлируйте программы, которые вы не используете). Если вы развернули его в `/tmp`, а не в `/usr/local/src`, это зря, потому что он напрасно занимает место на разделе `/tmp`. Проверьте, нет ли на диске файлов `core>`. Если найдутся, удаляйте их или заставьте их удалиться, если они принадлежат другому пользователю.

5. `/usr/bin/ld: cannot open -lglloq: No such file or directory`

   Это означает, что программа *`ld`* (используемая `gcc` во время последнего шага компиляции) не может найти библиотеку. Для того чтобы ее включить, *`ld`* ищет файл, чье имя является аргументом типа `-l<library>`. Это файл `lib<library>.so`. Если *`ld`* не находит его, получается ошибка. Для решения проблемы делайте следующее:

   a. Проверьте, есть ли файл на диске с помощью команды `locate`. Графические библиотеки обычно находятся в `/usr/X11R6/lib`. Например:

      ```
      $ locate libglloq
      ```

      Если поиск ничего не принес, вы можете попытаться поискать с помощью команды `find` command (то есть: `find /usr -name libglloq.so*`). Если и это ничего не дало, вам прийдется установить его.

   b. Как только библиотека будет размещена, проверьте ее на доступность для `ld`: файл `/etc/ld.so.conf` определяет, где искать библиотеки. Добавьте каталог-виновник в его конец (возможно, вам прийдется перегрузить компьютер, чтобы изменения вступили в силу). Кроме того, вы можете добавить этот каталог путем изменения содержимого переменной окружения `LD_LIBRARY_PATH`. Например, если каталог такой `/usr/X11R6/lib`, напишите:

      ```
      export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/X11R6/lib
      ```

      (если ваша shell это *`bash`*).

   c. Если до сих пор не работает, убедитесь что формат библиотеки это выполняемый файл (или ELF) командой `file`. Если он является символической ссылкой, проверьте что ссылка правильная и не указывает на несуществующий файл (например, так `nm libglloq.so`). Права файла тоже могут быть неверными (если вы используете аккаунт, отличный от `root`, и если библиотека защищена от чтения, например).

6. `glloq.c(.text+0x34): undefined reference to ‘glloq_init’`

Это проблема с символом, которая не была решена во время последнего шага компиляции. Обычно это проблема библиотеки. Может возникать по нескольким причинам:

- первое, что необходимо выяснить, это **предполагалось** ли наличие символа в библиотеке. Например, если символ в начале это `gtk`, он принадлежит библиотеке `gtk`. Если имя библиотеки легко определимо, (`frobnicate_foobar`), вы можете вывести список символов библиотеки командой `nm`. Например,

  ```
  $ nm libglloq.so
  0000000000109df0 d glloq_message_func
  000000000010a984 b glloq_msg
  0000000000008a58 t glloq_nearest_pow
  0000000000109dd8 d glloq_free_list
  0000000000109cf8 d glloq_mem_chunk
  ```

  Добавив параметр `-o` к `nm`, вы получите вывод имени библиотеки в каждой строке, что облегчит поиск. Давайте предположим, что мы ищем символ `bulgroz_max`, сырое решение поиска выглядит так:

  ```
  $ nm /usr/lib/lib*.so | grep bulgroz_max
  $ nm /usr/X11R6/lib/lib*.so | grep bulgroz_max
  $ nm /usr/local/lib/lib*.so | grep bulgroz_max
  /usr/local/lib/libfrobnicate.so:000000000004d848 T bulgroz_max
  ```

  Замечательно! Символ `bulgroz_max` определен в библиотеке `frobnicate` (большая буква `T` стоит перед ее именем). Теперь вам только нужно добавить строку `-lfrobnicate` в строку компиляции, отредактировав файл `Makefile` : добавьте ее в конец строки, где определены `LDFLAGS` или `LFGLAGS` (или, на худой конец, `CC`) , или в строку, соответствующую созданию конечного бинарного файла.

- компиляция производится с версией библиотеки, которая не подходит для данного программного обеспечения. Читайте `README` или `INSTALL` чтобы узнать, какая версия должна использоваться.
- не все объектные файлы поставки были корректно слинкованы. Файл, в котором определена функция, отсутствует. Напишите `nm -o *.o` чтобы узнать что это за файл и добавить соответствующий файл `.o` в строку компиляции, если его не хватает.
- проблемная функция или переменная может не существовать. Попробуйте ее удалить: отредактируйте проблемный файл (его имя указано в сообщении об ошибке). Это довольно отчаянное решение, которое может привести к неуправляемому выполнению программы (***segfault*** при запуске, и т.д.).

7. `Segmentation fault (core dumped)`

   Иногда компилятор падает сразу и выдает такое сообщение об ошибке. У меня нет иного совета на этот счет, кроме как поставить более свежую версию компилятора.

8. no space on `/tmp`

   Компиляции необходимо временное рабочее пространство во время ее различных шагов; если ей не хватит места на диске, она упадет. Поэтому вы можете почистить свои разделы диска, но будьте осторожны с некоторыми программами, которые выполняются (`X`, сервер, каналы и т.д.), так как они могут упасть, если некоторые файлы будут удалены. Вы должны знать что делаете! Если раздел `/tmp` является частью раздела, который содержит не только его (например, root), поищите и, по возможности, удалите несколько `core` файлов.

9. `make/configure` в бесконечном рекурсивном цикле

   Часто это проблема со временем в вашей системе. Действительно, `make` нужно знать дату в компьютере и дату файлов для проверки. Она сравнивает даты и использует результат для того, чтобы определить насколько цель отличается по времени создания от зависимостей.

   Проблемы с датой могут заставить `make` бесконечно пересоздавать саму себя (или формировать снова и снова поддерево в бесконечном цикле рекурсии). В таком случае проблема решается обычно

использованием команды `touch` (которая здесь используется для установки файлам по запросу текущего времени).

Например:

```
$ touch *
```

или так (грубо, но эффективно):

```
$ find . | xargs touch
```

12.5. Инсталляция

12.5.1. С помощью `make`

Теперь, когда все откомпилировано, нужно скопировать полученные файлы в подходящее место (обычно это один из подкаталогов `/usr/local`).

Это обычно является задачей `make`. Специальная цель это цель `install`. Таким образом, использование `make install` приводит к установке нужных файлов.

Процедура обычно описана в файле `INSTALL` или `README`. Но иногда разработчик забывает это описать. В таком случае вам прийдется устанавливать все самостоятельно.

Для этого скопируйте:

- Исполняемые файлы (программы) в каталог `/usr/local/bin`
- Библиотеки (файлы `lib*.so` files) в каталог `/usr/local/lib`
- Файлы заголовков (headers) (файлы `*.h`) в каталог `/usr/local/include` (будьте осторожны и не удаляйте оригиналы).
- Файлы с данными обычно отправляются в `/usr/local/share`. Если вам неизвестна процедура инсталляции, вы можете попытаться запустить программы, не копируя файлы с данными, а положить их в нужное место только тогда, когда программа спросит вас о них (в сообщении об ошибке, например `Cannot open /usr/local/share/glloq/data.db`).
- Ситуация с документацией несколько отличается:
  - Файлы `man` размещают обычно в подкаталогах `/usr/local/man`. Эти файлы обычно в формате `troff` (или `groff`) и имеют в качестве расширения число. Они называются по имени команды (например, `echo.1`). Если число это `n`, скопируйте файл в `/usr/local/man/man<n>`.
  - Файлы `info` размещаются в каталоге `/usr/info` или `/usr/local/info`

Вот все и завершилось! Поздравления! Теперь вы готовы компилировать хоть всю систему!

12.5.2. Проблемы

Если вы только что проинсталлировали свободное программное обеспечение, например *`GNU tar`*, и если запускается другая программа вместо ожидаемой, или она не работает так, как работала при тестировании в каталоге `src`, то это проблема с переменной окружения `PATH`, которая ищет программы не в том каталоге, где вы не инсталлировали свою новую программу. Проверьте это, выполнив `type -a <program>`.

Решением может быть разместить каталог с инсталляцией выше `PATH` и/или удалить/переименовать файлы, которые выполнялись, когда их не просили, и/или переименовать ваши новые программы (в нашем примере в `gtar`), чтобы недоразумений больше не возникало.

Можно также сделать alias, если shell это позволяет (например, сказать, что `tar` обозначает `/usr/local/bin/gtar`).

12.6. Поддержка

12.6.1. Документация

Вот несколько источников документации:

- *HOWTOs*, короткие документы на определенные темы (обычно далеки от того, что нам надо, но весьма полезны). Поищите на своем диске в `/usr/share/doc/HOWTO` (не всегда там, могут быть и в другом месте ; это можно сделать командой `locate HOWTO`),
- Страницы справочных руководств (manual pages). Наберите `man <command>`, чтобы получить документацию по команде `<command>`,
- Специализированная литература. Некоторые большие издания начали публикацию книг на тему свободных систем (особенно по *GNU/Linux*). Это часто полезно тогда, когда вы еще новичок и если вы не совсем понимаете всех тонкостей данной документации.

12.6.2. Техническая поддержка

Если вы купили "официальный" дистрибутив **Mandrake Linux**, вы можете спросить работника технической поддержки по поводу информации о вашей системе.

Вы также можете положиться на помощь сообщества свободного ПО:

- ***группы новостей*** (на *Usenet*) comp.os.linux.* (`news:comp.os.linux.*`) отвечает на все вопросы относительно *GNU/Linux*. Группы новостей такого типа `comp.os.bsd.*` имеют дело с системами BSD. Существуют и другие группы новостей, имеющие дело с другими системами *UNIX*. Помните, что сначала нужно почитать их, а потом уже писать туда.
- Несколько ассоциаций или групп энтузиастов в сообществе свободного ПО предлагают добровольную помощь. Наилучший способ найти наиболее подходящую для вас - это посматривать списки на специализированных web-сайтах, или читать некоторое время нужные группы новостей.
- На некоторых ***каналах IRC*** вы можете получить в реальном режиме времени (но не видя собеседника) помощь от ***гуру***. К примеру, смотрите канал `#linux` в большинстве сетей IRC, или `#linuxhelp` в *IRCNET*.
- И как последнее средство, спросите разработчика ПО (если он указал свое имя и ***email*** адрес в файле дистрибутива), если вам показалось что вы нашли баг (который мог возникнуть только на вашей архитектуре, ведь в конечном итоге свободный софт должен быть портируемым).

12.6.3. Как найти свободное программное обеспечение

Для поиска свободного ПО, вам могут пригодиться множество ссылок:

- Огромный FTP сайт sunsite.unc.edu (`sunsite.unc.edu`) или одно из его зеркал
- на следующих web-сайтах собираются каталоги свободного ПО, которые могут быть использованы на платформах *UNIX* (но на них можно найти и частное ПО):
  - FreshMeat (`http://www.freshmeat.net/`) возможно самый полный сайт ,
  - Linux France (`http://www.linux-france.org/`) содержит массу ссылок на софт, работающий с *GNU/Linux*. Большинство из них также работает и под другими свободными платформами *UNIX*,
  - GNU Software (`http://www.gnu.org/software/`) , на котором имеется исчерпывающий список всего программного обеспечения GNU. Все оно, конечно же, является свободным и имеет лицензию GPL.
  - SourceForge.net (`http://sourceforge.net/`) самый большой в мире сайт по разработке программного обеспечения Open Source, на котором имеется самый большой архив Open Source кодов и приложений, доступных в сети Интернет.

- вы можете также произвести поиск на поисковых системах, таких как Google/ (`http://www.google.com/`) и Lycos/ (`http://ftpsearch.lycos.com/`) и делать запрос типа : `+<software> +download` или `"download software"`.

12.7. Подтверждения и благодарности

- Корректура и неприятные комментарии (и в алфавитном порядке): SIbastien Blondeel, Mathieu Bois, Xavier Renaut и Kamel Sehil.
- ***Бета-тестирование***: Laurent Bassaler
- Английский перевод: Fanny Drieu Английская редакция: Hoyt Duff

Глава 13. Сборка и Установка Новых Ядер

Точно также, как монтирование файловых систем и сборка программ из исходных кодов, компиляция ядра - это тема, которая несомненно вызывает затруднения у начинающих. На самом деле, вам врядли прийдется перекомпилировать ядро, потому что скомпилированное ядро, которое поставляется с **Mandrake Linux**, включает в себя поддержку огромного числа устройств (фактически, гораздо больше, чем вы когда-либо будете использовать), а также набор заплаток (patches) и так далее. Но, тем не менее...

Возможно, вы захотите это сделать только из тех соображений, чтобы узнать "как это делается". Или, например для того, чтобы сделать работу вашего PC (или кофеварки ⌣) немного быстрее (действительно немного). Причиной может послужить также желание поставить новое эксперементальное ядро или, например, вам захочется добавить или убрать какие-то опции в скомпилированном ядре. Цель этой главы состоит в том, чтобы ваша кофеварка работала после компиляции ядра.

Существуют и другие причины для перекомпиляции ядра. Например, вы узнали что у ядра, которое вы используете, обнаружена ошибка (***bug***) в безопастности, и для того чтобы ее поправить, необходимо установить более новую версию ядра, или например, новая версия ядра включает поддержку устройства, которое вам необходимо. Конечно, в этом случае вы можете подождать готовых, скомпилированных обновлений, но более быстрым решением будет обновление исходных кодов ядра и компиляция.

Независимо от того, что вы собрались ковырять - обязательно запаситесь кофе.

13.1. Где Найти Исходный Код Ядра

В основном, исходники можно получить в двух местах:

1. **Официальное ядро Linux Mandrake.** В каталоге `SRPMS` в любом из `Cooker` mirrors (`http://www.MandrakeLinux.com/en/cookerdevel.php3`), вы можете найти следующие пакеты:

   kernel-2.4.??.?mdk-?-?mdk.src.rpm

   > Исходные коды ядра для компиляции ядра, используемого в дистрибутиве. Оно сильно изменено для внесения дополнительных функциональных возможностей.

   kernel-linus2.4-2.4.??-?mdk.src.rpm

   > Готовое ядро в таком виде, как оно было опубликовано разработчиками ядра *GNU/Linux*.

   Если вы избрали этот путь (что мы рекомендуем), тогда загрузите себе исходники в RPM, инсталлируйте их (как `root`) и переходите сюда Разд. 13.3.

2. **Официальное хранилище Linux Kernel.** Главное хранилище исходников ядра расположено тут ftp.kernel.org (`ftp://ftp.kernel.org`), но существует множество зеркал, которые носят имена ftp.xx.kernel.org (`ftp://ftp.xx.kernel.org`), где xx (`xx`) - код страны в формате ISO. После официального объявления о пригодности ядра, подождите хотя-бы пару часов, пока обновятся зеркала.

   На всех этих FTP серверах исходники ядра расположены в каталоге `/pub/linux/kernel`. Далее пройдите в каталог с названием серии интересующего вас ядра : наверняка это будет `v2.4`. Ничего не мешает вам использовать версию из серии 2.5, но помните - это экспериментальные ядра. Файлы, содержащие исходники ядра, называются `linux-<kernel.version>.tar.bz2`, например `linux-2.4.8.tar.bz2`.

   Также для того, чтобы обновить ядро частично, вы можете к исходникам ядра прикладывать патчи (patches): например если у вас есть исходники ядра 2.4.8, а вы хотите обновить его до версии 2.4.10, вам не обязательно качать полностью все исходники 2.4.10, вы можете выкачать только патчи (***patches***) `patch-2.4.9.bz2` и `patch-2.4.10.bz2`. Это хорошая идея, так как современные ядра имеют размер более 23MB.

