# Руководство пользователя Linux

## Larry Greenfield

### Содержание.

# Руководство пользователя Linux

## 1. Введение

### 1.1 Для кого предназначена эта книга

Тот ли вы, кто должен прочесть на эту книгу. Попробуйте ответить на несколько вопросов: Вы уже взяли где-то Linux, поставили его, и хотите знать что делать дальше? Или вы не являетесь пользователем Unix, и хотите узнать, что может вам дать Linux.

Если у Вас есть эта книга, ответы на эти вопросы вероятно "Да". Каждый, у кого есть Linux, свободно-распространяемая разновидность Unix, разработанная Линусом Торвальдсом (Linus Torwalds), на его компьютере, но не знает, что ему делать дальше, должен прочесть эту книгу. Она охватывает большинство основных команд Unix. Мы так же расскажем о GNU Emacs, мощном редакторе, и о некоторых других больших приложениях в Unix'е.

#### 1.1.1 Что вы должны сделать перед чтением этой книги.

Эта книга требует выполнения нескольких условий, которые не находятся во власти автора. Прежде всего, эта книга подразумевает, что Вы имеете доступ к системе Unix. (К сожалению, очень тяжело изучить Unix не имея его.) Более того, эта Unix-система должна быть системой Linux, работающей на Intel PC. Это требование не является обязательным, но когда что-либо зависит от версии Unix, я буду ссылаться на то, как ведет себя Linux.

Имеются различные формы Linux, называемые дистрибутивами. Мы надеемся, что Вы нашли полную поставку Linux, такую как SoftLanding Linux Systems или MCC-Interim, и установили ее. Различные дистрибутивы Linux несколько отличаются, но эти отличия обычно небольшие и несущественные. (Возможно в этой книге Вы встретите места, которые будут немного отличаться от того, что будете видеть Вы. Это вероятнее всего означает, что вы используете дистрибутив, отличный от моего. Автору хотелось бы знать обо всех таких отличиях.)

Если Вы являетесь суперпользователем (администратором или установщиком) системы, Вы должны также зарегистрировать себя в системе как нормального пользователя. Посмотрите руководство по установке системы для того, чтобы узнать, как это сделать. Если Вы не являетесь суперпользователем, то Вы должны попросить его зарегистрировать себя в системе.

У Вас должно быть време и терпение. Изучение Linux не так уж просто - большинство пользователей находят, что научиться использовать операционную систему Macintosh Operating System намного проще. Однако многие считают, что Linux намного мощнее.

Наконец, эта книга предполагает, что Вы более-менее знакомы с некоторыми компьютерными терминами и понятиями. Хотя это требование не столь важно, это делает чтение книги более простым. Вы должны знать, что означают такие слова, как "программа" и "исполнение". Если это не

так, то Вам может понадобится чья-нибудь помощь при изучении Unix.

## 1.2 Как избежать чтения этой книги

Лучший способ изучить почти любую компьютерную программу - это установить ее на своем компьютере. Большинство людей считают, что чтение книги без использования программы не очень то полезно. Точно также, способ научиться Unix и Linux - это использовать их. Используйте Linux для всего, что возможно. Экспериментируйте. Не бойтесь - возможно, Вы и потеряете что-либо, но вы всегда сможете это повторно инсталлировать. Делайте резервные копии и наслаждайтесь!

Хорошо это или плохо, но Unix не является такой же интуитивно понятной, как другие операционные системы. Поэтому, Вы вероятно все же прочтете по крайней мере две первые главы этой книги.

Способ номер один избежать чтения этой книги состоит в использовании доступной электронной документации. Научитесь использовать команду man - она описана в разделе 4.2.

## 1.3 Как читать эту книгу

Наиболее предпочтительный способ изучения Unix состоит в чередовании чтения книги и попыток проиграть прочитанное на системе. Играйтесь до тех пор, пока все понятия не станут для Вас привычными, а затем начинайте прыгать по всей книге. Вы увидите, что книга охватывает разные разделы, некоторые из которых могут Вас заинтересовать. После этого, Вы должны стать достаточно уверенным, чтобы начать использовать команды, не зная заранее что они должны делать.

То, что многие люди называют Unix - это оболочка Unix (shell), специальная программа, которая интерпретирует команды. Вообще, это хороший взгляд на вещи, но Вы должны знать, что Unix в действительности состоит из много большего (или много меньшего - в зависимости от точки зрения). Эта книга рассказывает о том, как использовать оболочку, программы, которые обычно поставляются с Unix, и некоторые программы, которые не всегда поставляются с Unix.

Эта глава - это мета-глава, в ней говорится об этой книге и о том, как использовать эту книгу в работе. Оставшиеся главы содержат:

Глава 2 обсуждает, как Unix и Linux появились, и в каком направлении они развиваются. В ней так же говорится о Фонде Свободного Программного Обеспечения (Free Software Foundation) и проекте GNU.

Глава 3 рассказывает о том, как запустить и остановить Ваш компьютер, и что происходит в это время. Многое из того, о чем говорится в этой главе, не нужно для использования Linux, но это довольно полезно и интересно знать.

Глава 4 является введением в оболочку Unix. Это та среда, в которой люди работают и запускают программы. Здесь говорится о основных программах и командах, которые Вы должны знать для использования Unix.

Глава 5 охватывает все, что относится к системе X Window System. X - это основной графический интерфейс для Unix, и некоторые дистрибутивы устанавливаются по умолчанию.

Глава 6 охватывает некоторые из более глубоких понятий Unix-оболочки. Изучение приемов, описываемых в этой главе, позволяет более эффективно использовать Unix.

Глава 7 содержит короткие описания различных Unix-команд. Чем большее количество инструментов умеет использовать пользователь, тем быстрее он будет выполнять свою работу.

Глава 8 описывает текстовый редактор Emacs. Emacs - это очень большая программа, которая объединяет многие инструменты Unix общим интерфейсом.

Глава 11 описывает некоторые большие и более сложные в использовании программы.

Глава 12 рассказывает о том, как избежать ошибок при использовании Unix и Linux.

### 1.4 Документация Linux.

Эта книга, Руководство пользователя Linux, предназначена для начинающих пользователей Linux. Однако, имеются книги и для более опытных пользователей.

#### 1.4.1 Другие книги о Linux

Остальные книги включают "Установка и начало использования", руководство о том, как получить и установить Linux, "Руководство Администратора Системы Linux", рассказывающее о том, как организовать и поддерживать работу системы Linux, и "Руководство по Изменению Ядра", книга о том, как модифицировать Linux. "Руководство по Сетевому Администрированию Linux" рассказывает о том, как установить, сконфигурировать и использовать сеть.

#### 1.4.2 HOWTO

Дополнительно к книгам, документация Linux включает набор небольших документов, описывающих как установить и использовать различные аспекты Linux. Например, SCSI-HOWTO описывает некоторые трудности в использовании совместрно с Linux SCSI - стандартного интерфейса общения с устройствами.

Документы HOWTO доступны в нескольких форматах.

#### 1.4.3 Проект Linux-документации

Как и почти все, что связано с Linux, Проект Linux-документации - это группа людей, работающих по всему миру. Изначально будучи организованный Ларсом Виржениусом (Lars Wirzenius), Проект в настоящее время координируется Маттом Вельшом (Matt Welsh) с помощью Майкла Джонсона (Michael K. Johnson).

Есть надежда, что Проект Linux-документации предоставит со временем все книги, необходимые для документирования Linux. Пожалуйста, сообщите нам о том, что нам удалось, и о том, что нам следует сделать. Вы можете связаться с автором по адресу

greenfie@gauss.rutgers.edu и с Маттом Вельшом по адресу mdw@cs.cornell.edu.

### 1.5 Операционные системы

Основное предназначение операционной системы состоит в поддержке исполнения программ, в работе которых Вы заинтересованы. Например, Вы можете использовать редактор для создания документа. Этот редактор не сможет работать без помощи операционной системы - ему нужна эта помощь для взаимодействия с Вашим терминалом, Вашими файлами, и т.д.

Если от операционной системы требуется только поддержка Ваших приложений, почему же требуется целая книга, рассказывающая об операционной системе? Существует множество вспомогательных действий, которые Вам приходится выполнять. В случае Linux, операционная система так же содержит несколько мини-приложений, помогающих Вам более

эффективно работать. Знание операционной системы может быть полезно, когда Вы не работаете в одном громадном приложении.

Операционные системы (ОС) могут быть простыми и минимальными, такие как DOS, или большими и сложными, как OS/2 или VMS. (Каждая из этих систем имеет свои плюсы.) Unix претендует на место "золотой середины". Предоставляя больше ресурсов и делая больше, чем ранние операционные системы, он не пытается делать ВСЕ, как некоторые другие операционные системы.

Исходная философия для разработки Unix состоит в распределении функциональности по нескольким маленьким частям, программам. Изначально это было требованием, исходящим из аппаратуры, на которой Unix изначально работал. По какой-то странной причине, получившаяся операционная система оказалось весьма полезной на другой аппаратуре. Вы можете относительно просто достичь новой функциональности и новых возможностей, объединяя маленькие части (программы) новым способом. Если появляются новые утилиты (так и происходит), Вы можете встроить его в Ваш старый инструментарий. К сожалению, в наше время программы для Unix становятся все большими, и включают в себя все больше возможностей, но некоторая гибкость и возможность взаимодействия по прежнему остается. К примеру, когда я писал этот документ, я активно использовал эти программы; fvwm - для управления "окнами", emacs для редактирования текста, LaTeX - для форматирования его, xdvi для просмотра отформатированного текста, dvips - для подготовки его к печати, и, наконец lpr для печати. Если я завтра найду новую лучшую программу просмотра dvi, я смогу использовать ее вместо старой, не изменяя остальных установок.

Когда Вы используете операционную систему, Вы хотите минимизировать количество работы, которую Вам необходимо выполнить для достижения цели. Unix предоставляет несколько инструментов, которые могут помочь Вам, но только в том случае, если Вы знаете, что эти инструменты позволяют делать. Потратить час пытаясь сделать что-либо, и наконец сделать это - не очень продуктивно. Хочется надеяться, что Вы уже знаете, как правильно использовать соответствующие инструменты - в этом случае Вы не станете использовать молоток, чтобы затянуть винт.

Основная часть операционной системы называется "ядро". Во многих опреационных системах, таких как Unix, OS/2 или VMS, ядро предоставляет функции, которые используются исполняемыми программами, и планирует исполнение этих программ. Проще говоря, программа A может получить столько-то времени процессора, программа B - столько-то,

и т.д. Одна школа говорит, что ядра должны быть очень маленькими, и не предоставлять много ресурсов программам. Это позволяет ядру быть маленьким и быстрым, но может сделать программы большими. Ядра, разработанные таким образом, называются микроядрами. Другая группа людей считает, что ядро, предоставляющее больше сервиса приложениям, лучше и делает операционную систему лучше. Большинство версий Unix, включая Linux, разработаны исходя из такого предположения. Хотя на первый взгляд все микро-ядра должны быть меньше, чем все макро-ядра, термины микро и макро на самом деле связаны не с размером ядра, а с философией, лежащей в основе разработки операционной системы.

## 2. Что же такое UNIX?

### 2.1 История UNIX'а

В 1965 году Bell Telephone Laboratories (Bell Labs, отдел AT&T) совместно с General Electric и проектом MAC MIT занимались созданием новой операционной системы, названной Multix. Не вдаваясь в подробности, скажем только, что Bell Labs решили не принимать больше участия в этом проекте и вышли из группы. Таким образом, они остались без операционной системы.