13.2. Распаковка Исходников, Патченье Ядра (при необходимости)

Исходники ядра должны размещаться в `/usr/src`. Поэтому переместитесь в этот каталог и распакуйте исходники следующим образом:

```
$ cd /usr/src
$ mv linux linux.old
$ tar xjf /path/to/linux-2.4.8.tar.bz2
```

Команда `mv linux linux.old` необходима: в случае, если у вас уже есть установленные исходники ядра другой версии. Эта команда позволит вам не запортить старую версию при распаковке новой. Когда архив будет распакован, вы получите каталог `linux` с исходниками нового ядра.

Теперь патчи. Мы настолько самонадеянны, что хотим пропатчить (***to patch***) ядро с версии 2.4.8 до версии 2.4.10 и загрузили все необходимые для этого патчи: заходите в свежесозданный каталог `linux`, и примените эти патчи (apply the patches):

```
$ cd linux
$ bzcat /path/to/patch-2.4.9.bz2 | patch -p1
$ bzcat /path/to/patch-2.4.10.bz2 | patch -p1
$ cd ..
```

В общем случае, для перехода от версии 2.4.x к версии 2.4.y требуется последовательно применить все патчи с номерами 2.4.x+1, 2.4.x+2, ..., 2.4.y. Для возвращения от версии 2.4.y к версии 2.4.x, нужно провести те же самые действия, но применять патчи в обратной последовательности, а также использовать опцию `-R` в команде `patch` (`R` - сокращение от *Reverse*). Так для того, чтобы вернуться обратно от ядра 2.4.10 к ядру 2.4.8 нужно сделать следующее:

```
$ bzcat /path/to/patch-2.4.10.bz2 | patch -p1 -R
$ bzcat /path/to/patch-2.4.9.bz2 | patch -p1 -R
```

Если вы хотите проверить корректность применения патча перед тем как действительно его приложить, добавьте опцию `--dry-try` к команде `patch`.

Далее, для большей ясности (и для того чтобы вы знали, где находитесь), переименуйте `linux` таким образом, чтобы в имени содержалась версия ядра и создайте символическую ссылку на этот каталог:

```
$ mv linux linux-2.4.10
$ ln -s linux-2.4.10 linux
```

Теперь самое время заняться конфигурированием. Для этого перейдите в каталог с исходниками.

```
$ cd linux
```

13.3. Конфигурирование Ядра

Для начала зайдите в каталог `/usr/src/linux`.

Для начала маленький трюк: если хотите, то вы можете изменить версию вашего ядра. Версия ядра определяется в первых четырех строках файла `Makefile`:

```
$ head -4 Makefile
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 10
EXTRAVERSION =
```

Ниже в файле `Makefile` вы можете увидеть, что версия ядра формируется следующим образом:

```
KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
```

Все, что нужно сделать, это изменить одно из этих полей, чтобы изменить вашу версию. Но предпочтительно менять только поле EXTRAVERSION. Скажем для примера, вы поменяли его на `-foo`. Теперь новая версия вашего ядра 2.4.10-foo. Изменяйте без колебаний это поле при каждой перекомпиляции вашего ядра с новыми опциями, Это даст вам возможность проверять новые опции без потери результатов предыдущих компиляций.

Теперь начнем конфигурацию. Вам необходимо выбрать между:

- `make xconfig` для того чтобы использовать графический интерфейс;
- `make menuconfig` для того чтобы использовать интерфейс, основанный на `ncurses` (текстовый с меню);
- `make config` для использования рудиментального (очень простого, текстового) интерфейса, строка за строкой, секция за секцией;
- `make oldconfig` тоже самое что описано выше, но основано на предыдущей конфигурации. Смотрите Разд. 13.4.

Вы можете конфигурировать последовательно секцию за секцией, также вы можете пропускать секции и переходить к интересующим вас секциям в случае, когда вы используете `menuconfig` или `xconfig`. Существуют следующие опции: **y** для Yes (функциональная возможность жестко вкомпилированная в ядро), **m** для Module (функциональная возможность, скомпилированная как модуль) или **n** для No (исключить эту возможность из ядра).

В обеих вариантах `make xconfig` и `make menuconfig` присутствует иерархическое разделение опций ядра на группы. Например, `Processor family` находится в `Processor type and features`.

В `xconfig`, кнопочка Main Menu используется для возвращения назад в главное меню во время нахождения внутри иерархической группы опций; соответственно кнопочка Next служит для перехода к следующей группе опций; а кнопочка Prev для возвращения в предыдущую группу. В `menuconfig`, используйте кнопку `Enter` для выбора секции, и для изменения состояния опции используйте клавиши **y**, **m** или **n**, также можно сделать выбор, нажимая клавишу `Enter`. Exit служит для выхода из секции или вообще из конфигурации в случае, когда вы находитесь в главном меню. Также есть кнопка Help.

Мы не собираемся перечислять тут все опции ядра, так как их насчитывается много сотен. Кроме того, если вы читаете эту главу, то наверное, так или иначе знаете что вам необходимо. Так что вы брошены на самостоятельное изучение конфигурации ядра и самостоятельную установку опций по вашему усмотрению. Однако мы приведем некоторые советы, которые помогут вам избежать сборки неработоспособного ядра:

1. если вы не используете для начальной загрузки виртуальный диск (initial ramdisk), **никогда** не компилируйте драйвера, необходимые для монтирования вашей корневой файловой системы (драйвера оборудования и драйвера файловых систем) как модули! И, если вы используете initial ramdisk, установите `Y` для поддержки ext2FS support, так как эта файловая система используется для виртуальных дисков. Вы также должны включить поддержку `initrd`;
2. если в вашей системе присутствуют сетевые карты, компилируйте их драйвера как модули. В результате вы сможете определять последовательность карт путем помещения псевдонимов (aliases) в файл `/etc/modules.conf`. В случае, когда вы вкомпилируете драйверы в ядро - они будут загружаться в том порядке, как они были влинкованы в ядро, при этом вы не сможете изменить это порядок по собственному усмотрению;
3. и наконец: если вы не знаете на что влияет данная конкретная опция - читайте помощь! Если текст помощи так и не внушил вам уверенности - оставьте эту опцию так как она была. (для получения помощи в `config` и `oldconfig` нажмите кнопочку **?**).

Кроме того вы можете получить дополнительную информацию в файле `/usr/src/linux/Documentation/Configure.help`, откуда берется текст справки для всех опций во время настройки. В заголовке этого файла вы также можете найти ссылки на переводы.

И вот наконец! Конфигурирование закончено. Сохраните свою конфигурацию и выходите.

13.4. Сохранение Файлов Конфигурации Ядра для Повторного Использования

Настройки ядра хранятся в файле `/usr/src/linux/.config`. Очень рекомендуется делать резервную копию этого файла, например в каталоге `/root`, для того чтобы иметь возможность использовать в последствии, причем сохранять с разными именами.

Один из возможных вариантов именования файлов конфигурации - по версии ядра. Так, при изменении версии ядра как было показано в Разд. 13.3, вы делаете следующее:

```
$ cp .config /root/config-2.4.10-foo
```

Если вы решитесь (для примера) обновить ядро до версии 2.4.12, вы можете использовать повторно этот файл, а различия в конфигурации этих двух ядер будет очень маленькими. Просто используйте резервную копию:

```
$ cp /root/config-2.4.10-foo .config
```

Но копирование этого файла обратно все же не подразумевает того, что ядро готово к компиляции. Вы снова должны вызвать команду `make menuconfig` (или другую команду, которую вы хотите использовать), для того чтобы были созданы (или изменены) некоторые файлы, необходимые для компиляции.

Однако, во время механической работы по повторному прохождению всех меню, вы можете не заметить какую-то новую интересную опцию. Чтобы избежать такой неприятности, мы советуем использовать `make oldconfig`. В этом имеются два преимущества:

1. это быстро;
2. если в файле конфигурации ядра появится новая опция, которой не было ранее, то при выборе этого способа процесс остановится на ней и будет ждать вашего выбора..

После того, как вы скопируете файл `.config` в домашний каталог `root`, как было описано ранее, вызовите командочку `make mrproper`. В этом случае будет гарантия того, что от старой конфигурации ядра не осталось ничего лишнего и вы получите чистенькое ядро.

Далее, настал черед компиляции.

13.5. Компиляция Ядра и Модулей, Установка Бестии

Для начала маленькое примечание: Если вы перекомпилируете ту же самую версию ядра которая уже скомпилирована в вашей системе, не забудьте сначала удалить старые модули. К примеру, если вы перекомпилируете ядро 2.4.10, вы должны удалить каталог `/lib/modules/2.4.10`.

Скомпилировать ядро и модули, потом проинсталлировать их можно с помощью следующих команд:

```
make dep
make clean bzImage modules
make modules_install install
```

Небольшой словарик: `dep`, `bzImage`, и т.д., также как и `oldconfig` и прочие, которые были использованы выше называются **целями** (targets). Если вы укажете несколько целей для программы `make`, как было сделано раньше, то эти цели будут выполнены в том порядке, в котором они были перечислены. Но, в случае сбоя любой из целей, программа `make` не будет продолжать сборку последующих целей [1].

Давайте посмотрим что происходит при создании различных целей:

- `dep`: вычисляет зависимости между различными файлами исходников. Это необходимо делать при любом изменении настроек ядра, в противном случае некоторые файлы не будут изменены надлежащим образом и компиляция не сможет пройти успешно;

1. В случае, если произошла ошибка, это значит что ошибка содержится в ядре... Если дело обстоит именно таким образом, пожалуйста сообщите о этом!

- `bzImage`: собирает ядро. Обратите внимание, что эта цель верна только для процессоров *Intel*. Для других процессоров существует `zImage`. Различия между `bzImage` и `zImage` заключается в том, что в случае `zImage` генерируется ядро, которое может быть значительно больше. Эта цель также создает файл `System.map` для этого ядра. То, как можно использовать этот файл, мы рассмотрим далее;
- `modules`: эта цель генерирует модули для ядра, которое вы только что собрали. Если вы собирали ядро без модулей - то по понятным причинам, создавать эту цель не нужно;
- `modules_install`: инсталлирует модули. По умолчанию модули будут установлены в каталог `/lib/modules/<kernel-version>`. Эта цель также вычисляет зависимости модулей (в отличии от 2.2.x);
- `install`: это последняя цель, которая производит копирование ядра и модулей в надлежащие места и изменяет конфигурацию загрузчика системы таким образом, чтобы в процессе загрузки было доступно новое ядро. Не используйте эту цель, если вы предпочитаете ручную настройку, как это описано в Разд. 13.6.

На данный момент у нас все откомпилировано, корректно установлено и готово к проверке! Просто перегрузите вашу машину и выберите новое меню в меню загрузчика. Обратите внимание на то, что старое ядро осталось доступным и может быть использовано в случае, если у вас возникнут проблемы с новым ядром. Кстати вы самостоятельно можете менять меню загрузки так, как вам нравится. Об это мы напишем в следующем разделе.

13.6. Инсталяция Нового Ядра Вручную

Ядро размещается в `arch/i386/boot/bzImage` (или в `zImage` если вы использовали `make zImage`). Стандартным каталогом для установки ядра является `/boot`. Вам также нужно скопировать файл `System.map` чтобы обеспечить правильную работу некоторых программ (например `top`). Довольно полезно называть эти файлы по версии ядра. Предположим, что версия вашего ядра 2.4.10-foo. Вам следует сделать следующее:

```
$ cp arch/i386/boot/bzImage /boot/vmlinux-2.4.10-foo
$ cp System.map /boot/System.map-2.4.10-foo
```

Теперь вам необходимо оповестить загрузчик системы о появлении нового ядра. Существует два варианта: вы используете *grub* или *LILO*. Примечание: **Mandrake Linux** по умолчанию сконфигурирован для использования *LILO*.

13.6.1. Перенастройка Grub

Очевидно, что важно сохранить возможность загрузки вашего старого ядра. Самый простой способ изменить настройки *grub* - это использовать *drakboot* (смотрите главу Изменение Загрузочной Конфигурации в книге *Стартовое Руководство Пользователя*). Альтернативой может служить изменение файла следующим образом.

Вам нужно изменить файл `/boot/grub/menu.lst`. Примерно так обычно выглядит файл `menu.lst` после инсталляции дистрибутива **Mandrake Linux** и до того, как вы что-то в нем поменяли:

```
timeout 5
color black/cyan yellow/cyan
i18n (hd0,4)/boot/grub/messages
keytable (hd0,4)/boot/fr-latin1.klt
default 0

title linux
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5

title failsafe
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5  failsafe

title Windows
root (hd0,0)
makeactive
chainloader  +1

title floppy
```

```
root (fd0)
chainloader +1
```

Этот файл содержит две части: заголовок с общими настройками (первые пять строк), и образы, которые содержат информацию о различных ядрах *GNU/Linux* или о других операционных системах (OS) `timeout 5` определяет задержку (в секундах) перед тем, как *grub* начнет автоматическую загрузку образа по умолчанию (он устанавливается строкой `default 0` в общих настройках, то есть будет загружен первый образ в данном списке). Директива `keytable` определяет где найти раскладку (keymap) для вашей клавиатуры. В данном примере, это французская раскладка. Для установки русской раскладки нужно поставить какую-либо из русских, например `keytable (hd0,4)/boot/ru4.klt`. В случае, когда неопределена ни одна из раскладок, будет использована plain `QWERTY`. Надпись `hd(x,y)` обозначает ссылку на номер диска `x` (как в *BIOS*) и номер раздела на нем `y`.

Затем идут различные образы. В приведенном примере определено четыре образа: `linux`, `failsafe`, `windows`, и `floppy`.

- Секция `linux` рассказывает *grub* про ядро, которое должно быть загружено (`kernel hd(0,4)/boot/vmlinuz`), тут же присутствуют опции, указывающие где найти ядро. В данном случае, `root=/dev/hda5` корень файловой системы расположен на `/dev/hda5`. Фактически `/dev/hda5` для *grub* является эквивалентом `hd(0,4)`, но ничего не мешает ядру находится на разделе, отличном от корневого;
- Секция `failsafe` похожа на предыдущую, за исключением того, что ядру передается параметр (`failsafe`) который заставляет ядро грузиться в "однопользовательском" режиме (single or rescue mode);
- секция `windows` просто говорит *grub* загрузиться с первого сектора первого раздела, где содержится загрузочный сектор *Windows*;
- секция `floppy` просто грузит систему с дискеты, находящейся в первом дисководе (конечно, если она там есть). Это может быть загрузочный диск *Windows* или ядро *GNU/Linux*, расположенное на дискете;

В зависимости от уровня безопасности, который вы используете в вашей системе, в этом файле могут отсутствовать некоторые записи, описанные здесь.

Вернемся к делу. На этом этапе нам нужно добавить новую секцию в конфигурационный файл *grub*, описывающую загрузку нового ядра. В этом примере мы разместим новую секцию впереди других, но нам ничего не мешает поместить эту секцию где-нибудь в другом месте:

```
title foo
kernel (hd0,4)/boot/vmlinux-2.4.10-foo root=/dev/hda5
```

Не забудьте адаптировать этот файл к вашей конфигурации системы! В данном примере корневая файловая система *GNU/Linux* находится тут: `/dev/hda5`, Но в вашей системе это может быть совсем другое место.

Вот и все. В отличии от программы *LILO* (которая будет рассмотрена ниже), больше ничего делать не надо. Просто перезагрузите свой компьютер и новая запись, которую вы только что определили, уже появится. Теперь выберите новое ядро из меню и новое ядро должно загрузится.

Если вы скомпилировали ядро с поддержкой видеобуфера (framebuffer), вы вероятно захотите его использовать: в этом случае вам нужно добавить параметр ядру, указывающий разрешение в котором должна стартовать система. Список режимов можно вычитать в файле `/usr/src/linux/Documentation/fb/vesafb.txt` (это верно только если включен VESA framebuffer! Иначе обратитесь к соответствующему файлу). Для режима 800x600 32бита [2], соответствует режим с номером `0x315`, поэтому вам нужно добавить к параметрам ядра следующую директиву:

```
vga=0x315
```

2. 8 бит поддерживает 2^8 цветов, то есть. 256; 16 бит поддерживает 2^{16} цветов, то есть. 64k, или. 65536; 24 бита как и 32 кодируется при помощи 24 бит, то есть 2^{24} возможных цветов, другими словами 16M, или немного больше чем 16 миллионов цветов.

и ваша запись преобразится в:

```
title foo
kernel (hd0,4)/boot/vmlinux-2.4.10-foo root=/dev/hda5 vga=0x315
```

Для получения более подробной информации обратитесь к справочной системе относительно *grub* (`info grub`).

13.6.2. Перенастройка LILO

Наиболее простой способ для перенастройки *LILO* это использование *drakboot* (смотрите главу Изменение Загрузочной Конфигурации в книге *Стартовое Руководство Пользователя*). Другой способ - вы можете отредактировать файл конфигурации вручную следующим образом.

Конфигурационным файлом для *LILO* является `/etc/lilo.conf`. Мы приводим типичный файл конфигурации `lilo.conf`:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
vga=normal
default=linux
keytable=/boot/fr-latin1.klt
lba32
prompt
timeout=50
message=/boot/message
image=/boot/vmlinuz-2.4.8-17mdk
        label=linux
        root=/dev/hda1
        read-only
other=/dev/hda2
        label=dos
        table=/dev/hda
```

Файл `lilo.conf` состоит из основного раздела и дополнительных разделов для каждой из операционных систем. В приведенном выше примере основной раздел содержит следующие директивы:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
vga=normal
default=linux
keytable=/boot/fr-latin1.klt
lba32
prompt
timeout=50
message=/boot/message
```

Директива `boot=` сообщает *LILO* где размещается загрузочный сектор; в данном случае в MBR (*Master Boot Record*) на первом жестком диске IDE. Если вы хотите чтобы *LILO* грузилось с дискеты, просто замените `/dev/hda` на `/dev/fd0`. Директива `prompt` заставляет *LILO* показывать меню при старте. Из установки времени задержки (пауза) следует, что *LILO* запустит образ по умолчанию через 5 секунд (`timeout=50`). Если вы уберете директиву `timeout`, то *LILO* будет ждать до бесконечности пока вы что нибудь не выберете.

Теперь идет раздел `linux`:

```
image=/boot/vmlinuz-2.4.8-17mdk
        label=linux
        root=/dev/hda1
```

```
        read-only
```

Секция для загрузки ядра *GNU/Linux* всегда начинается с директивы `image=`, в которой указывается полный, правильный путь к ядру. Любой раздел содержит директиву с уникальным идентификатором `label=`, в данном случае это `linux`. Директива `root=` указывает *LILO* на каком разделе находится корневая файловая система для этого ядра. Разумеется, что в вашей конфигурации корневая файловая система может находится в другом месте... Директива `read-only` объясняет *LILO*, что в процессе загрузки корневую файловую систему нужно примонтировать в режиме только для чтения (read-only): если эта директива отсутствует - вы получите предупреждение.

Рассмотрим раздел *Windows*:

```
other=/dev/hda2
        label=dos
        table=/dev/hda
```

Если раздел начинается с директивы `other=`, то фактически *LILO* может запускать любую операционную систему, отличную от *GNU/Linux*: аргументом этой директивы служит расположение загрузчика этой системы, в данном случае это загрузчик *Windows*. Для того чтобы найти начало раздела, содержащего загружаемую операционную систему *GNU/Linux*, также нужно знать где находится таблица разделов диска. Это указывается с помощью директивы `table=`. Как и в случае с разделом `linux`, директива `label=` содержит уникальный идентификатор раздела.

Теперь пришло время добавить раздел для вашего нового ядра. Вы можете разместить этот раздел где угодно, ниже основного раздела, но только не внутри другого раздела. Вообщем, для добавления в загрузчик нового ядра нужно вписать в файл `/etc/lilo.conf` что-то наподобие этого:

```
image=/boot/vmlinux-2.4.10-foo
        label=foo
        root=/dev/hda1
        read-only
```

Не забудьте адаптировать этот пример для вашей конфигурации системы! Мы преднамеренно привели пример для ситуации отличной от той, что была рассмотрена в примере по настройке *grub*

Если вы скомпилировали ваше ядро с поддержкой видеобуфера (framebuffer), то нужно указать опции ядру аналогично, как это было показано для *grub*. Единственным отличием является то, что каждая опция ядра занимает отдельную строку:

```
vga=0x315
```

Приводим вид файла `/etc/lilo.conf` после внесения изменений, с некоторыми комментариями для красоты (все строки, начинающиеся с `#`), будут игнорированы программой *LILO*):

```
#
# Общий раздел
#
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
# во  время запуска, мы хотим normal VGA. Framebuffer переключит разрешение
# в случае если будет использовано это:
vga=normal

# сообщение, которое будет показано при старте системы...
message=/boot/message

# Что будет грузиться по умолчанию.
# Внесем сюда наше свежеоткомпилированное ядро:
default=foo
# показать вопрос о том, что грузить...
```

```
prompt
# ... ждать пять секунд
timeout=50
#
# Ваше новое ядро: будет грузится по умолчанию
#
image=/boot/vmlinux-2.4.10-foo
        label=foo
        root=/dev/hda1
        read-only

#если есть VESA framebuffer - используйте:
        vga=0x315
#
# "родное" ядро
#
image=/boot/vmlinuz-2.4.8-17mdk
        label=linux
        root=/dev/hda1
        read-only
#
# Windows ....
#
other=/dev/hda2
        label=dos
        table=/dev/hda
```

Примерно так должен выглядеть файл `/etc/lilo.conf`... но не забудьте адаптировать его к конфигурации своей системы.

Теперь, когда файл был изменен соответствующим образом, вы должны внести изменения в загрузочный сектор (этого не надо было делать в случае с *grub*):

```
$ lilo
Added foo *
Added linux
Added dos
$
```

Таким образом, вы можете скомпилировать сколько угодно ядер, и добавлять столько вариантов загрузки, сколько необходимо. Все, что теперь нужно - это перезагрузить машину и опробовать новое ядро.

Приложение A. Универсальная Общественная Лицензия GNU (GPL)

Следующий текст является лицензией GPL , которая применяется к большинству программ, поставляемых в дистрибутивах **Mandrake Linux**.

Версия 2, Июнь 1991 Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Этот документ можно копировать, а также распространять его дословные копии, однако вносить в него изменения запрещено.

A.1. Преамбула

Лицензии на большую часть программного обеспечения (ПО) составлены так, чтобы лишить вас свободы совместно использовать и изменять его. В противоположность этому, предназначение Универсальной Общественной Лицензии GNU состоит в том, чтобы гарантировать вашу свободу совместно использовать и изменять свободное ПО, т.е. обеспечить свободу ПО для всех его пользователей. Данная Универсальная Общественная Лицензия применима к большей части ПО Фонда Свободного ПО и ко всем другим программам, чьи авторы принимают на себя обязательство ее использовать. (Для некоторых программ Фонда Свободного ПО вместо нее применяется Универсальная Общественная Лицензия GNU для библиотек.) Вы тоже можете применить ее к своим программам.

Когда мы говорим о свободном ПО, мы имеем в виду свободу, а не бесплатность. Наши Универсальные Общественные Лицензии разрабатывались для того, чтобы гарантировать, что вы пользуетесь свободой распространять копии свободного ПО (и при желании получать за это вознаграждение); что вы получаете исходный код или можете получить его, если захотите; что вы можете изменять ПО или использовать его части в новых свободных программах; и что вы знаете обо всех этих правах.

Чтобы защитить ваши права, нам нужно ввести некоторые ограничения, которые запретят кому бы то ни было отказывать вам в этих правах или потребовать от вас отказаться от этих прав. Эти ограничения накладывают на вас некоторые обязательства, если вы распространяете копии ПО или изменяете его.

Например, если вы распространяете копии такой программы бесплатно или за вознаграждение, вы должны предоставить получателям все права, которыми обладаете вы сами. Вы должны гарантировать, что они тоже получат или смогут получить исходный код. Наконец, вы должны показать им текст данных условий, чтобы они знали о своих правах.

Мы защищаем ваши права в два этапа:

1. сохраняем авторские права на ПО, и
2. предлагаем вам эту лицензию, которая дает вам законное право копировать, распространять и/или модифицировать ПО.

Кроме того, в целях защиты как каждого автора, так и нас, мы хотим удостовериться, что каждый понимает, что гарантий на это свободное ПО нет. Если ПО модифицируется и передается кем-то еще, мы хотим, чтобы получатели ПО знали, что то, что у них есть, - это не оригинал, чтобы любые проблемы, созданные другими, не отразились на репутации первоначальных авторов.

И наконец, каждой свободной программе постоянно угрожают патенты на ПО. Мы хотим избежать той опасности, что повторные распространители свободной программы самостоятельно получат патенты, делая программу таким образом частной собственностью. Чтобы предотвратить это, мы со всей определенностью заявляем, что любой патент должен быть либо предоставлен всем для свободного использования, либо не предоставлен никому.

Ниже следуют точные определения и условия для копирования, распространения и модификации.

A.2. Определения и условия для копирования, распространения и модификации

- 0. Эта Лицензия применима к любой программе или другому произведению, содержащему уведомление, помещенное держателем авторских прав и сообщающее о том, что оно может распространяться при условиях, оговоренных в данной Универсальной Общественной Лицензии. В дальнейшем термин "Программа" относится к любой такой программе или произведению, а термин "произведение, основанное на Программе" означает Программу или любое произведение, содержащее Программу или ее часть, дословную, или модифицированную, и/или переведенную на другой язык. (Здесь и далее перевод включается без ограничений в понятие "модификация".) Каждый обладатель лицензии адресуется как "вы".

  Виды деятельности, не являющиеся копированием, распространением или модификацией, не охватываются данной Лицензией; они лежат за пределами ее влияния. Использование Программы по ее функциональному назначению не ограничено, а выходные данные Программы охватываются этой Лицензией, только если их содержание является произведением, основанным на Программе (вне зависимости от того, были ли они получены в процессе использования Программы). Являются ли они таковыми, зависит от того, что именно делает Программа.

- 1. Вы можете копировать и распространять дословные копии исходного кода Программы по его получении на любом носителе, при условии что вы соответствующим образом помещаете на видном месте в каждой копии соответствующее уведомление об авторских правах и отказ от предоставления гарантий; оставляете нетронутыми все уведомления, относящиеся к данной Лицензии и к отсутствию каких-либо гарантий; и передаете всем другим получателям Программы копию данной Лицензии вместе с Программой.

  Вы можете назначить плату за физический акт передачи копии и можете по своему усмотрению предоставлять гарантии за вознаграждение.

- 2. Вы можете изменять свою копию или копии Программы или любой ее части, создавая таким образом произведение, основанное на Программе, и копировать и распространять эти модификации или произведение в соответствии с Разделом 1, приведенным выше, при условии, что вы выполните все нижеследующие условия:

  1. Вы обязаны снабдить модифицированные файлы заметными уведомлениями, содержащими указания на то, что вы изменили файлы, и дату каждого изменения.
  2. Вы обязаны предоставить всем третьим лицам лицензию на бесплатное использование каждого произведения, которое вы распространяете или публикуете, целиком, и которое полностью или частично содержит Программу или какую-либо ее часть, на условиях, оговоренных в данной Лицензии.
  3. Если модифицированная программа обычно читает команды в интерактивном режиме работы, вы должны сделать так, чтобы при запуске для работы в таком интерактивном режиме обычным для нее способом она печатала или выводила на экран объявление, содержащее соответствующее уведомление об авторских правах и уведомление о том, что гарантий нет (или, наоборот, сообщающее о том, что вы обеспечиваете гарантии), и что пользователи могут повторно распространять программу при этих условиях, и указывающее пользователю, как просмотреть копию данной Лицензии. (Исключение: если сама Программа работает в интерактивном режиме, но обычно не выводит подобных сообщений, то ваше произведение, основанное на Программе, не обязано выводить объявление.)

  Эти требования применяются к модифицированному произведению в целом. Если известные части этого произведения не были основаны на Программе и могут обоснованно считаться независимыми и самостоятельными произведениями, то эта Лицензия и ее условия не распространяются на эти части, если вы распространяете их как отдельные произведения. Но если вы распространяете эти части как часть целого произведения, основанного на Программе, то вы обязаны делать это в соответствии с условиями данной Лицензии, распространяя права получателей лицензии на все произведение и, таким образом, на каждую часть, вне зависимости от того, кто ее написал.

Таким образом, содержание этого раздела не имеет цели претендовать на ваши права на произведение, написанное полностью вами, или оспаривать их; цель скорее в том, чтобы реализовать право управлять распространением производных или коллективных произведений, основанных на Программе.

Кроме того, простое нахождение другого произведения, не основанного на этой Программе, совместно с Программой (или с произведением, основанным на этой Программе) на одном носителе для постоянного хранения или распространяемом носителе не распространяет действие этой Лицензии на другое произведение.

- 3. Вы можете копировать и распространять Программу (или произведение, основанное на ней) согласно Разделу 2) в объектном коде или в выполнимом виде в соответствии с Разделами 1 и 2, приведенными выше, при условии, что вы также выполните одно из следующих требований:

  1. Сопроводите ее полным соответствующим машиночитаемым исходным кодом, который должен распространяться в соответствии с Разделами 1 и 2, приведенными выше, на носителе, который обычно используется для обмена ПО; или,
  2. Сопроводите ее письменным предложением, действительным по крайней мере в течение трех лет, предоставить любому третьему лицу за вознаграждение, не превышающее стоимость физического акта изготовления копии, полную машиночитаемую копию соответствующего исходного кода, подлежащую распространению в соответствии с Разделами 1 и 2, приведенными выше; или,
  3. Сопроводите ее информацией, полученной вами в качестве предложения распространить соответствующий исходный код. (Эта возможность допустима только для некоммерческого распространения, и только если вы получили программу в объектном коде или в исполняемом виде с предложением в соответствии с Пунктом b) выше.)

  Исходный код для произведения означает его вид, предпочтительный для выполнения в нем модификаций. Для исполняемого произведения полный исходный код означает все исходные коды для всех модулей, которые он содержит, плюс любые связанные с произведением файлы определения интерфейса, плюс сценарии, используемые для управления компиляцией и установкой исполняемого произведения. Однако, в виде особого исключения распространяемый исходный код не обязан включать то, что обычно предоставляется (как в объектных, так и в исходных кодах) с основными компонентами (компилятор, ядро и так далее) операционной системы, под управлением которой работает исполняемое произведение, за исключением случая, когда сам компонент сопровождает исполняемое произведение.

  Если распространение исполняемого произведения или объектного кода происходит путем предоставления доступа для копирования с обозначенного места, то предоставление доступа для копирования исходного кода с того же места считается распространением исходного кода, даже если третьи лица не принуждаются к копированию исходного кода вместе с объектным кодом.