Кен Томпсон и Деннис Ритчи решили набросать эскиз операционной системы, которая удовлетворяла бы нужды Bell Labs. Когда Томпсону в 1970 году понадобилась среда разработки для PDP-7, он воплотил в жизнь их идеи. В противовес Multix'у Брайан Керниган дал своей системе имя UNIX.

Позднее Деннис Ритчи разработал язык программирования C. В 1973 году UNIX был переписан на C, что дало мощный толчок к дальнейшему. В 1977 г. UNIX был перенесен с PDP на новую машину, именно благодаря этому.

Постепенно UNIX стал популярным. Сегодняшний UNIX весьма отличается от UNIX'а семидесятых. Существовало две основных его версии: System 5, созданная в UNIX System Laboratories (USL), филиале Novell, и версия BSD, Berkeley Software Distribution. Версия USL сейчас имеет хождение в своем четвертом релизе, SVR4, а последняя версия BSD имеет номер 4.4. Однако, кроме этих двух существует множество других версий. Большинство версий UNIX'а было созданы компаниями-разработчиками программного обеспечения и по большому счету могут быть отнесены к одной из двух групп (в основе которых лежат две версии, о которых говорилось выше). Недавно появились версии UNIX'а, объединяющие в себе свойства обоих групп.

В наше время UNIX стал более коммерческим, чем в былые дни, и лицензия на его использование весьма дорога. Новые версии UNIX'а для Intel PC стоят от 500 до 2000 долларов.

### 2.2 История Linux

Linux изначально был написан Линасом Торвальдсом, а затем улучшался бесчисленным количеством народа во всем мире. Он является клоном операционной системы UNIX. Ни USL, ни Университет Беркли не участвовали в его создании. Один из наиболее интересных фактов из истории Linux'а - это то, что в его создании принимали участие одновременно люди со всех концов света - от Австралии до Финляндии - и продолжают это делать до сих пор.

Вначале Linux разрабатывался для работы на 386 процессоре. Одним из первых проектов Линаса Торвальдса была программа, которая могла

переключаться между процессами, один из которых печатал AAAA, а другой - BBBB. Впоследствии эта программа выросла в Linux.

Linux поддерживает большую часть популярного UNIX'овского программного обеспечения, включая систему X Window. Это довольно большая программа, разработанная в Массачусетском Технологическом институте, позволяющая компьютерам создавать графические окна и используемая на многих различных UNIX-овских платформах. Linux по большей части совместим с System 5 и с BSD и удовлетворяет требованиям POSIX-1 (документа, пытающегося стандартизировать операционные

системы). Linux также во многом согласуется с POSIX-2, другим документом IEEE по стандартизации операционных систем. Он является смешением всех трех стандартов: BSD, System 5 и POSIX.

Большинство утилит, включаемых в дистрибутивы Linux'а получены от Free Software Foundation как часть проекта GNU. Проект GNU - это попытка написать переносимую продвинутую операционную систему, которая будет выглядеть также, как UNIX. Слово "переносимая" означает, что она будет работать на различных машинах, а не только на Intel PC, Macintosh или какой-нибудь еще. Linux тяжело переносится на другие компьютерные архитектуры, потому, что писался с расчетом на 80386.

### 2.2.1 Linux сегодняшнего дня

Развитие Linux'а разделилось на две ветви. Первая, с номерами версий, начинающимися с 1.0, считается более стабильной, надежной версией Linux'а. Вторая, чьи версии нумеруются 1.1, является более дерзкой и быстрее развивающейся и, следовательно (к сожалению), более богатой ошибками.

В данный момент изменения в Linux'е касаются поддержки TCP/IP и борьбы с ошибками. Linux - это достаточно большая система, но, к сожалению, содержит ошибки, которые находятся и исправляются. Хотя некоторые люди все еще регулярно сталкиваются с ошибками, как правило, это происходит из-за того, что они используют нестандартную или неадекватную аппаратуру. Очевидных ошибок становится все меньше, и они встречаются все реже.

Конечно, все это касается только ошибок, найденных в ядре. Проблеиы могут встречаться в любой части системы, и неопытные пользователи не могут определить, в какой программе случился сбой. Например, компьютер выдает нечто непонятное, - что это - ошибка или правильный результат? Предположим, что это все-таки правильный результат, тогда чем он вызван - последней командой или чем либо еще? Надеюсь, эта книга поможет разобраться в различных ситуациях.

### 2.2.2 Тривиальнейшая вещь

Перед тем, как мы пустимся в наше путешествие, позвольте заострить ваше внимание на чрезвычайно важном моменте:

Вопрос: как правильно произносится "Linux"

Ответ: По Линасу Торвальдсу он должен произноситься с короткой i, как в словах "prInt", "mInimal" и т.д. Linux должен рифмоваться с Minix-ом, другим клоном UNIX'а.

### 2.2.3 Коммерческое программное обеспечение в Linux'е

Хорошо это, или плохо, но в настоящее время для Linux'а имеется коммерческое программное обеспечение. Хотя Motif и не является фантастическим средством подготовки текстов, это пакет должен быть куплен, и исходные тексты для него не поставляются. Motif - это пользовательский интерфейс для системы X Window System, смутно напоминающий Microsoft Windows.

Читатели, интересующиеся законностью использования Linux, могут обратится к лицензии Linux. В то время как Общая Лицензия GNU (GNU General Public License), приведенная в приложении A, применима к ядру Linux, Общая Библиотечная Лицензия GNU (GNU Library General Public License, текст которой приведен в приложении B, применима к большей части кода приложений, выполняемых в Linux.

# 3. НАЧИНАЕМ

## 3.1 Начинаем работу с компьютером

До прочтения этой книги Вы могли иметь опыт работы в MS-DOS или другой однопользовательской операционной системе, такой как OS/2 или Macintosh. В таких операционных системах вам не нужно было проходить процедуру идентификации перед началом работы с компьютером;предполагалось, что вы были единственным пользователем системы, и имели доступ ко всему. Unix - многопользовательская операционная система - ее могут использовать одновременно несколько человек, и разные люди порой обслуживаютя по-разному.

Для того, чтобы раыличать людей, Unix требует, чтобы пользователь идентифицировал себя с помощью процесса, называемого loggin in. Когда вы впервые включаете компьютер, происходит несколько вещей. Поскольку это руководство составлено для Linux, я расскажу вам что происходит во время загрузочной последовательности Linux'а.

Обратите внимание, что при использования Linux'а на некоторых типах компьютеров, отличных от Intel PC, кое-какие вещи в этой главе будут неприменимы. (В основном, из разделов 3.1.1 и 3.1.2.)

### 3.1.1 Включение компьютера

В самом начале, когда вы включаете Intel PC, загружается BIOS. BIOS установливается для поддержки основной системы ввода/вывода. Это программа, постоянно хранящаяся в компьютере в чипах доступных только для чтения. BIOS никогда не может быть изменен для приспособления к нашим нуждам. Он выполняет несколько тестов, а после этого пытается обратиться к флоппи-диску в первом устройстве. Если это ему удается, то он ищет boot-сектор, и начинает выполнение кода, находящегося там. Если дискета есть, но boot-сектора найти не удалось, BIOS выдаст сообщение типа:

Non-system disk or disk error

Удаление дискеты из дисковода и нажатие клавиши вызовет продолжение процесса загрузки.

Если в дисководе нет дискеты, BIOS ищет Master Boot Record (MBR) на жестком диске, после чего начинается выполнение расположенного там кода, который загружает операционную систему. В Linux-системах, LILO, (LInux LOader - Загрузчик Linux'а) может занимать позицию в MBR и будет загружать Linux. Сейчас мы предполагаем, что это произошло, и Linux начал загружаться. ( Ваш конкретный дистрибутив может управлять загрузкой с жесткого диска по-разному. Посмотрите документацию к нему. Можете также почитать документация к LILO,[1].)

### 3.1.2. Linux принимается за дело

Перед чтением этого раздела, вам следует знать, что в нем нет ничего действительно необходимого для работы с Linux'ом. Эту главу можете прочесть ради удовольствия или интереса, но если она покажется вам скучной или слишком технической, пропустите!

После того, как BIOS передал управление LILO, LILO передает управление Linux. (Подразумевается, что вы сконфигурировали Linux для загрузки по умолчанию. Также возможно, что LILO вызывает DOS или другую операционную систему для PC.) Первое, что делает Linux, когда начинает выполняться, это переходит в защищенный режим. 386 или 486 процессор, управляющий вашим компьютером, имеет два режима (для наших целей), называемые реальным и защищенным режимами. DOS работает в реальном режиме, также как и BIOS. Тем не менее, для более продвинутых операционных систем, необходимо работать в защищенном режиме. Поэтому, когда Linux загружается, он отказывается от BIOS.

Затем Linux проверяет тип аппаратуры, на которой он исполняется. Его интересует знать тип вашего жесткого диска, есть ли у вас мышь, в сети ли вы, и т.д. Linux не не хранит эту информацию от загрузки к загрузке, поэтому вынужден задавать эти вопросы при каждом запуске. К счастью, он задает эти вопросы не вам, а аппаратуре! Во время загрузки, ядро Linux будет печатать вариации некоторых сообщений. Вы сможете прочитать об этих сообщениях в разделе 3.3.

Ядро только управляет другими программами, так что, если все хорошо, оно должно запустить другую программу, чтобы сделать что-нибудь полезное. Программа, которую запускает ядро, называется init. (Заметьте различие в шрифтах. Слова в таком шрифте - обычно названия программ, файлов, директорий, или других элементов, связанных с компьютером.) После апуска init, ядро больше не запускает программ. Оно становится менеджером и поставщиком, но не активной программой.

Теперь, чтобы увидеть, что делает компьютер после загрузки ядра, мы должны иследовать init. init выполняет последовательность инициализации, различную для разных компьютеров. В Linux'е существует много разных init, и каждая делает инициализацию по-своему. Имеет немаловажное значение, включен ли ваш компьютер в сеть, и какой дистрибутив вы использовали для инсталляции Linux. Когда начинает исполняться init, могут происходить следующие вещи:

- Может быть проверена файловая система. Возможно, вас

интересует, что это такое? Файловая система - это расположение файлов на жестком диске. Она дает Unix'у возможность узнать, какие части диска уже используются, а какие нет. К сожалению, вследствие разных факторов, вроде отключения питания, то, что файловая система думает об остатке, и то, что происходит там на самом деле, может не совпадать. Специальная программа, которая называется fsck, может обнаруживать такие ситуации и исправлять их.

- Запускаются специальные программы для маршрутизации в сети.

- Могут быть стерты временные файлы, оставленные некоторыми программами.

- Могут быть "подправлены" системные часы. На самом деле это обман, так как Unix, по умолчанию, работает с временем по Гринвичу, а ваши CMOS часы, часы могут быть установлены по местному времени.

После того, как init закончил загрузку, он продолжает свою обычную плановую работу. init можно назвать предком всех процессов в Unix системе. Процесс - это запущенная программа; если какую-либо программу запустить несколько раз, одной программе будет

соответствовать несколько процессов. (Процесс также может быть подпрограммой, но сейчас это не важно.)

В Unix'е процесс, экземпляр программы, создается с помощью системного вызова, предоставляемого ядром, называемого fork. init fork - пара процессов, порождающих сами себя. В вашей Linux системе init запускает несколько экземпляров программы, называемой getty, о которой будет рассказано позднее.

### 3.1.3. Пользователь действует

Этот раздел содержит информацию, которую действительно нужно знать, чтобы работать Linux'ом.