- 4. Вы не можете копировать, изменять, повторно лицензировать, или распространять Программу никаким иным способом, кроме явно предусмотренных данной Лицензией. Любая попытка копировать, изменять или распространять Программу каким-либо другим способом или с измененной лицензией неправомерна и автоматически прекращает ваши права, данные вам этой Лицензией. Однако лицензии лиц, получивших от вас копии или права согласно данной Универсальной Общественной Лицензии, не прекращают своего действия, если эти лица полностью соблюдают условия.
- 5. Вы не обязаны соглашаться с этой Лицензией, так как вы не подписывали ее. Однако, ничто, кроме этой Лицензии, не дает вам право изменять или распространять эту Программу или основанные на ней произведения. Эти действия запрещены законом, если вы не принимаете к соблюдению эту Лицензию. А значит, изменяя или распространяя Программу (или произведение, основанное на Программе), вы изъявляете свое согласие с этой Лицензией и всеми ее условиями о копировании, распространении или модификации Программы или основанных на ней произведений.
- 6. Каждый раз, когда вы повторно распространяете Программу (или любое произведение, основанное на Программе), получатель этого произведения автоматически получает от первоначального выдавшего лицензию лица свою лицензию на копирование, распространение или модификацию Программы, обсуждаемую в этих определениях и условиях. Вы не можете налагать каких-либо дополнительных ограничений на осуществление получателем прав, предоставленных данным документом. Вы не несете ответственности за соблюдение третьими лицами условий этой Лицензии.
- 7. Если в результате судебного разбирательства, или обвинения в нарушении патента или по любой другой причине (не обязательно связанной с патентами), вам навязаны условия, противоречащие данной

Лицензии (по постановлению суда, по соглашению или иным способом), это не освобождает вас от соблюдения Лицензии. Если вы не можете заниматься распространением так, чтобы одновременно удовлетворить требованиям и этой Лицензии, и всем другим требованиям, то вы не должны заниматься распространением Программы. Например, если патент не позволяет безвозмездное повторное распространение Программы всем, кто получил копии от вас непосредственно или через посредников, то единственным способом удовлетворить и патенту, и этой Лицензии будет ваш полный отказ от распространения Программы.

Если какая-либо часть этого раздела не имеет силы или не может быть исполнена при некоторых конкретных обстоятельствах, то подразумевается, что имеет силу остальная часть раздела, а при других обстоятельствах имеет силу весь Раздел.

Цель этого раздела - не побудить вас делать заявления о нарушениях прав на патент, или заявлять о других претензиях на право собственности или оспаривать правильность подобных претензий; единственная цель этого раздела - защита целостности системы распространения свободного ПО, которая реализуется использованием общественных лицензий. Многие люди внесли щедрый вклад в широкий спектр ПО, распространяемого по этой системе, полагаясь на ее согласованное применение; только автору принадлежит право решать, хочет ли он или она распространять ПО в этой системе или в какой-то другой, и получатель лицензии не может влиять на принятие этого решения.

Этот раздел предназначен для того, чтобы тщательно прояснить, что полагается следствием из остальной части данной Лицензии.

- 8. Если распространение и/или применение Программы ограничено в ряде стран либо патентами, либо авторскими правами на интерфейсы, первоначальный обладатель авторских прав, выпускающий Программу с этой Лицензией, может добавить явное ограничение на географическое распространение, исключив такие страны, так что распространение разрешается только в тех странах, которые не были исключены. В этом случае данная Лицензия включает в себя это ограничение, как если бы оно было написано в тексте данной Лицензии.
- 9. Фонд Свободного ПО может время от времени публиковать пересмотренные и/или новые версии Универсальной Общественной Лицензии. Такие новые версии будут сходны по духу с настоящей версией, но могут отличаться в деталях, направленных на новые проблемы или обстоятельства.

  Каждой версии придается отличительный номер. Если в Программе указывается, что к ней относится некоторый номер версии данной Лицензии и "любая последующая версия", вы можете по выбору следовать определениям и условиям либо данной версии, либо любой последующей версии, опубликованной Фондом Свободного ПО. Если в Программе не указан номер версии данной Лицензии, вы можете выбрать любую версию, когда-либо опубликованную Фондом Свободного ПО.

- 10. Если вы хотите встроить части Программы в другие свободные программы с иными условиями распространения, напишите автору с просьбой о разрешении. Для ПО, которое охраняется авторскими правами Фонда Свободного ПО, напишите в Фонд Свободного ПО; мы иногда делаем такие исключения. Наше решение будет руководствоваться двумя целями: сохранения свободного статуса всех производных нашего свободного ПО и содействия совместному и повторному использованию ПО вообще.

НИКАКИХ ГАРАНТИЙ

- 11. ПОСКОЛЬКУ ПРОГРАММА ПРЕДОСТАВЛЯЕТСЯ БЕСПЛАТНО, НА ПРОГРАММУ НЕТ ГАРАНТИЙ В ТОЙ МЕРЕ, КАКАЯ ДОПУСТИМА ПРИМЕНИМЫМ ЗАКОНОМ. ЗА ИСКЛЮЧЕНИЕМ ТЕХ СЛУЧАЕВ, КОГДА ОБРАТНОЕ ЗАЯВЛЕНО В ПИСЬМЕННОЙ ФОРМЕ, ДЕРЖАТЕЛИ АВТОРСКИХ ПРАВ И/ИЛИ ДРУГИЕ СТОРОНЫ ПОСТАВЛЯЮТ ПРОГРАММУ "КАК ОНА ЕСТЬ" БЕЗ КАКОГО-ЛИБО ВИДА ГАРАНТИЙ, ВЫРАЖЕННЫХ ЯВНО ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ИМИ, ПОДРАЗУМЕВАЕМЫЕ ГАРАНТИИ КОММЕРЧЕСКОЙ ЦЕННОСТИ И ПРИГОДНОСТИ ДЛЯ КОНКРЕТНОЙ ЦЕЛИ. ВЕСЬ РИСК В ОТНОШЕНИИ КАЧЕСТВА И ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММЫ ОСТАЕТСЯ ПРИ ВАС. ЕСЛИ ПРОГРАММА ОКАЖЕТСЯ ДЕФЕКТНОЙ, ВЫ ПРИНИМАЕТЕ НА СЕБЯ СТОИМОСТЬ ВСЕГО НЕОБХОДИМОГО ОБСЛУЖИВАНИЯ, ВОССТАНОВЛЕНИЯ ИЛИ ИСПРАВЛЕНИЯ.
- 12. НИ В КОЕМ СЛУЧАЕ, ЕСЛИ НЕ ТРЕБУЕТСЯ СООТВЕТСТВУЮЩИМ ЗАКОНОМ, ИЛИ НЕ УСЛОВЛЕНО В ПИСЬМЕННОЙ ФОРМЕ, НИ ОДИН ДЕРЖАТЕЛЬ АВТОРСКИХ ПРАВ И НИ

ОДНО ДРУГОЕ ЛИЦО, КОТОРОЕ МОЖЕТ ИЗМЕНЯТЬ И/ИЛИ ПОВТОРНО РАСПРОСТРАНЯТЬ ПРОГРАММУ, КАК БЫЛО РАЗРЕШЕНО ВЫШЕ, НЕ ОТВЕТСТВЕННЫ ПЕРЕД ВАМИ ЗА УБЫТКИ, ВКЛЮЧАЯ ЛЮБЫЕ ОБЩИЕ, СПЕЦИАЛЬНЫЕ, СЛУЧАЙНЫЕ ИЛИ ПОСЛЕДОВАВШИЕ УБЫТКИ, ПРОИСТЕКАЮЩИЕ ИЗ ИСПОЛЬЗОВАНИЯ ИЛИ НЕВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ ПРОГРАММЫ (ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОТЕРЕЙ ДАННЫХ, ИЛИ ДАННЫМИ, СТАВШИМИ НЕПРАВИЛЬНЫМИ, ИЛИ ПОТЕРЯМИ, ПОНЕСЕННЫМИ ИЗ-ЗА ВАС ИЛИ ТРЕТЬИХ ЛИЦ, ИЛИ ОТКАЗОМ ПРОГРАММЫ РАБОТАТЬ СОВМЕСТНО С ЛЮБЫМИ ДРУГИМИ ПРОГРАММАМИ), ДАЖЕ ЕСЛИ ТАКОЙ ДЕРЖАТЕЛЬ ИЛИ ДРУГОЕ ЛИЦО БЫЛИ ИЗВЕЩЕНЫ О ВОЗМОЖНОСТИ ТАКИХ УБЫТКОВ.

КОНЕЦ ОПРЕДЕЛЕНИЙ И УСЛОВИЙ

Приложение B. GNU *Лицензия Свободной Документации*

B.1. GNU Free Documentation License

Версия 1.1, Март 2000. Перевод является неофициальным

0. ПРЕАМБУЛА

Цель настоящей Лицензии состоит в том, чтобы сделать руководство, учебник или любой другой письменный документ "свободным", что означает предоставить каждому человеку эффективную свободу копировать и распространять, с изменениями или без изменений, коммерческим или любым другим некоммерческим образом эти документы. Кроме того, эта Лицензия защищает права автора и издателя на признание их работы и снимает с них ответственность за модификации, выполненные другими.

Данная Лицензия является своего рода "copyleft" (сохранение прав копирования); это означает, что любые производные документы должны быть сами по себе свободны, - в том же самом смысле, что и сам документ. Эта лицензия является дополнением к Универсальной Общественной Лицензии GNU (GNU General Public License), которая является "copyleft" лицензией, разработанной для свободного программного обеспечения.

Мы разработали эту Лицензию для того, чтобы использовать ее применительно к руководствам для свободного программного обеспечения, потому что свободное программное обеспечение нуждается в свободной документации: свободная программа должна поставляться с руководствами, обеспечивающими те возможности, которые предоставляет программное обеспечение. Действие этой Лицензии может распространяться не только на руководства по программному обеспечению; она может использоваться для любого текстового документа, независимо от темы, содержимого или способа публикации, в том числе и для опубликованных традиционным способом книг. Мы рекомендуем эту Лицензию преимущественно для работ, которые являются руководствами и справочниками.

1. ОПРЕДЕЛЕНИЯ И ПРИМЕНИМОСТЬ

Настоящая Лицензия может быть применена к любому руководству или другому документу, который содержит сообщающее об этом примечание, помещенное владельцем авторских прав. Словом "Документ" ниже называется любая такая документация или работа. Каждый член общества лицензирован, и это в полной мере относится и к "вам".

"Измененная версия" (Modified Version) Документа - это любая работа, содержащая Документ частично или полностью, любая копия Документа, точная или измененная, а также любой перевод на другой язык.

"Дополнительный Раздел" (Secondary Section) - это именованное приложение или титульная часть основного Документа, которые содержат исключительно вопросы отношения издателей или авторов Документа к самому Документу и не должны содержать ничего, что могло бы быть изложено непосредственно в пределах основной темы. (Например, если Документ или его часть - это учебник математики, Дополнительные Разделы не могут содержать темы, касающиеся математики). Эти отношения могут отображать историческую связь с предметом или смежными областями, или юридическую, коммерческую, философскую, этическую или политическую позицию, которой придерживались авторы при рассмотрении предмета.

"Неизменяемые Разделы"(Invariant Sections) - это те Дополнительные Разделы, заголовки которых описаны как заголовки Неизменяемых Разделов в тексте, указывающем, что этот Документ выпущен под данной Лицензией.

"Тексты Обложек"(Cover Texts) - это текстовые фрагменты, помещенные на передней или задней обложке; в них содержится уведомление о том, что данный Документ выпущен согласно данной Лицензии.

"Прозрачная"(Transparent) копия Документа означает машинно-читаемую копию, представленную в формате, спецификация которого доступна широкой публике; ее содержание может быть просмотрено и отредактировано прямо и непосредственно универсальными текстовыми редакторами, или (для изображений,

состоящих из пикселов) универсальными редакторами изображений, или (для рисунков) с помощью доступных всем редакторов рисунков. Копия, сделанная в "прозрачном" формате таким образом, чтобы помешать или воспрепятствовать последующему изменению читателем, не является Прозрачной. Копия, не являющаяся Прозрачной, называется "Темной"(Opaque).

Примеры форматов, подходящих для Прозрачных копий, включают plain ASCII без разметки, Texinfo input format, LaTeX input format, SGML или XML с использованием публично доступных DTD, а также стандартный простой HTML. Темные форматы включают PostScript, PDF, патентованные форматы компаний, которые могут быть прочитаны и отредактированы только патентованными текстовыми редакторами, SGML или XML, для которых DTD и/или средства обработки не являются публично доступными, а также автоматически сгенерированные HTML, производимые некоторыми редакторами только для вывода.

"Титульная страница"(Title Page) печатного документа обозначает собственно титульную страницу плюс последующие страницы, необходимые для четкого изложения материалов, которые по условиям этой Лицензии должны находиться на титульной странице. При работе в форматах, которые не имеют титульной страницы как таковой, "Титульная страница" обозначает текст, расположенный в самой заметной области около заголовка работы, перед началом собственно текста.

2. ДОСЛОВНАЯ КОПИЯ

Вы имеете право копировать и распространять этот Документ любыми способами, коммерческими или некоммерческими, с условием, что эта Лицензия, уведомление о копирайте и уведомление о лицензии, приложенные к Документу, будут воспроизведены во всех копиях, и что вы не добавите какие бы то ни было условия к изложенным в этой Лицензии. Вы имеете право не использовать технические средства для ограничения или учета чтения или дальнейшего копирования копий, распространяемых вами. Однако вы можете взымать плату за копии. Если вы распространяете достаточно большое число копий, вы должны выполнять условия, описанные в п.3

На условиях, указанных выше, вы можете также одалживать копии или публично демонстрировать их.

3. МНОЖЕСТВЕННОЕ КОПИРОВАНИЕ

Если вы публикуете копии Документа в количестве, превышающем 100, и лицензия Документа перечисляет Тексты Обложек, вы должны заключить каждую копию в обложку, которая содержит, ясно и четко, все указанные Тексты Обложек: Текст Верхней Обложки на верхней обложке и Текст Нижней Обложки на нижней обложке. Обе обложки должны также ясно и четко ссылаться на вас как на издателя этих копий. На верхней обложке должен быть указан полный заголовок, причем все слова заголовка должны быть в равной мере заметны и видны. Вы имеет право добавить дополнительный материал на обложку. Копирование с изменениями, которые касаются только обложки, если они сохраняют заголовок документа и удовлетворяют изложенным условиям, может считаться во всех отношениях дословным копированием.

Если текст, который требуется расположить на какой-то из обложек, слишком велик, чтобы поместиться на ней, вы должны опубликовать его начало (в разумном объеме) на вашей обложке и продолжить его на соседних страницах.

Если вы публикуете или распространяете Темные копии Документа в количестве, превышающем 100, вы должны либо приложить машинно-читаемую Прозрачную копию к каждой Темной копии, либо указать в каждой Темной копии публично доступный компьютерный адрес, содержащий полную Прозрачную копию Документа, свободную от дополнительного материала, которую любой пользователь компьютерной сети мог бы скачать анонимно и бесплатно, используя публичный стандарт сетевого протокола. Если вы используете последний вариант, вы должны предпринять разумные меры, чтобы при распространении большого числа Темных копий вы могли бы гарантировать, что Прозрачная копия останется доступной указанным способом в указанном месте по меньшей мере в течение года после того, как вы публично распространили последнюю Темную копию (непосредственно или через ваших дилеров или агентов).

Является очень желательным, хотя и необязательным, связаться с авторами Документа перед распространением любого заметного числа его копий, чтобы его авторы могли обеспечить вас обновленной версией Документа.

4. ИЗМЕНЕНИЯ

Вы имеете право копировать и распространять Измененную версию Документа с соблюдением правил, изложенных в пп.2 и 3 (см. выше), при условии, что вы издаете Измененную версию в точности под этой Лицензией, где Измененная версия выполняет роль Документа, разрешая таким образом распространение и изменение Измененной версии всем, имеющим ее копию. Кроме этого, в Измененной версии вы должны сделать следующее:

A. использовать на Титульной странице (и на обложках, если таковые есть) заголовок, отличный от заголовка Документа и от заголовков его предыдущих версий (которые, если есть, должны быть перечислены в разделе "История" этого Документа). Вы можете использовать заголовок предыдущей версии, если первичный издатель этой версии разрешает его использование;

B. перечислить на Титульной странице в качестве авторов одно или несколько имен или названий, ответственных за авторство изменений в Измененной версии, а также список, упоминающий по меньшей мере пять исходных авторов Документа (если их меньше пяти, то всех);

C. указать на Титульной странице в качестве издателя издателя Измененной версии;

D. сохранить все уведомления в Документе, касающиеся копирайта;

E. добавить соответствующие уведомления о копирайте, касающиеся ваших изменений, рядом с другими уведомлениями о копирайте;

F. вставить, сразу после уведомления о копирайте, лицензионное уведомление, дающее разрешение на публичное использование Измененной версии под условием настоящей Лицензии в форме, указанной ниже в Приложении;

G. сохранить в этом лицензионном уведомлении полный список Неизменяемых разделов и требуемых Текстов Обложек, указанных в лицензионном уведомлении Документа;

H. включить неизмененную копию этой Лицензии;

I. сохранить раздел, озаглавленный "История", и его заголовок, и добавить в него указание по меньшей мере заголовка, года, новых авторов и издателей Измененной версии, как это указано на Титульной странице. Если в Документе нет раздела, озаглавленного "История", создать такой, указав в нем заголовок, год, авторов и издателей Документа, как это указано на его Титульной странице, а затем добавить описание Измененной версии так, как указано в предыдущем предложении;

J. сохранить компьютерный адрес (если есть), указанный в Документе в качестве адреса публичного доступа к Прозрачной копии Документа, а также компьютерные адреса, указанные в Документе, для доступа к предыдущим его версиям. Адреса могут быть размещены в разделе "История". Вы можете опустить упоминания адреса работы, если она была опубликована более чем за четыре года до самого Документа, или если первичный издатель версии, находящейся по данному адресу, дал на это свое разрешение;

K. в любом разделе, названном "Благодарности" или "Посвящения", сохранить заголовок раздела, а также содержание и тон каждой благодарности и/или посвящения, в них размещенных;

L. сохранить все Неизменяемые разделы Документа, не внося никаких изменений ни в их текст, ни в их заголовки. Номера разделов не считаются частью заголовков раздела;

M. удалить любой раздел, озаглавленный "Поддержка". Такой раздел не может быть включен в Измененную версию;

N. не переименовывать никакой существующий раздел в "Endorsements" (Поддержка), или таким образом, чтобы он совпал с заголовком любого Неизменяемого раздела.

Если в Измененную версию включены новые предисловия или приложения, которые могут быть определены как Дополнительные разделы и которые не содержат текста, скопированного из Документа, вы имеет право по своему усмотрению определить все или некоторые из этих разделов как Неизменяемые. Для этого надо добавить их заголовки в список Неизменяемых разделов в уведомлении о лицензии Измененной версии. Эти заголовки должны отличаться от заголовков всех остальных разделов.

Вы можете добавить раздел с заголовком "Поддержка", при условии, что он не содержит ничего, кроме информации о поддержке вашей Измененной версии разными сторонами - например, изложение экспертной оценки, или что этот текст был утвержден какой-то организацией в качестве официального определения стандарта.

Вы имеете право добавить до пяти слов в Текст Верхней Обложки и до 25 слов в Текст Нижней обложки, в конец списка Текстов Обложек в Измененной версии. Может быть добавлен только один отрывок из Верхней Обложки и один из Нижней Обложки. Если Документ уже включает текст для той же самой обложки, который был ранее добавлен вами или усилиями стороны, от имени которой вы выступаете, вы можете не добавлять другого текста, но вы можете заменить старый текст, если на это дал разрешение издатель, включивший в Документ старый текст.

Автор (авторы) и издатель (издатели) Документа не дают по данной Лицензии разрешения использовать их имена для рекламы или для поддержки любой Измененной версии.

5. ОБЪЕДИНЕНИЕ ДОКУМЕНТОВ

Вы можете объединять Документ с другими документами, изданными под этой Лицензией, при соблюдении условий, описанных в секции 4 для измененных версий, включив объединение всех Неизменяемых разделов всех исходных документов в неизмененном виде и перечислив их как Неизменяемые разделы вашего объединенного документа в его лицензионном уведомлении.

В объединенном документе должна содержаться только одна копия Лицензии, а совпадающие Неизменяемые разделы могут быть представлены только одной копией. Если существует несколько Неизменяемых разделов с одним и тем же заголовком, каждый заголовок надо сделать уникальным, добавив после него, в скобках, имя первичного автора или издателя этого раздела, если таковой известен, или уникальный номер. Соответствующие добавления надо произвести в заголовках разделов в списке Неизменяемых разделов в лицензионном уведомлении объединенного документа.

В объединенном документе вы должны объединить все разделы "История" всех исходных документов в один раздел "История", таким же образом поступить с разделами под заголовками "Благодарности" или "Посвящения". Вы должны удалить все разделы с заголовком "Поддержка".

6. СОБРАНИЕ ДОКУМЕНТОВ

Вы можете создать сборник, состоящий из Документа и других документов, изданных под настоящей Лицензией, и заменить отдельные копии этой Лицензии, приложенные к каждому Документу, единственной копией, приложенной к сборнику, при условии, что вы следуете правилам этой Лицензии для дословного копирования каждого документа сборника во всех других отношениях.

Вы можете взять один документ из сборника и распространять его отдельно под настоящей Лицензией, при условии, что вы прилагаете копию этой Лицензии к извлеченному документу и следуете Лицензии во всех отношениях, касающихся дословной копии этого документа.

7. КОМПИЛЛЯЦИЯ С НЕЗАВИСИМЫМИ ДОКУМЕНТАМИ

Компилляция Документа или его производных с другими отдельными и независимыми документами или работами на одном устройстве хранения или распространения информации не влечет за собой возникновения Измененной версии Документа, при условии, что не заявляется никакой копирайт на создание этой компилляции. Такая компилляция называется "подборкой", и данная Лицензия неприменима к другим самостоятельным работам, находящимся в одной подборке с Документом, если эти работы не являются производными Документа.

Если требования по Тексту Обложки (п. 3) применимы к этим копиям Документа, и если при этом Документ занимает менее четверти всего объема подборки, Тексты Обложек Документа могут быть размещены на обложках только вокруг самого Документа внутри подборки. В противном случае они должны быть размещены на обложках самой подборки.

8. ПЕРЕВОД

Перевод считается одним из способов изменения, так что вы имеете право распространять переводы Документа на условиях, описанных в п.4. Замена Неизменяемых разделов их переводами требует специального разрешения от соответствующих правообладателей, но вы можете включить переводы некоторых или всех Неизменяемых разделов в дополнение к исходным версиям этих Неизменяемых разделов. Вы можете включить перевод этой Лицензии с условием, что вы приложите к нему исходную

английскую версию Лицензии. В случае расхождений между переводом и исходной английской версией этой Лицензии предпочтение отдается тексту Лицензии на английском языке.

9. ПРЕКРАЩЕНИЕ ДЕЙСТВИЯ

Вы не имеете права копировать, изменять, передавать права или распространять Документ иначе, как на условиях настоящей Лицензии. Любая другая попытка копировать, изменять, передавать права или распространять Документ является недействительной и автоматически ведет к прекращению ваших прав, полученных по этой Лицензии. Однако стороны, получившие копии или права от вас по условиям этой Лицензии, не лишаются своих прав при условии полного соблюдения настоящей Лицензии.

10. ПЕРЕСМОТР УСЛОВИЙ ЛИЦЕНЗИИ

Free Software Foundation (Фонд свободного программного обеспечения) может время от времени публиковать новые версии GFDL (GNU Free Documentation License). Эти версии обязательно сохранят дух настоящей Лицензии, однако могут отличаться от нее в деталях, регулирующих правоотношения, которые возникли после опубликования предыдущих версий. Смотрите Copyleft (`http://www.gnu.org/copyleft/`).

Каждой версии Лицензии присваивается свой собственный номер версии. Если в Документе указано, что к нему применима Лицензия данной версии и все более поздние, вы можете выполнять условия этой или любой более поздней версии Лицензии, которая была опубликована Free Software Foundation (не как проект). Если в Документе не указан номер версии Лицензии, вы можете выбрать любую опубликованную Free Software Foundation (не как проект) версию Лицензии.

B.2. Как использовать Лицензию для ваших документов

Чтобы использовать Лицензию для созданных вами документов, включите копию Лицензии в документ и разместите следующие уведомления о правах и Лицензии сразу за Титульной страницей:

> Copyright (c) ГОД ВАШЕ ИМЯ. Разрешается копировать, распространять и/или изменять этот документ при соблюдении условий Лицензии GNU Free Documentation License, версия 1.1 или любая более поздняя версия, опубликованная Free Software Foundation; Неизменяемые разделы: СПИСОК ЗАГОЛОВКОВ, с Текстом Верхней Обложки СПИСОК, и с Текстами Нижней Обложки СПИСОК. Копия Лицензии включена в раздел, озаглавленный "GNU Free Documentation License".

Если у вас нет Неизменяемого раздела, укажите "Неизменяемого раздела нет" вместо списка. Если у вас нет Текста Верхней Обложки, напишите "Текста Верхней Обложки нет" вместо перечисления списка; аналогично для Текста Нижней Обложки.

Если в вашем документе содержатся оригинальные примеры программного кода, мы советуем выпустить их отдельно в соответствии с условиями одной из лицензий на свободное программное обеспечение, например, GNU General Public License, чтобы разрешить использовать их как свободное программное обеспечение.

Приложение C. Глоссарий

APM

Advanced Power Management (расширенное управление питанием). Используемая некоторыми *BIOS*-ами возможность, благодаря которой машина может войти в состояние standby ("заснуть") через некоторый промежуток времени неактивности. На портативных компьютерах (laptop) APM также отвечает за статус состояния батарей и (если это поддерживается) оставшееся время жизни батарей.

ASCII

American Standard Code for Information Interchange (Американский Стандартный Код для Информационного Обмена). Стандартный код, используемый для сохранения символов, включая управляющие символы, на компьютере. Множество 8-битных кодов (таких, как ISO 8859-1, стандартный набор символов для Linux), содержат ASCII в своей нижней половине.
See Also: .

BSD

Berkeley Software Distribution. Вариант *UNIX*, разработанный в компьютерном департаменте университета Berkeley. Эта версия всегда рассматривалась как технически более расширенная, чем другие, и принесла множество инноваций миру вообще и *UNIX* в частности.

CHAP

Challenge-Handshake Authentication Protocol: протокол, используемый ISP для авторизации своих клиентов. По этой схеме, клиенту (машине, которая устанавливает соединение) посылается некоторое значение, которое используется для вычисления *hash*, основанного на значении. Клиент посылает хеш обратно на сервер для сравнения с хешем, подсчитанным сервером. Этот метод авторизации отличается от PAP тем, что периодически происходит переавторизация в процессе сеанса.
See Also: PAP.

CIFS

Common Internet FileSystem (Общесетевая файловая система) Потомок файловой системы SMB, используемый для систем типа *DOS*.

DHCP

Dynamic Host Configuration Protocol (Протокол динамической настройки хоста). Протокол разработан для того, чтобы машины локальной сети могли получать динамически IP адрес от DHCP сервера.

DMA

Direct Memory Access (Прямой доступ к памяти). Это возможность, используемая в архитектуре *PC*, которая позволяет периферийному оборудованию получить доступ к главной памяти (читать или писать), не используя CPU. PCI-периферия использует bus mastering (управление по шине) и не нуждается в DMA.

DNS

Domain Name System (Система доменных имен). Распределенный механизм сопоставления имени и адреса, используемый в Интернет. Этот механизм связывает доменное имя с IP адресом, позволяя искать сайт по доменному имени, не зная его IP адреса. DNS также дает возможность обратного поиска, то есть получения IP машины по ее доменному имени.

DPMS

Display Power Management System. Используемый всеми современными мониторами протокол для управления возможностями экономии энергии. Мониторы, которые поддерживают эти возможности, обычно называют "green" мониторами (зелеными).

ELF

Executable and Linking Format (Исполняемый и линкуемый формат). Это бинарный формат, используемый большинством *GNU/Linux* дистрибутивов.

ext2

сокращение от "Extended 2 file system" (вторая расширенная файловая система). Это родная для *GNU/Linux* файловая система и у нее есть все типичные параметры для любой файловой системы *UNIX*: поддержка специальных файлов (символьных устройств, символических ссылок и т.д.), разграничение прав доступа к файлам и другие.

FAQ

Frequently Asked Questions (Часто задаваемые вопросы). Это документ, содержащий ряд вопросов и ответов по определенной теме. Исторически, FAQ появились в группах новостей, то теперь документы такого типа имеются на многих web-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, которая используется в *DOS* и *Windows*.

FDDI

Fiber Distributed Digital Interface (распределенный интерфейс передачи данных по волоконно-оптическим каналам) . Высокоскоростной сетевой протокол физического уровня, который использует для передачи информации оптоволоконный кабель. Используется преимущественно в больших сетях, в основном из-за своей стоимости. Редко используется для подключения компьютера к сетевым коммутатору (switch).

FHS

File system Hierarchy Standard (Стандарт иерархии файловой системы). Документ, содержащий рекомендации для последовательной организации файлового дерева в системах *UNIX*. **Mandrake Linux** в основном придерживается этого стандарта.

FIFO

First In, First Out (первым вошел, первым вышел). Структура данных или аппаратный буфер, в которых элементы используются в порядке их поступления. Трубы в *UNIX* являются самым типичным примером FIFO.

FTP

File Transfer Protocol (Протокол передачи файлов). Это стандартный для Интернет протокол передачи файлов от одной машины к другой.

GFDL

Лицензия Свободной Документации GNU (GNU Free Documentation License). Документация к **Mandrake Linux** поставляется по этой лицензии.

GIF

Graphics Interchange Format (формат графического обмена). Формат графического файла, широко используемый в web. Изображения GIF бывают сжатыми или содержат анимацию. В связи с проблемами копирайта на этот формат, использовать его не очень хорошая идея, поэтому мы рекомендуем заменять их по возможности форматом PNG.

GNU

GNU's Not Unix (GNU это не UNIX) . Проект GNU был основан в начале 80-х годов. Основателем был Richard Stallman. Цель этого проекта это разработка свободной операционной системы ("свободный" в смысле "свободы слова"). В настоящий момент в него входят практически все программы, кроме... ядра. Проект GNU kernel, *Hurd*, еще не является полностью монолитным. *Linux*, среди прочего, заимствует две вещи из GNU: его компилятор *C* , gcc, и его лицензию GPL.
See Also: GPL.

GPL

General Public License (Универсальная Общественная Лицензия) . Лицензия ядра *GNU/Linux*, которая идет в противовес всем частным лицензиям, в которых не дается прав на копирование, изменение и распространение программного обеспечения, пока исходный код не стал доступным. Единственным ограничением является то, что человек, которому вы передаете код, должен тоже иметь возможность извлечь выгоду из тех же самых прав.

GUI

Graphical User Interface (Графический интерфейс пользователя) . Интерфейс к компьютеру, содержащий окна с меню, кнопками, пиктограммами и т.д. Большинство предпочитает GUI вместо CLI ((*Command Line Interface* (Интерфейс Командной Строки)) за то, что он проще в использовании, несмотря на то, что последний более универсальный.

HTML

HyperText Markup Language (гипертекстовый язык разметки). Язык, используемый для создания web-документов.

HTTP

HyperText Transfer Protocol (Протокол передачи гипертекстов) . Протокол, используемый для соединения с web-сайтами и получения документов HTML или файлов.

IDE

Integrated Drive Electronics (встроенный интерфейс накопителей) . Наиболее широко используемая в современных *компьютерах* шина для жестких дисков. Шина IDE может содержать до двух устройств, а скорость шины ограничена устройством с более медленной очередью команд (но не более медленной скоростью передачи!)
See Also: ATAPI.

IP masquerading

(IP маскарадинг) Это методика, в которой используется файервол для того, чтобы спрятать снаружи реальный IP адрес вашего компьютера. Таким образом все внешние соединения, которые вы проводите сквозь файервол, будут наследовать IP адрес файервола. Это полезно в ситуациях, когда вам надо сделать быстрое подключение к сети Интернет, имея только один IP адрес и несколько машин в вашей внутренней локальной сети.