Первое, что вы должны сделать, чтобы использовать Unix-машину, это пройти процедуру идентификации. Этот процесс, называемый "logging in" - способ Unix различения пользователей, которым разрешено использовать систему. Он запрашивает имя пользователя и пароль. Это имя обычно похоже на ваше нормальное имя, вы уже должны были его получить от вашего систеного администратора или создать его сами, если вы системный администратор. (О том, как это сделать, рассказывается в Installation и Getting Started или в Linux System Administrator's Guide.)

После того, как инициализационные процедуры закончились, вы увидите что-нибудь вроде:

```
Welcome to the mousehouse. Please, have some cheese.
mousehouse login:
```

Однако, возможно, система, используемая вами представится по-другому. Вместо скучного алфавитно-цифрового режима, она может использовать графику. Тем не менее, она тоже попросит вас пройти процедуру идентификации, и будет функционировать во многом похоже. Если ваша система действительно графическая, то вы по всей вероятности используете систему X Windows. Это значит, что вы будете работать с оконной системой. В главе 5 будут обсуждены некоторые отличия, возникающие при работе с графическими системами. Однако, вход в

систему будет похожим. Если вы используете X, ищите большую букву X на полях.

Итак - приглашение для login. В этом руководстве, для примеров мы будем использовать фиктивного пользователя larry. Везде, где вы видите larry, подразумевается ваше пользовательское имя. Пользовательские имена часто базируются на реальных; большие, более серьезные Unix-системы используют в качестве пользовательского имени фамилию, или некоторую комбинацию имени и фамилии, или даже числа. Возможные имена для пользователя Larry Greenfield могут быть: larry, greenfie, lgreenfi, lg19.

Между прочим, "имя" машины, на которой я работаю - mousehouse. Возможно, что когда вы инсталлировали Linux, вам было предложено какоенибудь очень остроумное имя. Это не имеет особого значения, но всякий раз, в подобных случаях, я использую mousehouse, или, реже lionsden.

После ввода larry, я вижу следующее:

```
mousehouse login: larry
Password:
```

Это Linux спрашивает ваш пароль. Когда вы вводите ваш пароль, вы не можете видеть, что вы набираете. Набирайте осторожно, его можно стереть, но вы не будете видеть, что вы редактируете. Не набирайте очень медленно, если люди смотрят - они могут узнать ваш пароль. Если вы сделали опечатку, вам будет представлена новая возможность для login'а.

Если вы ввели свои имя и пароль правильно, появится короткое сообщение, называемое сообщением дня. Оно может быть любого содержания - это решает системный администратор. После этого появляется подсказка - приглашение дать следующую команду системе. Она выглядит примерно так:

```
/home/larry#
```

Если вы уже определили, что вы используете X Windows, вы возможно увидите похожее приглашение в "окне" где-то на экране. ("Окно" - это просто прямоугольник.) Чтобы печатать в приглашении, поместите курсор мыши (он возможно выглядит как большое "x") в окно.

### 3.2 Покидаем компьютер

В отличие от многих версий DOS, нельзя просто выключить питание, после того, как вы поработали на компьютере. Также не рекомендуется перезагружать компьютер (с помощью кнопки reset) без некоторых приготовлений. Linux, для улучшения работы, кэширует диск. Это значит, что он временно хранит часть постоянного запоминающего устройства в оперативной памяти. Информация, хранящаяся в памяти и информация, хранящаяся на жестком диске синхронизируются каждые 30 секунд. Для того, чтобы выключить или перезагрузить компьютер, вы должны пройти через процедуру, дающую сигнал прекратить кэшировать дисковую информацию.

Если вы закончили работать с компьютером, но прошли login (т.е. ввели свои имя и пароль), то сперва вы должны выйти. Для этого введите команду logout. Все команды передаются на исполнение нажатием кнопки, на которой написано "Enter" или "Return". До тех пор, пока вы не нажали enter, вы можете стереть то, что вы набрали, и напечатать

снова.

```
/home/larry#logout

Welcome to mousehouse. Please, have some cheese.
mousehouse login:
```

Теперь другой пользователь может войти.

### 3.2.1 Выключение компьютера

Если вы работаете в однопользовательской системе, вы можете пожелать выключить компьютер после того, как вы закончили с ним работать. Чтобы сделать это, вы должны войти в систему под специальным именем root. Это имя системного администратора, который имеет доступ ко всем файлам системы. Если вы хотите выключить компьютер, вы должны получить пароль от системного администратора. (В однопользовательской системе - это вы! Убедитесь, что вы знаете пароль root по умолчанию.) Входите как root:

```
mousehouse login: root
Password:

Linux, version 0.99pl10.
/# shutdown now

********** GET THE SHUTDOWN MESSAGE CORRECT ***********
```

Команда shutdown now готовит систему для перезагрузки или выключения. Ждите сообщения о том, что питание компьютера можно отключить, и затем перезагружайте или выключайте систему. Вы должны пройти через эту процедуру, иначе вы рискуете потерять данные.

Быстрое сообщение для ленивых: альтернатива к login/logout подходу - использовать команду su и return. У вас должны спросить у вас пароль root, и затем дать вам его привилегии. Теперь вы можете скомандовать системе: shutdown.

## 3.3 Сообщения ядра

Сообщения, выдаваемые ядром различаются от машины к машине, и от версии ядра к версии. Версия Linux, которая обсуждается в этом разделе - "0.99.10" . (Пожалуйста, имейте в виду, что это большая книга, а Linux быстро развивается. Номера версий в других разделах могут отличаться от этого. Обычно эти различия не существенны.)

### 3.3.1 Стартовые сообщения

Когда Linux запускается впервые, он выдает много сообщений на экран, которые вы можете не успеть просмотреть. Linux содержит специальный файл, называемый /proc/kmsg, который содержит все эти сообщения для последующего просмотра, и я описал здесь примерную

стартовую последовательность.

- Первое, что делает Unix, это определяет тип ваших видеокарты и монитора, чтобы подобрать подходящий размер шрифта. (Чем меньше шрифт, тем больше может поместиться на экране.) Linux может спросить вас, хотите ли вы специальный шрифт, или выберете из имеющихся.

```
Console: colour EGA+ 80x25, 8 virtual consoles Serial driver version
```

В этом примере, владелец машины решил, что он хочет стандартный, большой шрифт во время компиляции.

- Linux переходит в защищенный режим и запускает драйвер последовательного порта, который задает вопросы об аппаратуре. Драйвер - это часть ядра, которая обычно контролирует переферийные устройства.

```
Serial driver version 3.95 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
tty02 at 0x03e8 (irq = 4) is a 16450
```

Здесь, он нашел 3 последовательных порта. Последовательный порт - это эквивалент DOS COM порта, устройство , обычно используемое модемами и мышами.

Он пытается сообщить, что последовательный порт 0 (COM 1) имеет адрес 0x03f8. Когда он прерывает работу ядра, обычно для того, чтобы сказать, что у него есть данные, он использует IRQ 4. IRQ - это другое общения периферии с програмным обеспечением. Каждый последовательный порт также имеет чип-контроллер. (Обычно 16450-й; другие возможные величины 8250 и 16550. Их различия не обсуждаются в этой книге.)

- Следующим идет драйвер параллельных портов. Параллельный порт обычно подсоединен к принтеру, и названия параллельных портов в Linux начинаются на lp. lp используется для построчного принтера, хотя принтер может быть и лазерным.

```
lp_init: lp0 exists (0), using polling driver
```

Это сообщение говорит о том, что драйвер нашел один параллельный порт, и использует для него стандартный драйвер.

- Ядро Linux'а также печатает информацию, связанную с использованием памяти:

```
Memory: 7296k/8192k available (384k kernel code, 384k reserved,
                               128k data)
```

Это значит, что машина имеет 8 мегабайт памяти. Какая-то часть этой памяти зарезервирована для ядра. Остальная может использоваться программами.

Другой тип памяти называется жестким диском. Это как большая постоянная дискета в вашем компьютере - информация на ней не исчезает, даже когда компьютер выключен.

- Теперь ядро исследует ваши дисководы. В данном примере машина имеет два дисковода. В DOS дисковод "A" это 5 1/4 дюймовый дисковод, и "B" 3 1/2 дюймовый. Linux называет дисковод "A" fd0, а "B" - fd1.

```
Floppy drive(s): fd0 is 1.2M, fd1 is 1.44M
```

```
floppy: FDC version 0x90
```

- А теперь Linux занимается менее необходимыми вещами, такими как сетевая карта. Нижеследующее должно быть описано в Linux Networking Guide, и выходит за пределы компетенции этого документа.

```
SLIP: version 0.7.5 (4 channels): OK
plip.c:v0.04 Mar 19 1993 Donald Becker (becker@super.org)


plip0: using parallel port at 0x3bc, IRQ 5.
plip1: using parallel port at 0x378, IRQ 7.
plip2: using parallel port at 0x278, IRQ 2.
```

## 8390.c:v0.99-10 5/28/93 for 0.99.6+ Donald Becker (becker@super.org)

```
WD80x3 ethercard probe at 0x280: FF FF FF FF FF FF not found (0x7f8).
3c503 probe at 0x280:  not found.
8390 ethercard probe at 0x280 failed.
HP-LAN ethercard probe at 0x280: not found (nothing there).
No ethernet device found.
dl0: D-Link pocket adapter: probe failed at 0x378.
```

- Следующее сообщение вы обычно не видите при загрузке. Linux поддерживает FPU - устройство для работы с числами с плавающей точкой. Если одно из таких устройств плохое, то когда Linux пытается их идентифицировать, машина "ломается", т.е. перестает функционировать. Если это произошло, то вы увидите:

```
You have a bad 386/387 coupling.
```

Или:

```
Math coprocessor using exception 16 error reporting.
```

если вы используете 486DX. Если у вас 386 с 387, вы увидите:

```
Math coprocessor using irq13 error reporting.
```

Если у вас нет никакого математического сопроцессора, вы увидите:

```
What will they see?
```

- Ядро также ищет ваш жесткий диск. Если находит (должно найти), то смотрит, какие разделы на нем присутствуют. Разделы - это логическое деление на диске, используемое для хранения операционных систем, чтобы они не влияли друг на друга. В этом примере, компьютер имеет один жесткий диск (hda) с четырьмя разделами.

```
Partition check:
  hda: hda1 hda2 hda3 hda4
```

- Наконец, Linux монтирует раздел, в котором находится корневая файловая система. Там же распожен Linux. Когда Linux монтирует этот раздел, он делает его доступным для пользователей.

```
VFS: Mounted root (ext filesystem).
```

### 3.3.2 Сообщения времени выполнения

Linux иногда посылает сообщения на экран. Далее приведен список некоторых сообщений и их значений. Иногда эти сообщения показывают, что что-то не в порядке. Некоторые из них - критические, которые значат, что операционная система (и все ваши программы) перестали работать. Когда такие сообщения возникают, запишите их и то, что вы в это время делали, и пошлите их в Linux. См. раздел 12.2.2

К счастью, некоторые из тех сообщений просто информационные - надеюсь, их вы видите значительно чаще!

```
Adding Swap: 10556k swap-space
lp0 on fire
******** OBVIOUSLY INCOMPLETE
```

# 4. Unix Shell

## 4.1 Команды Unix

Когда вы впервые входите в систему, вы видите что-то вроде:

```
/home/larry#
```

Это называется приглашеним. Как следует из названия, этой строчкой UNIX приглашает вас ввести следующую команду. Каждая команда в Unix - это последовательность букв, цифр и символов, но без пробелов. Так, настоящие команды Unix включают в себя mail, cat, и CMU_is_Number-5 . Некоторые символы использовать нельзя - об этом будет рассказано позже. Unix также чувствителен к регистру. Это означает, что cat и Cat - разные команды.

Чуствительность к регистру характерна именно для UNIX'а. Некоторые операционные системы, такие как OS/2 или Windows NT, сохраняют регистр , но не придают ему значения. На практике, Unix редко использует разные регистры. Ситуация с cat и Cat нетипична.

Приглашение выдается специальной программой, называемой shell (оболочка). MS DOS shell называется command.com, и является очень простой, по сравнению со многими Unix-овскими shell-ами. Shell принимает и выполняет команды. Пользователь может также создать свои команды, запрограммировав соответствующую последовательность действий на специальном языке. Такие небольшие программы называются скриптами.

Существуют два основных типов оболочек в Unix'е. Bourne shell'ы, названы в честь их изобретателя, Стивена Барни. Существует много

реализаций этих оболочек. Другой тип shell'ов - C shell'ы (первоначально разработанные Биллом Джоем), также весьма распространены. По традиции, Bourne shell'ы использовались для достижения совместимости, а C shell - для интерактивного использования.

Linux имеет Bourne shell, называемый bash, написанный Free Software Foundation. bash значит Bourne Again Shell, один из многих неудачных

каламбуров в Unix. Это расширенный Bourne shell, который обладает многими свойствами C shell'ы, и работает по умолчанию.

Когда вы впервые регистрируетесь в системе, bash выдает вамподсказку, и вы запускаете вашу первая программу в UNIX'е - bash shell.

### 4.1.1 Типичные команды UNIX'а

Первая команда, которую вам следует знать - cat. Чтобы ей воспользоваться, наберите cat и нажмите Return.

```
/home/larry# cat
```

Если у вас на строке нет ничего, кроме курсора, то вы сделали все правильно. Существует несколько вариантов, которые может напечатать пользователь, и далеко не все будут работать.

- Если вы сделали опечатку в слове cat, то вы увидите примерно следующее:

```
/home/larry# ct
ct: command not found
/home/larry#
```

Таким образом, shell информирует вас, что он не может найти программу с названием ct и выдает новое приглашение для работы. Запомните, Unix - чувствителен к регистру: CAT - будет опечаткой.

- Вы можете поставить несколько пробелов перед командой:

```
/home/larry#     cat
```

Это приведет к желаемому результату и запустит программу cat.

- Вы можете также нажать return просто так. Сделайте это прямо сейчас - это не повлечет за собой абсолютно ничего!

Я предполагаю, что вы сейчас в cat. Надеюсь, вам интересно, что она делает. Вопреки всем надеждам, увы, это не игра. cat - это полезная утилита, которая сначала таковой не кажется. Наберите что-нибудь и нажмите return. Вы увидите:

```
/home/larry\# cat
Help! I'm stuck in a Linux program!
Help! I'm stuck in a Linux program!
```

Может показаться, что cat - это просто эхо (на самом деле, это не совсем так). Иногда это полезно, но не сейчас. Поэтому, давайте выйдем из этой программы и займемся командами, которые имеют более очевидную пользу.

Выполнение многих команд UNIX'а можно завершить, нажав Ctrl-d. Ctrl-d - это символ конца файла, короче EOF. Еще он может рассматриваться как конец текста, в зависимости от книги, которую вы читаете. Я буду называть его концом файла. Это управляющий символ, который говорит UNIX-программам, что вы (или другая программа) закончили ввод данных. Когда cat видит, что вы ничего больше не печатаете, она завершается.

Чтобы освоиться с этой идеей, попробуйте программу sort. Как видно из названия, это сортирующая программа. Если вы наберете пару строк, а затем нажмете Ctrl-d, то она выдаст эти строки в отсортированном порядке. Между прочим, такие типы программ называются фильтрами, потому что они берут текст, фильтруют его и выводят слегка измененным. ( cat - самый простой фильтр, он не изменяет введенного.) Мы еще поговорим о фильтрах позже.

## 4.2 Помоги себе сам

man - команда, которая показывает справочные страницу, в которых можно найти информацию о команде (системном вызове, подпрограмме, формате файлов и т.д.), которую вы указали. Например:

```
/home/larry# man cat



cat(1)                                                    cat(1)

NAME
  cat - Concatenates or displays files

SYNOPSIS
  cat [-benstuvAET] [--number] [--number-nonblank] [--squeeze-blank]
  [--show-nonprinting] [--show-ends] [--show-tabs] [--show-all]
  [--help] [--version] [file...]

DESCRIPTION
  This manual page documents the GNU version of cat ...
```

Здесь около одной полной страницы информации о cat. Попробуйте сделать это. Не ожидайте, что вам все будет понятно. Использование man page предполагает, что вы уже немного знаете UNIX. Когда вы прочитаете страницу, возможно, внизу экрана будет написано --more--, Line 1 или что-то вроде этого. Это приглашение читать дальше, и вы его еще оцените.

Вместо того, чтобы просто позволить тексту прокрутиться, man останавливается после каждой страницы и ждет, что вы будете делать дальше. Если вы хотите продолжать, нажмите Space, и вы перелистнете страницу. Если вы хотите выйти из man page, которую вы читаете, просто нажмите q. Вы попадете назад в приглашение shell, ожидающее вашей следующей команды.

man можно вызывать с различными ключами. Например, вас интересуют все команды, которые имеют отношение к Postscript, управляющему языку принтера от Adobe. Наберите man -k ps или man -k Postscript и вы получите список команд, системных вызовов и других документированных частей UNIX'а, которые содержат `ps' (или `Postscript') в своем названии или коротком описании. Это может быть очень полезно, если вы ищете средство для того, чтобы что-то сделать, но не знаете ни его названия, ни даже того, существует ли оно в природе!

## 4.3 Хранение информации.

Фильтры очень полезны, если вы - опытный пользователь, но с ними связана одна маленькая проблема. Как хранить информацию? Конечно, от вас не ждут, что вы будете все печатать все заново каждый раз, когда вы собираетесь использовать программу. Конечно нет. UNIX предоставляет вам для хранения информации файлы и директории (каталоги). Директория - это как папка: она содержит листы бумаги или файлы. Большая папка может даже содержать другие папки - директории могут быть внутри директорий. В UNIX'е, набор директорий и файлов называется фвйловой системой. Изначально, файловая система состоит из одной директории, называемой кореневой ('root'). Внутри этой директории есть еще директории, внутри которых могут быть файли и даже другие директории. Может быть, а может и не быть ограничение на "глубину" файловой системы. Вы легко можете создать уровней вложенности каталогов.

У каждого файла и у каждого каталога есть имя. У него есть и короткое имя, которое может совпадать с именем другого файла или каталога где-то еще в системе, и длинное уникальное имя. Короткое имя может быть joy, тогда как длинное имя будет /home/larry/joy. Полное имя называется также путем (path). Путь можно расшифровать, как последовательность вложенных каталогов. Например, расшифруем /home/larry/joy:

```
/home/larry/joy
Сначала, мы находимся в корневой директории.
 Это обозначает директорию home. Она находится в корневой директории.
       Это директория larry, внутри home.
            joy лежит внутри  larry.
```

Путь может обозначать директорию или файл, поэтому joy может быть и тем и другим. Все элементы до короткого имени должны быть каталонами.

Поясним это с помощью диаграммы в виде дерева. На рисунке 4.1 изображена типичная Linux-система. Обратите внимание, что диаграмма не полная - полная Linux система имеет больше 8000 файлов! - и показывает только некоторые из стандартных директорий. Таким образом, здесь могут

быть некоторые каталоги, которых нет в вашей системе, и ваша система почти наверняка имеет директории, не указанные здесь.

Рисунок 4.1: Типичное дерево каталогов в UNIX'е (в сокращении)

```
 /ддбдд bin
    цдд dev
    цдд etc
    цдд home ддбдддlarry
    Ё          юдд sam
    цдддlib
    цдд proc
    цдд tmp
    юдд usr ддбдддX386
              цдд bin
              цдд emacs
              цдд etc
              цдд g++-include
              цдд include
```

```
цдддlib
цдддlocalддбдд bin
Ё          цдд emacs
Ё          юдд etc
Ё              lib
цдд man
цдд spool
цдд src дддддлinux
юдд tmp
```

### 4.3.1 Просмотр содержимого каталогов с помощью ls

Теперь вы знаете, существуют файлы и директории, и должет быть какой-то способ работать с ними. Действительно, есть. Команда ls одна из наиболее важных. Она выдает список файлов. Если вы попробуете ее ввести, то увидите:

```
/home/larry# ls
/home/larry#
```

Правильно, вы не увидите ничего. Unix очень лаконичен: он не говорит ничего, даже не "нет файлов", если никаких файлов нет. Таким образом, отсутствие вывода есть способ , которым команда ls говорит, что она не нашла никаких файлов.

Но если я только что сказал, что в системе может быть 8000 и более файлов, то где они? Так вы пришли к концепции "текущей директории". Вы можете видеть в приглашении, что ваша текущая директория /home/larry, где у вас нет никаких файлов. Если вы хотите поолучить список файлов более активной директории, попробуйте корневую:

```
/home/larry# ls /
bin      etc      install  mnt      root     user     var
dev      home     lib      proc     tmp      usr      vmlinux
/home/larry#
```

В команде выше, "ls /", директория - это параметр. Первое слово команды - это имя команды, а все после него - параметры. Некоторые команды имеют специальные параметры, называемые опции или переключатели. Проиллюстрируем это:

```
/home/larry# ls -F /
bin/     etc/     install/  mnt/     root/    user/    var@
dev/     home/    lib/      proc/    tmp/     usr/     vmlinux
/home/larry#
```

-F это опция, которая позволяет вам видеть, какие файлы являются директориями, какие - специальными файлами, какие - программами, а какие - нормальными файлами. Все, что со слэшем - это директории. Позднее мы еще поговорим о свойствах ls. Это удивительно сложная программа!

Сейчас вы должны усвоить две вещи. Во-первых, что делает ls в принципе. Попробуйте посмотреть содержимое еще несколько директорий из

тех, что показаны на рисунке 4.1. Некоторые из них могут оказаться пустыми, а в других вы найдете огромное число файлов. Попробуйте ls и

с опцией -F, и без нее. Например, результат выполнения команды ls /usr/local выглядит как :

```
/home/larry# ls /usr/local
archives  bin       emacs     etc       ka9q      lib       tcl
/home/larry#
```

Вторая вещь, которую вы должны усвоить, более абстрактна. Многие Unix команды похожи на ls. Они могут иметь опции, которые обычно выглядят как один символ после черточки, и параметры. Порой разница между ними не совсем ясна. В отличие от ls, многие команды требуют обязательных параметров и/или опций. Чтобы показать общий вид команды, мы будем использовать следующую форму:

```
ls [-arF] [directory]
```

Это шаблон команды, и вы будете встречать его каждый раз при знакомстве с новой командой. Все, что находится в скобках ( [ и ]) - необязательно.

### 4.3.2 Текущая директория и cd

Работа с каталогами была бы затруднительной, если бы для доступа к ним вам каждый раз пришлось писать полный путь до него. В Unix shells вводится понятие "текущей", "настоящей" или "рабочей" директорией. Ваша система, скорее всего, сконфигурирована так, что в подсказке показывается вашу директорию: /home/larry. Если нет, то введите команду pwd для того, чтобы получить имя текущей рабочей директории.

```
mousehouse>pwd
/home/larry
mousehouse>
```

По-умолчанию многие команды работают с текущей директорией, например ls. Мы можем изменить рабочую директорию, используя команду

cd. Например:

```
/home/larry# cd /home
/home# ls -F
larry/     sam/       shutdown/  steve/     user1/
/home#
```

Шаблон команды:

```
cd [directory]
```

Если вы не напишите директории, ничего не произойдет. В противном случае, cd изменит текущую директорию на указанную. Например:

```
/home# cd
/home/larry# cd /
/# cd home
/home# cd /usr
/usr# cd local/bin
/usr/local/bin#
```

Как видите, cd позволяет вам указывать как полные, так и относительные пути. "Полный" путь начинается с / и указывает все каталоги вплоть до нужного вам. "Относительный" путь привязан к вашей текущей директории. В приведенном выше примере, когда я был в /usr, я

сделал относительное перемещение в local/bin - local это директория в usr, а bin - директория в local!