IRC

Internet Relay Chat . Один из нескольких стандартов Интернет для живого общения. Позволяет создавать каналы, вести частные беседы и обмениваться файлами. Кроме того, сервера могут соединяться друг с другом, поэтому сегодня существует несколько сетей IRC: **Undernet**, **DALnet**, **EFnet** и другие.

IRC :0=0;K

это "места" внутри IRC серверов, где можно общаться с другими людьми. Каналы созданы в IRC серверах и пользователи могут присоединяться (join) к этим каналам, чтобы общаться друг с другом. Сообщения, написанные на канале, видны только тем людям, которые подключены к этому каналу. Два или более пользователей могут создать "частный" канал и общаться, не мешая другим пользователям. Названия каналов начинаются с **#**.

ISA

Industry Standard Architecture (архитектура, соответствующая промышленному стандарту) . Самая первая шина, которая использовалась на *PC*. Она постепенно была заменена шиной PCI. ISA еще можно встретить на SCSI картах поддержки сканеров, пишущих CD и некотором старом оборудовании.

ISDN

Integrated Services Digital Network (цифровая сеть связи с комплексными услугами). Набора коммуникационных стандартов для голосовой связи, сервисов цифровых сетей и передачи видео. Разработана для замены существующей системы телефонной связи, известной как PSTN (*Public Switched Telephone Network*) или POTS (*Plain Ole Telephone Service*). ISDN известна как сеть передачи данных с коммутацией каналов.

ISO

International Standards Organization (Международная организация по стандартизации) . Группа компаний, консультантов, университетов и других источников, которая составляет стандарты в различных дисциплинах, включая компьютеры. Документы, описывающие стандарты, пронумерованы. Например, номер стандарта iso9660, описывает файловую систему, используемую на носителях CD-ROM.

ISP

Internet Service Provider (Поставщик Услуг Интернет) . Компания, которая продает доступ к Интернет своим клиентам, по телефонным линиям или широкополосным линиям, например выделенным линиям T-1, DSL или кабельным модемам.

JPEG

Joint Photographic Experts Group (Совместная экспертная группа по фотографии) . Еще один очень распространенный формат файлов изображений. JPEG больше всего подходит для фотографий реального мира, и не очень хорошо работает с рисованной графикой.

LAN

Local Area Network (Локальная сеть) . Универсальное название, которое дано сети машин, связанных одним и тем же физическим проводом.

LDP

Linux Documentation Project (Проект документации по Linux) . Некоммерческая организация, которая поддерживает документацию по *GNU/Linux*. Наиболее известные ее документы это *HOWTOs*, кроме этого она также занимается FAQ и даже некоторыми книгами.

MBR

Master Boot Record (главная загрузочная запись) . Так называется первый сектор загрузочного жесткого диска. MBR содержит код, который используется для загрузки операционной системы в память или загрузчик (например, *LILO*), а также таблицу разделов жесткого диска.

MIME

Multipurpose Internet Mail Extensions (многоцелевые расширения электронной почты в сети Internet) . Строка в форме тип/подтип, описывающая содержимое файла, присоединенного (attached) к электронному письму (e-mail). Позволяет почтовым клиентам, которые понимают MIME, определять какие действия производить с файлом, в зависимости от его типа.

MPEG

Moving Pictures Experts Group . Комитет ISO, который устанавливает стандарты для видео и аудио компрессии. MPEG также является названием алгоритма. К сожалению, лицензия на данный формат весьма ограничена и, как следствие, пока не существует полностью *Open Source* MPEG проигрывателей...

NCP

NetWare Core Protocol. Протокол, определенный **Novell** для доступа к файлу Novell NetWare и сервисам печати.

NFS

Network FileSystem . Сетевая файловая система, созданная **Sun Microsystems** для прозрачного совместного использования файлов в сети.

NIC

Network Interface Controller (контроллер сетевого интерфейса). Адаптер, установленный на компьютер, который предоставляет физическое соединение с сетью, например *Ethernet* карта.

NIS

Network Information System (Сетевая Информационная Система). NIS еще известна как "Yellow Pages" (Желтые страницы), но копирайт на это имя принадлежит **British Telecom** . NIS это протокол, разработанный в **Sun Microsystems**, для совместного использования общей информации в **домене** NIS, который может состоять из полной локальной сети или только ее части. Он может экспортировать базы паролей, базы данных сервисов, информацию о группах и другое.

PAP

Password Authentication Protocol (протокол авторизации паролей). Этот протокол используется большинством ISP для авторизации своих клиентов.По этой схеме клиент (то есть вы) посылает пару идентификатор/пароль на сервер, но эта информация не является шифрованной. Читайте описание CHAP, более защищенной системы.
See Also: CHAP.

PCI

Peripheral Components Interconnect. Созданная в **Intel** шина, которая сегодня является стандартом для *персональных компьютеров* и других архитектур. Она является потомком ISA и предлагает множество сервисов: идентификация устройства, информация о конфигурации, разделение IRQ, bus mastering и другое.

PCMCIA

Personal Computer Memory Card International Association (Международная ассоциация производителей плат памяти для персональных компьютеров) . Чаще, для простоты, называемая "PC Card", является стандартом для внешних карт, подключаемых к переносным компьютерам (laptop): модемы, жесткие диски, карты памяти, *Ethernet* карты и другое. Иногда, в шутку акроним расшифровывают как *People Cannot Memorize Computer Industry Acronyms* (люди не в силах запоминать акронимы компьютерной промышленности) ...

PNG

Portable Network Graphics (портируемая сетевая графика). Формат файла изображений, созданный преимущественно для использования в web, разработанный как свободная от патентов замена GIF. Имеет несколько дополнительных возможностей.

PnP

Plug'N'Play. Первое дополнение к ISA для того, чтобы получить возможность добавлять конфигурационную информацию к устройствам. Стало более широко распространенным условием, по которому группируются все устройства, способные сообщить о их параметрах конфигурации. Все PCI устройства являются 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.

PPP

Point to Point Protocol (протокол точка-точка) . Этот протокол используется для передачи данных по последовательным линиям. Обычно используется для посылки IP пакетов в Интернет, но может также использоваться с другими протоколами, такими как протокол Novell IPX.

RAID

Redundant Array of Independent Disks (Избыточный Массив Независимых Дисков). Проект запущен департаментом компьютерных наук университета Berkeley. В этом проекте сохранение данных происходит на массиве дисков по различным схемам. Сперва он был использован для дискет, откуда произошел акроним *Redundant Array of Inexpensive Disks* (Избыточный Массив Недорогих Дисков).

RAM

Random Access Memory (Оперативная память) . Термин используется для обозначения главной памяти компьютера. "Random" здесь обозначает то, что любая часть памяти может быть непосредственно доступна.

RFC

Request For Comments (Запрос комментариев) . RFC это официальные Интернет-документы стандартов, опубликованные IETF (*Internet Engineering Task Force*). В них описаны все протоколы, их использование, требования и так далее. Если вам нужно узнать, как работает какой-либо протокол, обращайтесь к соответствующему RFC.

RPM

Red Hat Package Manager (менеджер пакетов Red Hat) . Формат паковки, разработанный в **Red Hat** для создания пакетов программного обеспечения. Используется во многих дистрибутивах *GNU/Linux*, в том числе и в **Mandrake Linux**.

SCSI

Small Computers System Interface (Интерфейс небольших компьютерных систем) . Шина с высокой производительностью, разработанная для подсоединения некоторых типов периферии. В отличии от IDE, шина SCSI не ограничена скоростью приема команд устройством. Только высокопроизводительные машины имею интегрированную в материнскую плату шину SCSI, а остальным компьютерам требуется для этого дополнительные карты.

SMB

Server Message Block (блок серверных сообщений). Протокол, используемый машинами *Windows* (*9x* или *NT*) для совместного использования файлов и принтеров в сети.
See Also: CIFS.

SMTP

Simple Mail Transfer Protocol (Протокол простой передачи почты). Это обычный протокол передачи электронной почты (email). Агенты передачи почты (Mail Transfer Agents), такие как *sendmail* или *postfix* используют SMTP. Иногда их называют SMTP серверами.

SVGA

Super Video Graphics Array (массив супер видео графики) . Стандарт видео дисплея, определенный VESA для архитектуры *персональных компьютеров* Его разрешение 800x 600 x 16 цветов.

TCP

Transmission Control Protocol (Протокол контроля за передачей данных) . Это обычный и надежный протокол, который использует IP для передачи пакетов в сети. TCP добавляет необходимые проверки в IP для того, чтобы удостовериться, что пакеты доставлены. В отличии от UDP, работает в режиме соединения, что означает, что две машины должны установить соединение прежде чем обмениваться данными.

URL

Uniform Resource Locator (унифицированный указатель информационного ресурса) . Строка специального формата, используемая для уникальной идентификации ресурса в сети Интернет. Ресурс может быть файлом, сервером или чем-то другим. Синтаксис URL такой:
`protocol://server.name[:port]/path/to/resource` (протокол`://`сервер`.`имя`[:`порт`]/`путь`/`к`/`ресурсу).
Когда указывается только имя машины и протокол `http://`, то по умолчанию запрашивается с сервера файл `index.html`.

VESA

Video Electronics Standards Association (Ассоциация стандартов видео электроники). Ассоциация промышленных стандартов для архитектуры *персональных компьютеров*. Например, она является автором стандарта SVGA.

WAN

Wide Area Network (глобальная сеть) . Это сеть, которая хотя и похожа на LAN, но соединяет компьютеры, которые физически разнесены на большие расстояния и не соединяются общим кабелем.

account

в системах *`UNIX`* это комбинация имени, персонального каталога, пароля и *`shell`*, которая позволяет человеку подключаться к данной системе.

alias

механизм, используемый в *`shell`* для замены одной строки другой перед выполнением команды. Все, определенные в текущем сеансе alias-ы, вы можете посмотреть, написав `alias` в командном запросе.

arp

Address Resolution Protocol (протокол разрешения адресов) . Интернет протокол, используемый для преобразования Интернет адреса в физический (на уровне оборудования) адрес в локальной сети. Его использование ограничено сетью, которая поддерживает широковещательные запросы на аппаратном уровне (hardware broadcasting).

assembly language

(язык ассемблера) это язык программирования, наиболее близкий к компьютеру, и является так называемым языком программирования “низкого уровня (low level)”. Преимущество ассемблера состоит в скорости, потому что программам на ассемблере для создания выполняемого кода требуется немного или вообще не требуется трансляции в инструкции процессора. Его главный недостаток в том, что он зависим от процессора (архитектуры). Написание полноценной программы весьма трудоемко. Таким образом, ассемблер самый быстрый язык программирования, но не портируется с одной архитектуры на другую.

ATAPI

(“AT Attachment Packet Interface”) Расширение спецификации ATA (“Advanced Technology Attachment”, более известной как IDE, *Integrated Drive Electronics*), которое предоставляет дополнительные команды для управления устройствами CD-ROM и накопителями на магнитной ленте. IDE контроллеры, оснащенные этим расширением, также называются контроллерами EIDE (*Enhanced IDE*).

ATM

Это акроним для **Asynchronous Transfer Mode** (асинхронный режим передачи). Сеть ATM формирует из данных блоки стандартных размеров (53 байта: 48 для данных и 5 для заголовка), что позволяет эффективно передавать их из точки в точку. ATM это сетевая технология коммутации пакетов, ориентированная на высокоскоростные (мульти-мегабитные) оптические сети.

atomic

набор операций считается атомарным, когда все эти операции выполняются в один момент и не могут быть выгружены.

background
в контексте *shell*, процесс выполняется в фоновом режиме (background), если вы можете вводить команды, которые принимаются процессом во время его выполнения.
See Also: job, foreground.

backup
это означает сохранение ваших важных данных на безопасный носитель или в безопасное место. Backup (резервная копия или процесс создания резервной копии) должен производиться регулярно, особенно это касается критической информации и конфигурационных файлов (наиболее важные каталоги для backup-а это `/etc`, `/home` и `/usr/local`). Обычно для создания резервных копий каталогов и файлов используют `tar` с `gzip` или `bzip2`. Вы можете использовать эти утилиты или другие программы, например `dump` и `restore`, а также многие другие свободные или коммерческие решения по организации резервного копирования.

batch
режим выполнения, когда задания принимаются процессором, который затем выполняет их последовательно, пока последнее задание не будет закончено (пакетный режим).

beep
это небольшой шум, исходящий из динамика вашего компьютера, который предупреждает вас о нестандартной ситуации, например при использовании заполнения команд в командной строке, когда есть более одного варианта подстановки. Возможно, другие программы тоже будут производить beep, чтобы дать вам знать о некоторых специфических ситуациях.

beta testing
(beta-тестирование) это название процесса тестирования beta-версии программы. Программы обычно выпускаются в состоянии alpha и beta для тестирования перед окончательным релизом.

bit
(бит) сокращение от *BInary digiT* (двоичное число). Простое число, которое может принимать значения 0 или 1, поэтому вычисления могут производиться с основанием два.

block mode files
(файлы блочного режима) файлы, содержимое которых буферизируется. Все операции чтения/записи для таких файлов производятся через буферы, которые разрешают асинхронную запись на используемое оборудование, а в случае чтения позволяют избегать доступа к диску, если данные уже находятся в буфере.
See Also: buffer, buffer cache, character mode files.

boot
процедура, происходящая при включении компьютера, при которой производится поиск периферийных устройств и загрузка операционной системы.

bootdisk
(загрузочный диск) дискета, содержащая код, необходимый для загрузки операционной системы с жесткого диска (в некоторых случаях можно провести загрузку системы только с дискеты)

bootloader
(загрузчик) программа, которая запускает операционную систему. Многие загрузчики предоставляют возможность загружать более чем одну операционную систему, путем выбора ее в меню. Загрузчики, имеющие такую возможность, например *grub*, очень популярны и полезны для систем, имеющих несколько вариантов загрузки (dual-boot, multi-boot).

buffer
небольшой кусок памяти фиксированного размера, который может быть связан с файлом блочного режима, системной таблицей, процессом и так далее. Всеми буферами управляет буферный кеш.
See Also: buffer cache.

buffer cache
(буферный кеш) критическая часть ядра операционной системы, которая отвечает за поддержание всех буферов в актуальном состоянии, сокращение кэша когда это необходимо, очистку ненужных буферов и т.д.
See Also: buffer.

bug

(баг) нелогичное или непоследовательное поведение программы в особых случаях, или поведение, которое не следует из документации или принятых для программы стандартов. Часто бывает, что новые возможности вносят новые баги в программы. Исторически, этот термин появился во времена перфокарт: баг (реальный жук!) отдыхал в дырке перфокарты, и из-за него программа не работала. Адмирал Grace Hopper, который это обнаружил, воскликнул "It's a bug!" (это жук!), и с тех пор это так и осталось в терминологии. Имейте в виду, что это только одна из многих историй, объясняющих причину появления термина *bug*.

byte

(байт) восемь последовательных бит. Байт при интерпретации в основание десять, может принимать значения от 0 до 255.
See Also: bit.

case

(регистр) если рассматривать применительно к строкам, case это различие между маленькими и большими (заглавными) буквами.

character mode files

(файлы символьного режима) файлы, содержимое которых не буферизируется. По отношению к физическому устройству 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.
See Also: block mode files.

client

программа или компьютер, который периодически обращается к другой программе или компьютеру, чтобы получить или отдать информацию. В случае систем **peer to peer** (равный с равным), таких как SLIP или PPP, клиент принимается за конец, который инициализирует соединение, а удаленный конец, который получает запрос, обозначается как сервер. Клиент является одной из составляющих **системы клиент/сервер**.