Существует два параметра-директории, используемого только для указания относительных путей: "." и ".." . "." означает текущий каталог, а директория ".." - родителя. Это директории существуют в каждом каталоге , но не совсем укладываются в концепцию "папок в папках". Даже корневая директория имеет родительскую директорию - она сама свой родитель!

Файл ./chapter-1 - это файл в текущей директории с названием chapter-1. Случается, что для некоторых команд нужно ставить "./", хотя это бывает редко. В большинстве случаев, ./chapter-1 и chapter-1

будут означать одно и тоже.

Директория ".." особенно полезна при восстановлении предшествующего состояния:

```
/usr/local/bin# cd ..
/usr/local# ls -F
archives/  bin/       emacs@     etc/       ka9q/      lib/       tcl@
/usr/local# ls -F ../src
cweb/      linux/     xmris/
/usr/local#
```

В этом примере, я переместился в родительскую директорию, используя cd .., и получил список содержания директории /usr/src из /usr/local, используя ../src. Заметьте, что когда я, будучи в /home/larry, набрал ls -F ../src это не привело к желаемому результату!

Еще одно сокрашение для ленивых пользователей: директория ~/ это ваша домашняя директория:

```
/usr/local# ls -F ~/
/usr/local#
```

Вы можете убедиться, что в вашей домашней директории ничего нет! ~/ станет полезнее, когда мы научимся работать с файлами.

### 4.3.3 Использование mkdir для создания каталогов

Создание собственных каталогов в UNIX'е чрезвычайно просто. Для этого служит команда mkdir. Ее название происходит от make directory.

```
mkdir directory
```

Приведем небольшой пример.

```
/home/larry# ls -F
/home/larry# mkdir report-1993


/home/larry# ls -F
report-1993/
/home/larry# cd report-1993
/home/larry/report-1993#
```

На самом деле, mkdir может иметь больше одного параметра, и вы можете указывать как полный, так и относительный путь; в приведенном выше примере, report-1993 - относительный путь.

```
/home/larry/report-1993# mkdir /home/larry/report-1993/chap1 ~/report-
1993/chap2
/home/larry/report-1993# ls -F
chap1/  chap2/
/home/larry/report-1993#
```

Наконец, существует противоположность mkdir, rmdir (от remove directory). rmdir делает именно то, что вы ожидаете:

```
rmdir directory
```

Вот пример rmdir:

```
/home/larry/report-1993# rmdir chap1 chap3
rmdir: chap3: No such file or directory
/home/larry/report-1993# ls -F
chap2/
/home/larry/report-1993# cd ..
/home/larry# rmdir report-1993
rmdir: report-1993: Directory not empty
/home/larry#
```

Как вы видите, rmdir отказывается стирать несуществующую директорию, также, как и директорию, в которой что-то есть. (Помните, report-1993 содержит в себе поддиректорию chap-2!) Еще один интересный момент в связи с rmdir: что получится, если попробовать стереть вашу текущую директорию? Давайте выясним:

```
/home/larry# cd report-1993
/home/larry/report-1993# ls -F


chap2/
/home/larry/report-1993# rmdir chap2
/home/larry/report-1993# rmdir .
rmdir: .: Operation not permitted
/home/larry/report-1993#
```

Другой момент, который может вас заинтересовать: можно ли стереть родительскую директорию? Впрочем, здесь проблемы нет: родитель текущей директории - не пуст, поэтому не может быть удален!

### 4.4 Перемещение информации

Все эти воображаемые директории очень милы, но в действительности они бесполезны, если у вас негде хранить данные. Творцы UNIX'а прониклись этой проблемой и разрешили ее, дав пользователям "файлы". Мы узнаем побольше о создании и редактировании файлов в следующих главах.

Основные команды для работы с файлами - это cp, mv и rm. Соответственно, они происходят от copy, move и remove.

#### 4.4.1 cp в качестве монаха

cp - очень полезная утилита в UNIX'е и чрезвычайно могущественная. Она позволяет одному человеку за секунду скопировать больше информации, чем средневековый монах мог сделать за год.

Будьте осторожны с cp, если у вас не очень много свободного места на диске. Никому не хочется увидеть Error saving--disk full. cp может также переписывать существующие файлы - об этой опасности я расскажу

позже.

Первый параметр для cp - это копируемый файл, последний - куда его копировать. Вы можете копировать файл под другим именем или в другую директорию. Давайте разберем несколько примеров:

```
/home/larry# ls -F /etc/rc
/etc/rc


/home/larry# cp /etc/rc .
/home/larry# ls -F
rc
/home/larry# cp rc frog
/home/larry# ls -F
frog  rc
/home/larry#
```

Первая команда cp взяла файл /etc/rc, который содержит команды, выполняемые UNIX'ом при загрузке, и скопировала его в мою домашнюю директорию. cp не может стереть файл-источник, поэтому я не сделал ничего, что может навредить системе. Теперь в системе существуют дле копии /etc/rc, обе называются rc, но одна находится в директории /etc, а другая - в /home/larry.

Затем я создал третью копию /etc/rc, написав cp rc frog - теперь есть три копии: /etc/rc, /home/larry/rc и /home/larry/frog. Содержание всех трех файлов одинаково, хотя имена разные.

Приведенный выше пример иллюстрирует два использования команды cp. А есть ли другие? Давайте посмотрим:

- cp может копировать файлы из каталога в каталог, если первый параметр - файл, а второй - директория.

- cp может копировать файл и изменять его имя, если оба параметра - названия файлов. Здесь таится одна опастность cp. Если бы я набрал cp /etc/rc /etc/passwd, то cp нормально создала бы новый файл с содержанием, идентичным rc и назвала его passwd. Однако, если /etc/passwd уже существовал, то cp затерла бы старый файл, не давая возможности спасти его!

- Позвольте привести еще один пример cp:

```
/home/larry# ls -F
frog  rc
/home/larry# mkdir rc_version
/home/larry# cp frog rc rc_version


/home/larry# ls -F
frog        rc        rc_version/
/home/larry# ls -F rc_version
frog  rc
/home/larry#
```

Как я использовал cp здесь? Очевидно, cp может иметь больше двух параметров. Приведенная выше команда скопировала все указанные файлы ( frog и rc) и разместила их в директории rc_version. На деле, команда cp может иметь любое количество параметров, и первые n-1 параметров интерпритируются как копируемые файлы, а последний - как директория, в которую нужно копировать.

Вы не можете переименовать файлы, если вы копируете больше одного за раз - они всегда сохраняют свои короткие имена. Это наталкивает на интересный вопрос. Что если набрать cp rc frog toad, где frog и rc существуют, а toad - не директория? Попробуйте и увидите.

И последнее в этом разделе - как изобразить параметры команды ср? Вы понимаете, что в разных случаях параметры значить две различные вещи. Так что:

ср источник имя_назначения
ср файл1 файл2 ... файлN  директория_назначения

### 4.4.2 Команда rm

Теперь, когда мы научились создавать миллионы файлов с помощью ср ( и поверьте, вы скоро найдете новые пути создания еще большего количества файлов), не вредно узнать, как удалить их. В действительности, это очень просто: нужная вам команда - rm, и работает она так, как вы и ожидали.

Любой файл, являющийся параметром rm удаляется:

rm файл1 файл2 ... файпN

Например:

```
/home/larry# ls -F
frog         rc        rc_version/
/home/larry# rm frog toad rc
rm: toad: No such file or directory
/home/larry# ls -F
rc_version/
/home/larry#
```

Как видите, rm очень недружелюбна. Она не только не спрашивает подтверждения, но и умудряется, даже если командная строка некорректна. Это может быть действительно опасно. Вы видите разницу между следующими двумя командами:

```
/home/larry# ls -F
toad  frog/
/home/larry# ls -F frog
toad
/home/larry# rm frog/toad
/home/larry#
```

и этой

```
 /home/larry# rm frog toad
rm: frog is a directory
/home/larry# ls -F
frog/
/home/larry#
```

Как видите, разница в один символ порождает огромную разницу в результате команды. Жизненно важно проверять командную строку до нажатия return!

### 4.4.3 Перемещение информации

Наконец, еще одна команда, которую надо знать - mv. mv очень похожа на cp, за исключением того, что она удаляет исходный файл после копирования его. Таким образом, она действует как тандем cp и rm вместе. Давайте посмотрим, что мы можем делать:

```
/home/larry# cp /etc/rc .
/home/larry# ls -F
rc
/home/larry# mv rc frog
/home/larry# ls -F
frog
/home/larry# mkdir report
/home/larry# mv frog report
/home/larry# ls -F
report/
/home/larry# ls -F report
frog
/home/larry#
```

Как видите, mv переименует файл если второй параметр - файл. Если второй параметр - директория, mv переместит файл в указанную директорию с тем же именем:

```
mv старое_имя новое_имя
mv файл1 файл2 ... файлN новая_директория
```

Вы должны быть очень осторожны с mv - она не проверяет, существует ли уже файл, и удалит его по пути. Например, если бы в моей директории report уже был фал frog, команда mv frog report удалила бы файл ~/report/frog и заменила его на ~/frog.

На самом деле, существует способ заставить rm,cp и mv запрашивать подтверждение. Это опция -i. Если вы используете синоним, вы можете заставить shell автоматически выполнять rm -i, когда вы набираете rm. Подробнее об этом вы узнаете позже.

## 5. Система X Window.

Эта глава касается только тех, кто использует Систему X Window. Если вы видите экран со множеством разноцветных окон, и курсором, который передвигается только с помощью мыши, вы используете X11 (если вы хотите использовать X11, но она не запускается по умолчанию, смотрите часть 5.4)

### 5.1 Что Такое Система X Window?

Система X Window - это распределенный, графический метод работы изначально разработанный в Массачусетском технологическом институте. Эта система была передана консорциуму из представителей фирм (названному, соответственно, "X консорциум") и в дальнейшем сопровождалась ими.

Каждые несколько лет появляются новые версии Системы X Window (здесь и далее Система X Window будет сокращаться как "X"), называемые

релиз (release). Последней версией является X11R6, или релиз 6. Число 11 в названии X11 - официальный номер версии.

Есть два термина, которые следует знать, когда вы имеете дело с X. Клиент - это X программа. Например, xterm - это клиент, который показывает ваш shell, когда вы входите в систему. Сервер - это программа, которая обслуживает программу-клиент. Например, сервер рисует окно для xterm и взаимодействует с пользователем.

Так как клиент и сервер - две отдельные программы, можно исполнять программу-клиент и программу-сервер на двух физически разделенных машинах. В этом настоящая красота X. В добавление, для поддержки стандартных методов изображения, вы можете запустить программу на удаленной машине (на другом краю земли, если вам хочется) и смотреть результаты работы прямо тут, на вашей рабочей станции.

Третий термин, с которым вам следует познакомиться это оконный менеджер. Оконный менеджер - это специальный клиент, который указывает серверу где располагать окна и обеспечивает пользователю способ

перемещения окон. Сам по себе сервер ничего не делает для пользователя. Он просто посредник между пользователем и клиентом.