A8AB5<0 :;85=B/A5@25@

система или протокол, состоящие из **сервера** и одного или нескольких **клиентов**.

command line

(командная строка) предоставляется в shell и позволяет пользователю непосредственно набирать команды. Также является темой вечных "перебранок" между ее поклонниками и противниками.

command mode

(командный режим) в *`Vi`* или ее аналогах, это состояние программы, при котором нажатие на клавиши не приводит к вставке символов в файл, который редактируется, а производится действие, связанное со специфической клавишей (если в аналоге нет перенастроенных клавиш и вы настроили вашу конфигурацию). Вы можете выйти из этого режима "обратно в режим вставки", набрав одну из команд: **i**, **I**, **a**, **A**, **s**, **S**, **o**, **O**, **c**, **C**, ...

compilation

(компиляция) это процесс транслирования исходного кода из человекочитаемого вида (конечно, при определенном уровне тренировки) и написанного на одном из языков программирования (*`C`*, например) в бинарный файл, который читается машиной.

completion

(заполнение) возможность в *`shell`* автоматически разворачивать подстроку до имени файла, имени пользователя или другого объекта, пока имеется соответствие.

compression

(сжатие, компрессия) способ сокращения файлов или уменьшения количества символов, посланных через коммуникационное соединение. Некоторые программы сжатия файлов включают *`compress`*, `zip`, `gzip`, и `bzip2`.

console

(консоль) так названо то, что обычно называли терминалами. Это были машины (экран плюс клавиатура), связанные с одной большой центральной универсальной ЭВМ (mainframe). На *`PC`* физический терминал это клавиатура и экран.
See Also: virtual console.

cookies

(кукисы) временные файлы, записываемые на локальный жесткий диск удаленным web-сервером. Эти данные позволяют серверам узнавать настройки пользователя, когда он снова заходит на сервер.

datagram

Датаграмма это дискретная порция данных и заголовков, в которых находятся адреса; она является базовым модулем передачи в сети IP. Вам возможно приходилось слышать другое ее название - “пакет”.

dependencies

(зависимости) это этапы компиляции, которые должны быть удовлетворены перед тем, как перейти к следующему этапу компиляции для успешной компиляции программы. Этот термин используется также в том случае, когда вы хотите установить набор программ, зависящий от других программ, которые могут быть установлены или нет в вашей системе. В таком случае вы можете получить сообщение, в котором система сообщает, что ей нужно “удовлетворить зависимости” для продолжения инсталляции.

desktop

(десктоп, рабочий стол) Если вы используете *X Window System*, то рабочий стол это место на экране, где вы работаете и где отображаются ваши окна и иконки. Еще его называют фон (background) и он обычно заполнен каким-либо цветом, градиентом или даже картинкой.
See Also: virtual desktops.

directory

(каталог, директория) Часть файловой системы. Файлы или другие каталоги могут храниться внутри каталога. Иногда имеются под-каталоги (или ветви) внутри каталога. Это часто упоминают как дерево каталога. Если вам нужно увидеть, что находится в другом каталоге, вы должны составить его список или зайти в него. Файлы внутри каталога выглядят как листья дерева, в то время как под-каталоги похожи на ветви. На каталоги распространяются такие же права, как и на файлы, хотя права для каталогов имеют несколько другой смысл. Специальные каталоги `.` и `..` относятся к самому каталогу и родительскому каталогу (верхнего уровня) соответственно.

discrete values

(дискретные значения) это значения, который не являются непрерывными. Собственно, это некоторый “промежуток” между двумя последовательными значениями.

distribution

(дистрибутив или распространение) термин, использующийся для отличия продуктов одного производителя *GNU/Linux* от другого. Дистрибутив включает в себя ядро Linux и утилиты, а также программы инсталляции, программы третьих лиц и, иногда, коммерческое программное обеспечение.

DLCI

DLCI (Data Link Connection Identifier) это уникальный идентификатор подключения в канале данных, который используется для уникального определения виртуального соединения типа точка-точка в сетях Frame Relay. Он обычно назначается поставщиком сети Frame Relay.

echo

вывод на экран символов в том виде, к котором они набираются на клавиатуре (“as is”) , например, в поле имени пользователя. В отличии от этого, в поле для введения пароля обычно показываются символы “*” вместо вводимых вами символов.

editor

(редактор) таким термином обычно называют программы, с помощью которых редактируются текстовые файлы (aka текстовый редактор). Наиболее известные редакторы в *GNU/Linux* это редактор GNU Emacs (*Emacs*) и *UNIX* редактор *Vi*.

email

сокращение от электронная почта (Electronic Mail). Это способ пересылки сообщений в электронном виде между людьми в одной сети. Так же как и обычному письму, для нормальной доставки, email-у необходимо иметь адрес назначения и обратный адрес отправителя. Адрес отправителя должен выглядеть примерно так “отправитель@домен.отправителя”, а получатель должен иметь адрес типа “получатель@домен.получателя.” Email это очень быстрый метод связи, обычно для доставки требуется всего несколько минут, в зависимости от того, в какой точке мира расположены

отправитель и получатель. Чтобы написать email, вам нужен почтовый клиент, например *pine* или mutt (клиенты текстового режима), или GUI клиенты, типа *kmail*.

environment
(окружение) контекст выполнения процесса. Включает всю информацию, которая необходима операционной системе для управления процессом и что требуется процессору для корректного выполнения процесса.
See Also: process.

environment variables
(переменные окружения) часть окружения процесса. Переменные окружения можно непосредственно посмотреть в *shell*.
See Also: process.

escape
в контексте shell , это действие заключения строки в кавычки для того, чтобы предотвратить интерпретацию строки в shell. Например, когда вам нужно использовать пробелы в некоторых командных строках и перенаправить результат в другую команду, вы должны написать первую команду в кавычках ("escape" команда), иначе shell будет интерпретировать строку некорректно и работать не так, как ожидается.

filesystem
(файловая система) схема, используемая для хранения файлов на физическом носителе (жесткий диск, дискета) в упорядоченном виде. Как пример файловых систем можно привести FAT, *GNU/Linux ext2fs*, ISO9660 (используется CD-ROM-ами) и так далее. Пример виртуальной файловой системы это файловая система /proc.

firewall
(файервол, брандмауэр, межсетевой экран) машина или определенная часть оборудования, которая по топологии локальной сети является единственной точкой соприкосновения с внешней сетью и которая фильтрует, контролирует активность на некоторых портах или ограничивает доступ наружу некоторых специфических интерфейсов.

flag
(флаг) это индикатор (обычно бит), который используется для сообщения о некотором состоянии программе. Например, у файловой системы есть, среди прочих, флаг индикации того, что она произвела резервное копирование, таким образом, когда флаг активен, она производит резервное копирование, когда неактивен - нет.

focus
(фокус) состояние окна, при котором оно принимает события от клавиатуры (нажатия клавиш, отпускания клавиш и щелчки по клавишам мыши), если они не перехватываются оконным менеджером.

foreground
(передний план) в контексте shell, процесс на переднем плане это тот, который сейчас выполняется. Вам прийдется ждать, пока процесс не закончится, чтобы снова вводить команды.
See Also: job, background.

Frame Relay
Frame Relay это сетевая технология, идеально подходящая для пропускания неравномерного трафика. Стоимость сети уменьшается за счет количества клиентов Frame Relay, совместно использующих одну и ту же пропускную способность сети и неравномерной нагрузки по времени.

framebuffer
(видеобуфер) это проекция памяти видеокарты (RAM) на адресное пространство машины. Он позволяет приложениям получать доступ к видеокарте без обращения к самой карте. Все высокопроизводительные графические рабочие станции используют видеобуферы.

full-screen
(полный экран) Термин применяется к приложениям, которые занимают всю видимую область экрана.

gateway
(шлюз, гейтвей) линк, соединяющий две IP сети.

globbing
(подстановка) в *shell*, это возможность группировать некоторый набор имен файлов по шаблонам подстановки.
See Also: globbing pattern.

globbing pattern
(шаблон подстановки) строка, составленная из обычных и специальных символов. Специальные символы интерпретируются и раскрываются в *shell*.

hardware address
Это уникальный номер, по которому идентифицируется хост в физической сети на уровне доступа к носителю. Примерами являются **адреса Ethernet** и **адреса AX.25**.

hidden file
(скрытый файл) файл, который не показывается при выполнении команды `ls` без параметров. Имена скрытых файлов начинаются с `.`. Скрытые файлы используются для хранения пользовательских настроек для различных программ. Например, история команд *bash* сохраняется в скрытом файле `.bash_history`.

home directory
(домашний каталог) часто называется "home", это название персонального каталога данного пользователя.
See Also: account.

host
(хост, узел) относится к компьютерам и обычно используется, когда говорят о компьютерах, находящихся в сети.

icon
маленькая картинка (обычно имеет размер 16x 16, 32x 32, 48x 48 и иногда 64x 64 пикселей), которая в графическом окружении представляет документ, файл или программу.

inode
(информационный узел, инод) точка входа, ведущая к содержимому файла в *UNIX*-подобных файловых системах. Инод уникальным образом определяется через число и содержит мета-информацию о файле, который на него ссылается, такую как время доступа к файлу, его тип, его размер, **но не его имя!**

insert mode
(режим вставки) в *Vi* или одном из его клонов, это состояние программы, при котором происходит вставка символов в редактируемый файл при нажатии клавиш (исключая особые случаи типа заполнения сокращения, выравнивания вправо в конце строки,...). Выйти из него можно по нажатию клавиши **Esc** (или **Ctrl-[**).

Internet
Интернет - это огромная сеть, соединяющая компьютеры по всему миру.

IP address
(IP адрес) это числовой адрес, состоящий из четырех частей, по которому определяется ваш компьютер в сети Интернет. IP адреса имеют иерархическую структуру, с верхним уровнем и национальными доменами, доменами, поддоменами, и каждая машина имеет свой адрес. IP адрес выглядит примерно так `192.168.0.1`. Персональный адрес машины может быть статическим или динамическим. Статические IP адреса никогда не меняются, то есть являются постоянными. Динамический IP адрес обозначает, что у вашей машины при каждом новом соединении с сетью будет меняться IP адрес. При использовании dial-up или кабельного модема, пользователям обычно выдаются динамические IP адреса, в то время как DSL и другие высокоскоростные соединения имеют постоянные IP адреса.

ISO 8859
Стандарт, включающий несколько 8-битных расширений для набора символов ASCII. Особенно важным является ISO 8859-1, "Latin Alphabet No. 1" (латинский алфавит номер 1), который широко используется и является *de facto* стандартом ASCII замены.

ISO 8859-1 поддерживает следующие языки: Afrikaans, Basque, Catalan, Danish, Dutch, English, Faroese, Finnish, French, Galician, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Scottish, Spanish, и Swedish.
Заметим, 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. Для этого существует ISO 8859-15 (модификация от ISO 8859-1), в которой сделаны необходимые изменения.
See Also: ASCII.

job
(задание) в контексте *shell*, задание это процесс, выполняемый в фоне. У вас может быть несколько заданий в одной shell и вы можете контролировать независимо все эти задания.
See Also: foreground, background.

kernel
(ядро) это сердце операционной системы. Ядро отвечает за распределение ресурсов и разделение процессов друг от друга. Ядро обрабатывает все низкоуровневые операции, что позволяет программам общаться непосредственно с оборудованием вашего компьютера, управляет буферным кешем и так далее.

kill ring
в *Emacs* это набор текстовых областей, которые были вырезаны или скопированы с момента запуска редактора. Эти текстовые области могут быть вызваны и снова вставлены, а их структура напоминает кольцо (ring).

launch
(запуск) это действие по вызову или старту программы.

library
(библиотека) это набор процедур и функций в бинарном виде, предназначенных для использования программистами в своих программах (при условии, что лицензия на библиотеку это позволяет). Программа, отвечающая за загрузку общедоступных библиотек во время выполнения, называется динамическим линковщиком (dynamic linker).

link
это ссылка на inode в каталоге, которая дает имя inode (файлу). Примерами инодов, которые не имеют ссылок (и следовательно, не имеют имен) могут служить: неименованные трубы (используются в shell), сокеты (сетевые соединения), сетевые устройства и т.д.

linkage
(линковка) это последняя стадия процесса компиляции, смысл которой состоит в линковке вместе всех object файлов, чтобы получить готовый бинарник, а также поиск соответствия неразрешенным символам динамических библиотек (если не требовалась статическая линковка, когда код этих символов будет включаться в исполняемый код).

Linux
UNIX-подобная операционная система, которая работает на различных компьютерах, а также является свободно доступной для всех в плане использования или изменения. Linux (ядро) был написан Linus Torvalds.

login
(логин) имя пользователя для входа в систему *UNIX* и действие по подключению.

lookup table
это таблица, в которой хранятся коды соответствия (или таги, tags) и их значения. Часто это файл данных, используемый программами для получения информации о конкретном элементе.
Например, *HardDrake* использует такую таблицу, чтобы узнать, что обозначает код производителя продукта. Вот одна из строк таблицы, в которой имеется информация о продукте CTL0001

```
CTL0001 sound   sb      Creative Labs   SB16 \
                         HAS_OPL3|HAS_MPU401|HAS_DMA16|HAS_JOYSTICK
```

loopback
(иногда переводят на русский как “возвратнопетлевое” устройство, или “устройство обратной петли”) это виртуальный сетевой интерфейс машины, завернутый сам на себя, Он позволяет выполняющимся

программам не принимать во внимание тот особый случай, когда два сетевых объекта фактически являются одной и той же машиной.

major
специфический номер для класса устройства.

manual page
(страница руководства) небольшой документ, содержащий определение команды и как ее использовать, предназначен для вызова по команде `man`. Это первая вещь, которую нужно использовать при изучении работы команды, с которой вы еще незнакомы.

minor
номер, идентифицирующий определенное устройство, о котором мы говорим.

mount point
(точка монтирования) каталог, с которым связывается раздел или другое устройство файловой системы *GNU/Linux*. Например, ваш CD-ROM монтируется в каталог `/mnt/cdrom`, где вы можете просматривать содержимое любых примонтированных CD-дисков.

mounted
(примонтированный) Так называется устройство, которое связано с файловой системой *GNU/Linux*. Когда вы монтируете устройство, вы можете просматривать его содержимое. Этот термин несколько выходит из употребления при использовании возможности "supermount", при которой пользователям не приходится вручную монтировать съемные устройства.
See Also: mount point.

MSS
Максимальный размер сегмента (Maximum Segment Size (**MSS**)) это самое большое количество данных, которые могут быть переданы за данный промежуток времени. Если вам нужно предотвратить локальную фрагментацию, MSS должен быть равен заголовку MTU-IP (header).