### 5.2 Что Находится на Моем Экране?

Когда вы запускаете X, запускается сразу несколько программ. Во-первых, запускается сервер. Во-вторых, обычно запускается несколько клиентов. К сожалению, это не стандартизовано в различных дистрибутивах. Вероятно, что среди этих клиентов есть оконный менеджер, fvwm или twm, окно со строкой приглашения на ввод xterm, и часы xclock.

#### 5.2.1 XClock

Объясню сначала простейшие вещи: xclock действует именно так, как вы думаете. XClock показывает секунды, минуты и часы в небольшом окне.

Сколько бы вы не щелкали мышью и не печатали в окне, это не приведет к никаким результатам - это все, что xclock умеет делать. Так ли это? На самом деле есть различные опции, которые вы можете установить для программы, чтобы заставить ее работать по другому. Например, xclock -digital изобразит на экране цифровые часы. xclock -update 1 создаст секундную стрелку, которая передвигается каждую секунду, а xclock -update 5 создаст секундную стрелку, которая передвигается каждые 5 секунд.

Для дополнительной информации об опциях, смотрите справочное руководство man xclock. Если вы хотите попытаться запустить несколько ваших собственных xclock, возможно вам следует прочитать часть 6.4.

#### 5.2.2 XTerm

Окно со строкой приглашения на ввод (что-то вроде home/larry# или тому подобное) управляется программой xterm. xterm это сложная программа. На первый взгляд кажется, что она не такая уж и замысловатая, но на самом деле это не так. xterm эмулирует терминал так, что обычные текстовые приложения Unix работают корректно.

В большей части этой книги мы будем изучать работу с командной строки Unix, и вы увидите это в окне xterm. Для того, чтобы напечатать

что-либо в xterm, вам обычно приходиться передвинуть курсор мыши (возможно имеющий форму "X" или стрелочки) в окно xterm. Однако, конкретная ситуация зависит от оконного менеджера.

### 5.3 Оконные Менеджеры

В Linux'е существуют два наиболее часто используемых оконных менеджера. Один из них - twm (сокращенное Tab Window Manager). Он больше, чем другой распространенный оконный менеджер fvwm (сокращенное F(?) Virtual Window Manager - автор не смог выяснить, что означает буква f в названии). Twm и fvwm очень гибко конфигурируются, поэтому я не могу точно сказать какие клавиши что делают в вашей конкретной настройке.

Чтобы изучить конфигурирование twm смотрите часть 9.2.1. Конфигурирование fvwm описано в части 9.2.2.

#### 5.3.1 Создание Новых Окон

Оконный менеджер при создании нового окна будет выполнять одно из трех действий. Можно устанавить конфигурацию оконного менеджера, так чтобы был виден контур нового окна на вашем экране, давая вам возможность поместить его в подходящее место. Это называется "размещение вручную".

Возможно, что оконный менеджер будет размещать новое окно на экране сам. Это называется "случайное размещение".

Наконец, иногда приложение будет просить задать точку на экране, или будет установлена такая конфигурация оконного менеджера, чтобы отображать конкретное приложение на одном и томмм же месте экрана. (Например, я могу задать, чтобы xclock всегда появлялись в верхнем правом углу экрана.)

#### 5.3.2 Фокус Ввода

Оконный менеджер управляет некоторыми важными вещами. Первое, что вам будет интересно - это фокус ввода. Фокус ввода сервера - это окно, которое получает то, что вы печатаете на клавиатуре. Обычно в X фокус ввода определяется позицией курсора мыши. Если курсор мыши находится в одном из окон xterm, xterm получит сообщение при нажатии клавиши. Обратите внимание на то, что такой подход отличается от многих других оконных систем, таких как Microsoft Windows, OS/2 или Macintosh, где вы должны щелкнуть мышью для того, чтобы окно получило поле ввода. Обычно под X, если курсор мыши вышел за пределы окна, поле ввода будет потеряно и вы не сможете больше печатать в нем.

Обратите внимание, что возможно установить конфигурацию как twm, так и fvwm, таким образом, что при нажатии кнопки мыши в окне оно приобретало фокус ввода, и при нажатии кнопки мыши вне окна теряло фокус. Попытайтесь понять, как устанавливается конфигурация оконного менеджера методом проб и ошибок, или почитайте документацию.

#### 5.3.3 Перемещение Окон.

Другая легко конфигурируемая вещь - перемещение окон. В моей личной конфигурации twm, есть три способа перемещения окон. Наиболее очевидный метод - поместить курсор мыши в строку заголовка и перемещать окно по экрану. К сожалению, это можно делать с помощью

левой, правой или средней кнопки (если мышь двухкнопочная, то средняя кнопка эмулируется одновременным нажатием левой и правой). Чтобы переместить окно - поместите курсор в строку заголовка и держите кнопку при перемещении мыши.

Другой способ передвижения окон заключается в том, чтобы держать нажатой некоторую клавишу при перемещении мыши. Например, в моей конфигурации, если держать клавишу Alt нажатой и передвигать курсор в окне, то можно перемещать окно.

Опять же, вы можете понять, как устанавливается конфигурация оконного менеджера методом проб и ошибок, или прочитать документацию. Кроме того, если вы хотите разобраться в конфигурационном файле

оконного менеджера, для описания twm смотрите часть 9.2.1 или часть

## 9.2.2 для описания fvwm.

### 5.3.4 Глубина

Так как окна могут перекрываться, в X введено понятие глубины. Несмотря на то, что как окна, так и экран двумерны, верхнее окно может частично или полностью скрывать нижнее.

Имеется несколько действий с глубиной:

- Перемещение окна наверх. Обычно это делают нажатием одной из кнопок мыши на строку заголовка окна. Это может быть любая кнопка (или несколько кнопок), в зависимости от того, как сконфигурирован оконный менеджер.

- Перемещение окна на задний план. Обычно это делают нажатием другой кнопки мыши на строку заголовка окна. Возможно установить конфигурацию некоторых окнных менеджеров так, чтобы нажатие кнопки мыши перемещало окно наверх, если есть какое-либо окно над ним, а нажатие той же кнопки будет перемещать окно вниз, когда оно находится на переднем плане.

- Циклическое перемещение окна - еще одна операция, которую поддерживают многие оконные менеджеры. Она перемещает окна в циклическом порядке.

### 5.3.5 Минимизация и Увеличение

Есть несколько операций, которые могут спрятать или увеличить окно. Первая из них - минимизация. В зависимости от конфигурации оконного менеджера минимизация может быть выполнена несколькими способами. В twm многие устанавливают конфигурацию менеджера иконок - это специальное окно, которое содержит список всех других окон на экране. Если вы нажимаете кнопкой мыши на название окна (в зависимости от настройки это может быть любая кнопка), окно исчезает - оно минимизируется. Окно все еще активно, но вы не можете этого видеть. Повторное нажатие кнопой мыши на название окна в менеджере иконок

восстанавливает окно на экране.

Это очень полезно. Например, вы имеете удаленные xterm'ы на нескольких компьютерах, но так как вы редко используете все xterm'ы одновременно, можно минимизировать некоторые из окон xterm'а с которым

вы в данный момент не работаете активно . Единственная проблема в том, что вы легко можете "потерять" окно. Так же очень легко создать новое окно, которое дублирует минимизированное, просто потому, что вы забыли о последнем.

Другие оконные менеджеры могут действительно создавать иконки внизу экрана, или просто разбрасывать иконки.

Другая операция, которую поддерживают многие оконные менеджеры - это увеличение. В twm, например, вы можете увеличить высоту, ширину окна или оба параметра. Это называетс "zooming" (распахивание окна) на языке twm, хотя мне больше нравится термин "maximization" (увеличение), так как различные приложения по разному реагируют на изменение их размера окна. (Например, xterm не делает шрифт больше, если я увеличу рабочую область.

К сожалению, нет стандартного способа увеличения окон.

### 5.3.6 Меню

Другая задача оконного менеджера - создание пользовательского меню, для быстрого выполнения задач. Например, я могу выбрать пункт меню, который запускает Emacs, мощный текстовый редактор, или еще один xterm. При этом не нужно печатать xterm - это особенно важно, когда ни один xterm не исполняется.

Как правило, вызвать меню можно нажатием на кнопку мыши в главном окне - неподвижном окне, находящимся позади всех остальных окон. По умолчанию оно окрашено в серый цвет, но может выглядеть и по другому. (Есть программы, которые что-нибудь рисуют на заднем плане). Чтобы меню появилось, нажмите кнопку мыши на панели экрана и держите ее нажатой. Чтобы выбрать пункт меню, передвиньте (не отпуская кнопку) курсор на один из пунктов и затем отпустите кнопку мыши.

## 5.4 Запуск и Остановка Системы X Window

### 5.4.1 Запуск X

Даже если X не запускается автоматически при входе в систему, возможно запустить его из строки запроса на ввод shell'а в обычном текстовом режиме. Есть две команды, которые запускает X, startx или xinit. Попробуйте сначала запустить startx. Если shell говорит, что такой команды нет ("command not found"), попытайтесь воспользоваться xinit и посмотреть, запустился ли X. Если не одна из команд не работает, возможно в вашей системе не установлен X - посмотрите документацию вашего дистрибутива.

### 5.4.2 Остановка X

В меню имеется один важный пункт: "Exit Window Maneger" (Завершить выполнение оконного менеджера) или "Exit X" (Выйти из X) или что-нибудь подобное. Попытайтесь найти и выбрать этот пункт меню (помните, что может быть более одного меню - попробуйте нажимать различные кнопки мыши). Если X запускался автоматически при входе в систему, вы выйдете из X. Для того, чтобы вернуться в X, просто войдите в систему. Если вы запускали X вручную, то выбор данного пункта меню возвратит вас к командной строке в текстовом режиме.

## 5.5 X-Программы

Многие программы используют X. Некоторые из них, например, emacs,

могут исполняться как программа в текстовом режиме или как программа, создающая свое собственное X окно. Однако большинство X программ могут исполняться только под X.

### 5.5.1 Геометрия

Есть несколько вещей, общих для всех программ, исполняющихся под X. В X, понятие геометрии - месторасположение окна и его размеры.

## 6. Работа с Unix

```
better !pout !cry
better watchout
lpr why
santa claus <north pole >town

cat /etc/passwd >list
ncheck list
ncheck list
cat list | grep naughty >nogiftlist
cat list | grep nice >giftlist
santa claus <north pole > town

who | grep sleeping
who | grep awake
who | egrep 'bad|good'
for (goodness sake) {
        be good
}
```

Unix - это мощная система, для тех, кто знает, как использовать ее мощь. В этой главе я постараюсь описать различные способы более эффективного использования оболочки (shell) Unix 'bash'.

### 6.1 Метасимволы

В предыдущей главе вы изучили команды работы с файлами cp, mv и rm. Иногда оказывается так, что вам нужно работать одновременно с несколькими файлами. Например, вы захотели скопировать все файлы, начинающиеся с 'data' в директорию ~/backup. Вы можете сделать это, несколько раз применяя команду cp, или можете составить список всех файлов в одной командной строке. Эти способы занимают много времени и у вас есть большой шанс допустить ошибку. Лучше решить эту задачу по-другому:

```
/home/larry/report# ls -F
1993-1          1994-1          data1           data5
1993-2          data-new        data2
/home/larry/report# mkdir ~/backup
/home/larry/report# cp data* ~/backup
/home/larry/report# ls -F ~/backup
data-new        data1           data2           data5
/home/larry/report#
```

Как вы можете видеть, звездочка говорит cp взять все файлы начинающиеся с 'data' и скопировать их все в директорию ~/backup.

Догадайтесь, что будет делать команда cp d*w ~/backup?

### 6.1.1 Что же Происходит на Самом Деле?

Хороший вопрос. На самом деле, есть пара специальных символов, которые перехватывает bash, shell Unix'а. Символ "*", говорит: "замени это слово на все имена файлов, которые подходят под этот шаблон. Так, команда cp data* ~/backup, как и команда выше, заменяется на cp data-new data1 data2 data5 /backup перед исполнением.

Чтобы проиллюстрировть это, давайте рассмотрим новую команду echo. echo чрезвычайно простая команда; она печатает свои аргументы:

```
/home/larry# echo Hello!
Hello!
/home/larry# echo How are you?
How are you?
/home/larry# cd report
/home/larry/report# ls -F
1993-1          1994-1          data1           data5
1993-2          data-new        data2
/home/larry/report# echo 199*
1993-1 1993-2 1994-1
/home/larry/report# echo *4*
1994-1
/home/larry/report# echo *2*
1993-2 data2


/home/larry/report#
```

Как вы видите, shell подставляет значение вместо "*" и передает все файлы программе, которая будет исполняться. При этом возникает интересный вопрос: что случиться, если нет ни одного файла, подходящего под шаблон? Попробуйте сделать так: echo /rc/fr*og и посмотрите, что произойдет... bash передаст определение шаблона программе не подставляя значения.

Другие интерпретаторы shell, как, например, tcsh, в таком случае ответят No match.

```
mousehouse>echo /rc/fr*og
echo: No match.
mousehouse>
```

Еще один вопрос, на который вы, может быть, захотите получить ответ - это что делать, если нужно, чтобы команда echo выдала data*, а не список имен файлов? Как в bash, так и в tcsh, просто заключите строку в кавычки:

```
/home/larry/report# echo "data*"
data*
/home/larry/report#
```

или

```
mousehouse>echo "data*"
data*
mousehouse>
```

### 6.1.2 Знак Вопроса

Кроме звездочки, shell интерпретирует и знак вопроса как

специальный символ. Знак вопроса соответствует одному, и только одному символу. Например, ls /etc/?? отобразит все файлы из двух букв, которые находятся в директории /etc.

## 6.2 Экономия Времени при Использовании bash

Бывает так, что вы напечатали длинную команду, но перед тем, как нажать клавишу ввода, обнаружили синтаксическую ошибку. Вы можете удалить всю строку, а затем напечатать ее заново, но при этом придется потратить много сил! Вместо этого, вы можете, используя клавиши стрелок, передвинуть курсор к месту ошибки, удалить символ или два, и напечатать их правильно.

Есть много специальных символов для редактирования командной строки, большая часть из них похожи на команды GNU Emacs. Например, C-t (обозначение для Ctrl-T) меняет местами два соседних символа. Вы можете найти большинство команд в главе 8, Emacs.

### 6.2.2 Завершение командной строки

Другая черта bash - это автоматическое завершение командной строки. Например, давайте посмотрим на пример типичной команды cp:

```
/home/larry# ls -F
this-is-a-long-file
/home/larry# cp this-is-a-long-file shorter
/home/larry# ls -F
shorter              this-is-a-long-file
/home/larry#
```

Очень обидно печатать this-is-a-long-file каждый раз, когда вы хотите доступиться к файлу. Создайте файл this-is-a-long-file, копируя его в /etc/rc. (команда cp /etc/rc this-is-a-long-file). Далее мы выполним ту же самую команду cp очень быстро и с небольшой вероятностью опечатки.

Вместо того, чтобы печатать имя целиком, напечатайте cp th, затем, нажмите и отпустите клавишу Tab. Как по волшебству, остальная часть имени файла появиться в командной строке, и затем вы сможете напечатать shorter. К сожалению, bash не может читать ваши мысли, и вам придется напечатать shorter целиком.

Когда вы нажимаете Tab, bash смотрит на то, что вы напечатали и ищет файл, который так начинается. Например, если я напечатал /usr/bin/ema, а затем нажал клавишу Tab, bash найдет /usr/bin/emacs, так как это единственный файл, который начинается на /usr/bin/ema в моей системе. Однако, если я напечатаю /usr/bin/ld и затем нажму клавишу Tab, bash издаст звуковой сигнал, потому что в моей системе есть три файла, /usr/bin/ld, /usr/bin/ldd и /usr/bin/ld86, которые начинаются на /usr/bin/ld.

Если вы пытаетесь завершить строку и bash издает звуковой сигнал, можно сразу же нажать клавишу Tab, чтобы получить полный список подходящих файлов. Таким образом, если вы не знаете точно, как пишется ваш файл, вы можете написать несколько букв, а затем просмотреть небольшой список файлов.

## 6.3 Стандартный Ввод и Стандартный Вывод

Давайте попытаемся взяться за задчу: получить список файлов в директории /usr/bin. Если мы просто сделаем ls /usr/bin, часть файлов не поместится на экране. Как же можно посмотреть все файлы?

### 6.3.1 Понятия Unix'а

Это делается очень просто в операционной системе Unix. Когда программа выводит что-то на экран, она пользуется стандартным выводом. Стандартный вывод, сокращенно stdout, это то, куда программа выводит результаты. Стандартный ввод, (stdin) - это то, откуда программа получает свои параметры. Возможно, что программа общается с пользователем, не используя стандартный ввод и вывод, но это случается очень редко - все команды, которые мы изучали, используют stdin и stdout.

Например, команда ls выводит список каталогов на стандартный вывод, который обычно связан с терминалом. Диалоговая программа, такая как bash, считывает ваши команды со стандартного ввода.

Есть возможность для программы выводить результаты в стандартный поток ошибок, так как очень легко переключить стандартый вывод на что-нибудь, отличное от терминала. Стандартный поток ошибок, почти всегда связан с терминалом так, чтобы человек действительно читал сообщение об ошибках.

В этой части мы собираемся рассмотреть три способа переключения ввода-вывода: перенаправление ввода, перенаправление вывода и каналы.

### 6.3.2 Перенаправление Вывода

Очень важная особенность Unix'a - возможность перенаправлять вывод. Это позволяет вам, вместо просмотра результатов работы команды, сохранить их в файле или послать их прямо на принтер. Например, чтобы перенаправить вывод команды ls /usr/bin, надо поместить знак > в конце строки, а затем написать, в какой файл вы хотите записать результаты работы.

```
/home/larry# ls
/home/larry# ls -F /usr/bin > listing
/home/larry# ls
listing
/home/larry#
```

Как вы можете видеть, вместо того, чтобы вывести имена всех файлов, команда создает новый файл в вашей персональной директории. Попытайтесь посмотреть содержимое файла, используя команду cat. Вспомните, cat - очень полезная команда, которая копирует то, что вы печатаете (стандартный ввод) на терминал (стандартный вывод). cat также может вывести файл на стандартный вывод, если вы передаете имя файла команде cat как параметр:

```
/home/larry# cat listing
...
/home/larry#
```

Вывод команды ls /usr/bin в точности представляет собой содержимое файла listing. Все хорошо, но это не решает поставленной

задачи. (Для нетерпеливых читателей: команда, которую вы можете запустить - это more. Однако, надо кое-что еще выяснить перед тем, как мы добьемся желаемого результата.)

Однако, cat делает некоторые интересные вещи при перенаправлении вывода. Что делает команда cat listing > newfile? Обычно, > newfile говорит "возьми весь вывод команды и помести его в файл newfile." Вывод команды cat listing - есть сам файл listing. Таким образом мы изобрели новый (но не такой эффективный) способ копирования файлов.

А как работает команда cat > fox? cat считывает все строки, выведенные на терминал (стандартный ввод) и печатает их в стандартный вывод, пока не считает строку Ctrl-d. В этом случае стандартный вывод перенаправлен в файл fox. Теперь cat работает как элементарный редактор:

```
/home/larry\# cat > fox
The quick brown fox jumps over the lazy dog.
{press Ctrl-d}
```

Сейчас мы создали файл "fox", содержащий предложение "The quick brown fox jumps over the lazy dog." Еще одно предназначение команды cat - конкатенация файлов. cat печатает все файлы, которые ему были переданы в качестве параметра, один за другим. Команда cat listing fox сначала напечатает список файлов в директории /usr/bin, а затем напечатает предложение из файла fox. Таким образом команда cat listing fox > listandfox создаст новый файл, в котором будет находится содержимое файлов listing и fox.

### 6.3.2 Перенаправление ввода

Подобно перенаправлению стандартного вывода можно перенаправить и стандартный ввод. Вместо считывания с клавиатуры программа будет считывать из файла. Так как перенаправление вывода логически связано с перенаправлением ввода, кажется естественным ввести специальный символ для перенаправления ввода таким образом: <. Этот символ также как и > используется после названия команды, которую вы хотите исполнить.

Обычно перенаправление ввода полезно, когда у вас есть файл с данными и команда, которая ожидает входные данные со стандартного ввода. Большинство команд позволяют задавать файл, с которым команда она будет работать, так что < не используется так часто в обычных коандах, как другие методы.

### 6.3.4 Решение: Канал

Команды Unix выводят большое количество информации. Например, обычно команда ls /usr/bin выводит больше информации, чем вы можете просмотреть на экране. Для того, чтобы было возможно просмотреть всю информацию, выданную командой, подобной ls /usr/bin, необходимо использовать другую команду Unix'а, more. (Программа more называется так потому что первоначально она выдавала приглашение --more--. Во многих версиях Linux кроме команды more есть более мощная команда, которая может делать все то, что и more, и даже больше. Ее название? Конечно, less. В английском языке more означает больше, а less - меньше.) Программа more останавливается каждый раз после того, как выдаст объем информации, равный размеру экрана. Например, more < /etc/rc выведет файл /etc/rc точно также как это сделала бы команда cat /etc/rc, позволяя вам, кроме прочего, прочесть файл. (more позволяет указывать имя просматриваемого файла как аргумент в командной строке: more /etc/rc).

Однако, это не решает той проблемы, что ls /usr/bin выводит больше информации, чем вы можете увидеть. more < ls /usr/bin не будет работать, так как перенаправления ввода работает только с файлами, а не с командами. Вы можете сделать следущее:

```
/home/larry# ls /usr/bin > temp-ls
/home/larry# more temp-ls
...
/home/larry# rm temp-ls
```

Однако в Unix есть более красивый способ сделать то же самое. Вы просто можете использовать команду ls /usr/bin | more. Символ "|" указывает на то, что это канал. Как и речной канал, канал в Unix управляет потоком. Вместо того, чтобы управлять потоком воды, мы

управляем потоком информации!

Полезным инструментом для работы с каналами являются программы, которые называются фильтрами. Фильтр - это программа, которая читает из стандартного ввода, преобразует его некоторым образом, и выводит на стандартный вывод. more является фильтром - она читает данные из стандартного ввода, и выводит их на стандартный вывод, таким образом, что видны данные, размером в один экран, позволяя вам таким образом прочесть файл.

Пример других фильтров - прграммы cat, sort, head и tail. Например, если вы хотите прочитать только верхние десять строк вывода команды ls, увы можете использовать команду ls /usr/bin | head.

## 6.4 Многозадачность

### 6.4.1 Основы

Управление задачами - это воможность заставить процессы (другими словами, программы) работать в фоновом режиме и возвращать их обратно на передний план. Пусть вы хотите запустить какой-то процесс, в то время как вы занимаетесь чем-то другим, но иметь возможность сообщить что-то процессу или остановить его. В Unix, главный инструмент управления задачами - это shell, он будет управлять вашими задачами, если вы научитесь говорить на языке shell'а.