MTU
Максимальный блок передачи данных (Maximum Transmission Unit (**MTU**)) это параметр, определяющий наибольший размер датаграммы (пакета), которую можно пропустить через интерфейс IP, не разбивая на более мелкие блоки. MTU должно быть больше самого большого пакета, который нужно пропустить нефрагментированным. Обратите внимание, что это предотвращает только локальную фрагментацию, некоторые линки на пути могут иметь меньшее значение MTU и на них пакет будет фрагментироваться. Типичное значение MTU для Ethernet интерфейса это 1500 байт, для PPP интерфейса - 576 байт.

multitasking
the ability for (многозадачность) это способность операционной системы разделять процессорное время (CPU time) между несколькими процессами. На низком уровне это также называется мультипрограммирование (multiprogramming). Переключение от одного процесса к другому требует сохранения всего , что относится к текущему процессу и восстановления этих данных при возобновлении работы процесса. Эта операция называется контекстным переключением (context switch) и в процессорах **Intel** это происходит 100 раз в секунду, что производит иллюзию, что операционная система выполняет несколько приложений одновременно. Существует два типа многозадачности: в приоритетной многозадачности система отвечает за увод процессора от одного процесса и перевод его на другой; кооперативная (совместная) многозадачность подразумевает что процесс сам отдает процессор. Первый вариант, очевидно, является лучшим выбором, потому что программа не может занять все процессорное время и блокировать другие процессы. *GNU/Linux* работает по принципу приоритетной многозадачности. Политика, по которой выбирается, какой процесс должен выполняться, зависит от нескольких параметров, и называется scheduling.

multiuser
(многопользовательский) используется для описания операционной системы, которая позволяет многим пользователям входить в систему одновременно, каждый из которых может выполнять свою работу независимо от других. Многозадачная операционная система необходима для обеспечения многопользовательской поддержки. *GNU/Linux* одновременно является многозадачной и многопользовательской операционной системой, как и любая другая *UNIX*.

named pipe
(именованная труба или канал) канал в *UNIX*, который связан (linked), в противоположность каналам, используемым в shell.
See Also: pipe, link.

naming
слово, обычно используемое в компьютерах для метода идентификации объектов. Вы часто будете слышать о "соглашениях об именах" для файлов, функции в программе и так далее.

newsgroups
(группы новостей, телеконференция) места для обсуждений или новостей, в которые можно попасть с помощью клиента новостей или USENET, чтобы прочитать или написать сообщения, относящиеся к теме группы новостей. Например, группа новостей `alt.os.linux.mandrake` это альтернативная группа новостей (alt), имеющая дело с операционной системой (os) *GNU/Linux*, и конкретно с **Mandrake Linux** (mandrake). Так группы новостей разделяются, чтобы проще было искать нужную тему.

null, character
(нуль-символ) символ или байт с номером 0. Используется для отметки конца строки.

object code
(объектный код) это код, генерируемый процессом компиляции для того, чтобы быть слинкованным (связанным) с другими объектными кодами и библиотеками для формирования исполняемого файла. Объектный код является машиночитаемым.
See Also: compilation, linkage.

on the fly
(на лету) Что либо делается, так сказать "на-лету", когда делается наряду с чем-то другим, незаметно для вас и явно не спрашивая.

open source
(открытый исходный код) так называется свободно доступный исходный код программ, который сделан доступным для сообщества разработчиков и публике в целом. Теория, лежащая в основе этого, заключается в разрешении использования и изменения исходного кода для широкого круга программистов, что в конечном счете, приведет к тому, что продукт станет более полезным для всех и каждого. Некоторые популярные программы с открытым исходным кодом это *Apache*, *sendmail* и *GNU/Linux*.

operating system
(операционная система) это интерфейс между приложениями и оборудованием. Задачей любой операционной системы является прежде всего управление всеми специфическими ресурсами машины. В *GNU/Linux* это делается ядром и подгружаемыми модулями. Другие широкоизвестные операционные системы это *AmigaOS*, *MacOS*, *FreeBSD*, *OS/2*, *UNIX*, *Windows NT*, и *Windows 9x*.

owner
(владелец) в контексте пользователей и их файлов, владелец файла это пользователь, который создал данный файл.

owner group
(владелец группы) в контексте групп и их файлов, владелец группы для файла это группа, к которой принадлежит пользователь, создавший данный файл.

pager
(пейджер) программа, отображающая один экран файла за раз, что упрощает перемещение вперед и назад по файлу и поиск строк в файле. Мы советуем использовать `less`.

password
(пароль) секретное слово или комбинация слов или букв, которая используется для защиты чего-либо. Пароли используются совместно с логинами пользователей в многопользовательских операционных системах, на web-сайтах, FTP-сайтах и так далее. Пароль должен быть трудноподбираемой фразой или сочетание букв и цифр и никогда не базироваться на словах из словарей. Защита паролем гарантирует то, что другие люди не смогут войти на ваш компьютер или сайт под вашим логином.

patch, to patch
(патч, патчить, накладывать патч) файл, в котором содержится список исправлений для исходного кода, использующийся для того, чтобы добавить новые возможности, удалить баги или изменить

его согласно каким-либо требованиям или необходимости. Действие состоит в наложении (или приложении) патча к файлу с исходным кодом (или архиву) (aka “patching”).

path

(путь) это присваивание файлов или каталогов к файловой системе. Различные слои пути разделяются слешом или символом '/'. Существуют два типа путей в системах *GNU/Linux*. **Относительный (relative)** путь это положение файла или каталога относительно текущего каталога. **Абсолютный (absolute)** путь это позиция файла или каталога относительно корневого (root) каталога.

pipe

(труба, канал) специальный тип файла в *UNIX*. Одна программа пишет данные в трубу, а другая программа читает данные из трубы на другом конце Трубы или каналы в *UNIX* являются FIFO, то есть данные на конце трубы читаются в порядке их поступления. Очень широко используется в shell. Смотрите также **named pipe (именованный канал)**.

pixmap

это сокращение от “pixel map”. Это еще один способ ссылки на bitmap изображения.

plugin

(плугин) дополнительная программа, используемая для отображения или проигрывания некоторого мультимедийного контента, найденного в web документе. Ее обычно несложно загрузить в ваш браузер, если он пока неспособен отобразить или проиграть такой тип информации.

porting

(портирование) один из двух способов выполнения программы на системе, для которой программа изначально не предназначена. К примеру, чтобы можно было выполнять *Windows* программы под *GNU/Linux* (естественным образом), программа должна сначала быть портирована под *GNU/Linux*.

precedence

(приоритет) диктует порядок вычисления операндов в выражении. Например: если есть `4 + 3 * 2`, вы получите в результате 10, так как умножение имеет больший приоритет, чем сложение. Если вам нужно произвести сначала сложение, тогда вам нужно добавить скобки, так, как показано здесь: `(4 + 3) * 2`. Если вы сделаете так, вы получите в результате 14, так как скобки имеют больший приоритет, чем сложение или умножение, следовательно операции в скобках будут произведены первыми.

preprocessors

(препроцессор) набор директив, которые сообщают компилятору, что нужно заменить эти директивы на код языка программирования, используемого исходным файлом. Примеры препроцессоров языка *C* это `#include`, `#define`, и т.д.

process

(процесс) в контексте операционной системы, процесс это экземпляр программы, выполняющейся в своем окружении.

prompt

(запрос, приглашение) в *shell*, это строка перед курсором. Когда вы видите его, вы можете вводить команды.

protocol

Протоколы организуют процесс передачи информации между различными машинами в сети, используя аппаратное или программное обеспечение. Они определяют формат переданных данных, управляет ли одна машина другой и т.д. Наиболее известные протоколы это HTTP, FTP, TCP и UDP.

proxy

(прокси) машина, которая находится между сетью и Интернет, и ее задача состоит в ускорении передачи данных по широко используемым протоколам (например, HTTP и FTP). Она поддерживает кеш предыдущих запросов, поэтому, если машина делает запрос на то, что уже находится в кеше, она получит данные быстрее, потому что она получает информацию из локального кеша. Прокси очень полезны в сетях с небольшой полосой пропускания (типа модемных соединений). Иногда прокси является единственной машиной, которая имеет доступ наружу сети.

pulldown menu

(выпадающее меню) это меню, которое имеет кнопку “прокрутки” в одном из своих углов. Когда вы нажимаете на такую кнопку, меню “выезжает” и показывает вам полное меню.

quota
(квота) это метод ограничения дискового пространства и установки лимитов для пользователей. Администраторы могут ограничивать размер домашних каталогов для пользователей путем установки лимитов квот на определенные файловые системы.

read-only mode
(режим только-для-чтения) для файлов это значит, что его нельзя записать. Вы можете читать содержимое файла, но не можете его изменять.
See Also: read-write mode.

read-write mode
(режим чтение-запись) для файла это обозначает, что файл может быть записан. Вы можете смотреть его содержимое и изменять его.
See Also: read-only mode.

regular expression
(регулярное выражение) мощное теоретическое средство для поиска соответствий текста в строках. Позволяет определять шаблоны, которым должны повиноваться строки. Множество утилит *UNIX* используют их: `sed`, *awk*, *grep*, *perl* и другие.

root
это суперпользователь любой системы *UNIX*. Обычно root (aka системный администратор) является персоной, ответственной за обслуживание и наблюдение за *UNIX* системой. Он же имеет полный доступ ко всему, что есть в системе.

root directory
(корневой каталог) Это каталог верхнего уровня файловой системы. У него нет родительского каталога, то есть для корневого каталога ссылается на него самого. Корневой каталог обозначается как '/'.

root filesystem
(корневая файловая система) Это файловая система верхнего уровня, в которую *GNU/Linux* монтирует все свое дерево каталогов. Для нее необходимо постоянно размещаться на своем собственном разделе. На ней находится корневой каталог.

route
Это путь, по которому ваши пакеты ходят по сети, прежде чем добраться до цели. Это путь в сети от одной машины к другой.

run level
(уровень выполнения) это конфигурация системного программного обеспечения, которая только позволяет выбранным процессам существовать. Для каждого runlevel в файле `/etc/inittab` определены процессы, которые разрешено запускать. Существует восемь определенных уровней выполнения: 0, 1, 2, 3, 4, 5, 6, S и переключение между ними может производить только привелегированный пользователь, выполняя команды `init` и `telinit`.

script
shell (скрипт) скрипты это последовательности команд для выполнения, так же, как они последовательно выполнялись бы в командной строке. scripts are sequences of commands to be executed as if they were sequentially entered in the console. Скрипты *shell* в *UNIX* чем-то похожи на batch-файлы в *DOS*.

security levels
(уровни безопасности) уникальная возможность **Mandrake Linux**, которая позволяет вам устанавливать различные уровни ограничений согласно тому, как вы хотите обезопасить вашу систему. Существует 6 предустановленных уровней, в пределах от 0 до 5, где 5 это самая надежная защита. Вы также можете определять свои собственные уровни безопасности.

server
(сервер) программа или компьютер, которые предоставляют возможность или сервис и ожидают соединения от **клиентов** для выполнения их запросов или выдачи запрошенной ими информации. В случае систем **peer to peer**, таких как SLIP или PPP, сервер принимается за тот конец, который вызывается, а тот конец, который вызывает принимается за клиента. Сервер является одним из составляющих **системы клиент/сервер**.

shadow passwords
(теневые пароли) набор управления паролями в системах *UNIX*, в котором файлы, содержащие пароли, не являются читаемыми из мира. В системе обычных паролей это не так. Эта система еще предлагает возможность слежения за устареванием паролей.

shell
(оболочка) *shell* это базовый интерфейс к ядру операционной системы. Предоставляет командную строку, в которой пользователи могут выполнять программы и системные команды. Все shell-ы поставляют скриптовые языки, которые могут быть использованы для автоматизации задач или упрощения часто используемых сложных задач. Эти скрипты *shell* похожи на batch файлы в операционной системе *DOS*, но намного мощнее. Примерами оболочек являются *bash*, *sh*, и *tcsh*.

single user
(однопользовательский) используется для описания состояния операционной системы, или даже описания самой операционной системы, при котором входить в систему и использовать ее позволено только одному пользователю.

site dependent
(зависимый от места) обозначает, что информация, используемая такими программами, как *imake* и `make` для компиляции некоторых файлов исходного кода зависит от расположения, архитектуры компьютера, установленных на компьютере библиотек и так далее.

socket
(сокет, штепсель, розетка) тип файла, соответствующий любому сетевому соединению.

soft links
(мягкие ссылки) см. “symbolic links (символические ссылки)”.

standard error
(стандартная ошибка) файловый дескриптор номер 2, открываемый каждым процессом, используемый по соглашению для вывода сообщений об ошибках на экран терминала.
See Also: standard input, standard output.

standard input
(стандартный ввод) файловый дескриптор номер 0, открываемый каждым процессом, используемый по соглашению как файловый дескриптор из которого процессы получают данные.
See Also: standard error, standard output.

standard output
(стандартный вывод) файловый дескриптор номер 0, открываемый каждым процессом, используемый по соглашению как файловый дескриптор, в который процессы направляют свой вывод.
See Also: standard error, standard input.

streamer
(стример) это устройство, которое работает с “потоками” (непрерывными или разделенными на части) символов на входе. Типичный стример это накопитель на магнитной ленте.

switch
(ключ) Ключи используются для управления поведением программ, и еще называются опциями или аргументами командной строки. Чтобы узнать какие опциональные ключи имеет программа и воспользоваться ими, читайте страницы `man` или попробуйте передать программе ключ `--help`(то есть.. `program --help`).

symbolic links
(символические ссылки) это специальные файлы, не содержащие ничего, кроме строки, которая делает ссылку на другой файл. Любое обращение к ним ничем не отличается от обращения к файлу, на чье имя указывает ссылка, который может существовать или не существовать, и путь к которому может быть задан абсолютным или относительным способом.

target
(цель) это объект компиляции, то есть бинарный файл, который должн быть создан компилятором.

telnet
: устанавливает соединение с удаленным хостом и позволяет входить на машину, на которой у вас имеется аккаунт. Telnet наиболее широко используемый метод удаленного логина, однако, в качестве альтернативы, существуют более защищенные и лучшие методы, например `ssh`.

theme-able
: (поддерживающий темы) графическое приложение является поддерживающим темы (theme-able), если оно позволяет в ходе своего выполнения изменять свой внешний вид. Большинство оконных менеджеров поддерживают эту возможность.

traverse
: для каталогов в системе *UNIX* это означает, что пользователю разрешено проходить сквозь каталог, и возможно каталоги, которые в нем содержатся. Для этого требуется, чтобы пользователь имел права выполнения на этот каталог.

username
: (имя пользователя) это имя (или в общем случае слово), которое идентифицирует пользователя в системе. Каждому пользователю присваивается уникальный целый UID (user ID) (пользовательский идентификатор).
See Also: login.

variables
: (переменные) это строки, которые используются в файлах `Makefile` для того, чтобы их заменили на их значения при каждом их появлении. Обычно они определяются в начале файла `Makefile`. Они используются для упрощения `Makefile` и управления деревом файлов исходных кодов.
Более обобщенно, переменные в программировании это слова, которые указывают на другие сущности (числа, строки, таблицы и т.д.), вероятнее всего изменяемые во время выполнения программы.

verbose
: (многословный) Для команд verbose режим означает что команда сообщает на стандартный вывод (или возможно стандартный поток ошибок) о всех действиях, которые она выполняет и о результатах этих действий. Иногда у команд есть способ определения "уровня verbosity", что означает, что результат выводимой командой информации может контролироваться.

virtual console
: (виртуальная консоль) так называется то, что обычно называют терминалами. В системах *GNU/Linux* у вас имеется то, что называют виртуальными консолями, что дает вам возможность использовать один экран или монитор для множества независимо запущенных сеансов. По умолчание, у вас есть шесть виртуальных консолей, которые можно вызвать по нажатию клавиш от **ALT-F1** до **ALT-F6**. Существует седьмая виртуальная консоль , **ALT-F7**, которая разрешает вам зайти в запущенные *X Window System*. В X в текстовые консоли можно попасть, нажимая от **CTRL-ALT-F1** до **CTRL-ALT-F6**.
See Also: console.

virtual desktops
: (виртуальные рабочие столы) В *X Window System* оконный менеджер предоставляет вам несколько рабочих столов. Эта удобная возможность позволяет вам организовать ваши окна, избегая проблем, связанных с наложением десятков окон одно поверх другого. Это работает так, как будто у вас есть несколько экранов. Вы можете переключаться из одного виртуального рабочего стола в другой так, как это задумано в оконном менеджере, который вы используете.
See Also: window manager, desktop.

wildcard
: (знак подстановки) Символы '*' и '?' используются как знаки подстановки и могут представлять что угодно. Символ '*' представляет любое число символов, включая отсутствие символов. Символ '?' представляет только один символ. Знаки подстановки часто используются в регулярных выражениях.

window
: (окно) касаемо сетей, **window** это наибольшая величина данных, которую получающий конец может принять за данный промежуток времени.

window manager
: (оконный менеджер) программа, отвечающая за "внешний вид (look and feel)" графического окружения, имеющая дело с оконными панелями, фреймами, кнопками, корневыми меню и некоторыми сочетаниями горячих клавиш. Без оконного менеджера трудно или вообще невозможно себе представить виртуальные рабочие столы, изменения окон на лету, перемещение окон, ... и т.д.

Предметный указатель