Два самых важных слова в этом языке - это fg, "фонововый процесс", и bg, "приоритетный процесс". Чтобы узнать, как они работают, напишите команду yes в командной строке.

```
/home/larry# yes
```

Результатом работы этой команды будет длинный столбец символов y в левом краю экрана, бегущий быстрее, чем вы можете следить за этим. (Есть два достаточно сильных основания существования этой странной команды, но мы не будем их объяснять их сейчас). Чтобы остановить эту программу, вы обычно пишете ctrl-C и уничтожаете ее, но вместо этого

сейчас вам надо написать ctrl-Z. Кажется, что она остановилась, но перед приглашением на ввод в командной строке появится сообщение, которое выглядит примерно так:

```
[1]+   Stopped                 yes
```

Это означает, что процесс yes был приостановлен. Вы можете запустить его опять, написав fg в командной строке, эта команда опять сделает этот процесс приоритетным. Если хотите, вы можете сделать что-нибудь другое, в то время как процесс приостановлен. Попробуйте, например, исполнить команду ls или какую-нибудь другую, перед тем, как сделать процесс активным.

После того, как вы сделали процесс yes приоритетным, 'y' опять начинает бежать по экрану, также как и раньше. Не следует беспокоиться о том, что пока процесс был приостановлен, он накопил побольше символов, и теперь посылает на экран и их; когда программа приостановлена, она не исполняется до тех пор, пока вы не вернете ее к жизни. (Теперь вы можете написать ctrl-C для того, чтобы уничтожить процесс, если вы достаточно поэкспериментировали с ним).

Давайте разберем по частям сообщение, которое мы получили от shell'а:

```
 [1]+   Stopped                 yes
```

Число в скобках - это индекс задачи, он используется, когда нам надо сослаться конкретно на нее. (Естественно, так как управление задачами дает полную информацию о запущенных процессах, нам надо уметь отличать один процесс от другого.) "+", который стоит после числа в скобках, говорит о том, что этот процесс является "текущим процессом", то есть, он был самым последним переведен из приоритетного режима в фоновый. Если вы напишете fg, вы переведете задачу с "+" в приоритетный режим. (Подробнее об этом позже, когда мы будем обсуждать исполнение нескольких задач одновременно.) Слово Stopped означает, что процесс приостановлен. Процесс не "умер", но сейчас он не выполняется. Linux хранит его в особом приостановленном состоянии, готовым продолжить работу, если будет дана соответствующая команда. И наконец,

yes - имя команды, которое было введено в командной строке при запуске программы.

Перед тем как продолжить, давайте уничтожим эту задачу и запустим ее по-другому. Команда уничтожения процесса называется kill и она используется следующим образом:

```
/home/larry# kill %1
[1]+  Stopped                 yes
```

Это сообщение о том, что процесс был опять остановлен, может ввести в заблуждение. Чтобы выяснить, "жив" ли еще процесс, (то есть, исполняется ли он или находится в приостановленном состоянии), напишите в командной строке jobs:

```
/home/larry# jobs
[1]+  Terminated                yes
```

Процесс был завершен! (Возможно, что команда jobs не выдаст никакого сообщения, что означает, что не один из процессов не запущен в фоновом режиме. Если вы уничтожаете процесс, и после этого команда jobs не выдаст никакого сообщения, вы можете убедиться, что процесс действительно был уничтожен. Обычно команда jobs сообщит о том, что процесс был завершен.)

Теперь, запустите yes еще раз, таким образом:

```
/home/larry# yes > /dev/null
```

Если вы прочитали часть книги о перенаправлении ввода и вывода, вы знаете, что таким образом вы посылаете вывод команды yes в файл /dev/null. /dev/null - это черная дыра, которая поглощает весь вывод, посланный ей (вы можете представить, что поток символов "y" выходит позади вашего компьютера и просверливает дыру в стене и там исчезает, если вам так больше нравится.)

После того, как вы напечатаете это, в командной строке не появится приглашение на ввод, но вы также не увидите колонку из

символов "y". Хотя вывод был перенаправлен в /dev/null, процесс все еще выполняется в фоновом режиме. Как обычно, вы можете приостановить его, нажав ctrl-Z. Сделайте это, чтобы вернуть приглашение на ввод в командной строке.

```
/home/larry# yes > /dev/null
[процесс "yes" выполняется; если напечатать ctrl-z, мы
приостановим процесс и возвратим приглашение на ввод в командной строке.
Представьте, что я только что сделал это...]
[1]+  Stopped                 yes >/dev/null
```

Гм... есть ли какой-нибудь способ действительно заставить исполняться процесс в фоновом режиме, чтобы при этом можно было вводить команды в командную строку? Конечно, есть, иначе бы я не задавал этот вопрос. Эта команда называется bg:

```
/home/larry# bg
[1]+ yes >/dev/null  &
/home/larry#
```

Сейчас вам придется поверить мне на слово: после того, как вы написали bg, команда yes > /dev/null опять начала исполняться, но уже в фоновом режиме. Действительно, если вы напишите в командной строке что-нибудь вроде ls, то можете заметить, что ваша машина начала работать немного медленнее, вывод потока символов "сзади" машины требует некоторой работы! Однако, кроме этого не проявляется никаких эффектов. Вы можете делать все, что вам угодно, и команда "yes" будет продолжать посылать свой вывод в "черную дыру".

Теперь есть два различных способа уничтожить процесс: командой kill, которую вы только что изучили, или помещением процесса в приоритетный режим и прерыванием его (ctrl-C). Давайте попытаемся сделать это вторым способом, просто для того, чтобы немного лучше понять взаимоотношение между fg и bg;

```
/home/larry# fg
yes >/dev/null
```

```
[теперь процесс опять находится в приоритетном режиме. Представьте себе, что
я нажал ctrl-C, чтобы завершить его]

/home/larry#
```

Сейчас запустим несколько процессов, исполняющихся одновременно, следующим образом:

```
/home/larry# yes  > /dev/null &
[1] 1024
/home/larry# yes | sort > /dev/null &
```

```
[2] 1026
/home/larry# yes | uniq > /dev/null
[здесь нажмите ctrl-Z, чтобы приостановить процесс]

[3]+  Stopped                 yes | uniq >/dev/null
```

Первое, на что вы можете обратить внимание - это & в конце первых двух команд. Наличие & в конце команды говорит shell'у о том, что надо исполнять процесс в фоновом режиме с самого начала. (Таким образом, можно избежать более сложного способа исполнения процесса в фоновом режиме, который мы уже рассматривали: запустив программу, нажав ctrl-Z, и затем написав bg.) Таким образом, мы запустили две команды в фоновом режиме. Третья приостановлена и является неактивной в данный момент. Вы можеет заметить, что машина стала работать медленнее, так как две исполняемых команды занимают значительное количество времени CPU.

Каждая задача сообщает свой номер. Первые две из них сообщают свой идентификационный номер процесса или PID, который расположен сразу после номера задачи. Обычно вам не надо знать PID, но иногда это бывает полезно.

Давайте уничтожим второй процесс, так как он замедляет работу вашей машины. Вы можете написать kill %2, но это будет слишком просто. Вместо этого сделайте так:

```
/home/larry # fg %2
[и затем нажмите ctrl-C, чтобы уничтожить процесс]
```

Как только что показано, параметры fg начинаются с %. На самом деле вы можете написать таким образом:

```
/home/larry # %2
[и затем нажмите ctrl-C, чтобы уничтожить процесс]
```

Такая команда будет работать, так как shell автоматически интерпретирует номер задачи, как требование поместить задачу в приоритетный режим. Shell может отличать номера задач, которые начинаются с %. Теперь напишите команду jobs, чтобы посмотреть, какие задачи сейчас исполняются:

```
/home/larry # jobs
[1]-  Running                 yes >/dev/null  &
[3]+  Stopped                 yes | uniq >/dev/null
```

'-' означает, что задача с номером 1 будет второй по очереди установлена в приоритетный режим, если вы напишите fg, не передавая этой команде никаких параметров. Однако, вы можете установить в приоритетный режим любую задачу, если передать команде номер задачи:

```
/home/larry # fg %1
yes >/dev/null
[и затем нажмите ctrl-Z, чтобы приостановить процесс]

[1]+  Stopped                 yes >/dev/null
```

Установка задачи в приоритетный режим и приостанавление ее изменяет приоритет всех ваших задач. Вы можете убедиться в этом, при помощи команды jobs:

```
/home/larry # jobs
[1]+  Stopped                 yes >/dev/null
[3]-  Stopped                 yes | uniq >/dev/null
```

Сейчас оба процесса не исполняются (так как они были приостановлены посредством ctrl-Z), и задача с номером 1 - первая в очереди быть установленой в приоритетный режим по умолчанию. Это происходит потому, что вы поместили ее в приоритетный режим вручную, а затем приостановили ее. '+' всегда указывает на самую последнюю задачу, приостановленную в приоритетном режиме. Вы можете запустить ее вновь:

```
/home/larry # bg
[1]+ yes >/dev/null  &
/home/larry# jobs
[1]-  Running                 yes >/dev/null
[3]+  Stopped                 yes | uniq >/dev/null
```

Обратите внимание на то, что сейчас задача с номером 1 исполняется, а другая переместилась в очереди, и теперь имеет '+'. Хорошо, давайте теперь уничтожим все процессы, чтобы возвратить машину в исходное состояние:

```
/home/larry# kill %1
/home/larry# kill %3
```

Вы можете увидеть различные сообщения о завершении процесса - ничто не умирает спокойно. Обобщим то, что вы должны были узнать к настоящему моменту об управлении задачами:

[ctrl-z] эквивалент в DOS: Ха! В DOS'е нет настоящего управления задачами... Такая комбинация клавиш вызывает приостановление задачи, хотя некоторые программы игнорируют его. После того, как задача приостановлена, она может исполняться в фоновом

### 6.3.3 Перенаправление ввода

не настоящая команда, а просто сигнал.

[fg] эквивалент в DOS: никакого. Может быть когда-нибудь... Это встроенная команда shell'а устанавливает задачу в приоритетный режим. Чтобы понять, какая задача будет установлена в приоритетный режим по умолчанию, напишите jobs, и найдите задачу с '+'. Параметры: номер задачи (или по

умолчанию будет установлена задача с '+').

[&] Когда & добавляется в конец командной строки, это заставляет команду исполняться в фоновом режиме автоматически. Это соответствует всем обычным методам управления задачами, изложенным здесь.

[bg] Это встроенная команда shell'а, которая устанавливает задачу в фоновый режим. Чтобы понять, какая задача будет установлена по умолчанию, напишите jobs, и найдите задачу с '+'. Вы можете представлять bg, как fg&! Параметры: номер задачи (или по умолчанию будет задача с '+').

[kill] Эта команда завершает задачу в фоновом режиме, приостановленную или ту, которая исполняется. Вы должны всегда задавать номер задачи или PID, и, если вы используете номера задач, не забывайте ставить % перед ними. Параметры: номер задачи

(перед которым стоит %) или PID (% ставить необязательно).

[jobs] Эта команда shell'а просто перечисляет информацию о задачах, которые исполняются или приостановлены. Иногда она также сообщают о процессах, которые благополучно завершились или были завершены.

[ctrl-c] Это общий символ прерывания. Обычно, если вы нажимаете эту комбинацию клавиш, когда программа исполняется в приоритетном режиме, то уничтожаете программу (иногда для этого надо несколько попыток). Однако, не все программы будут реагировать на этот способ прерывания.

### 6.4.2 Что же происходит на самом деле?

Важно понимать, что управление задачами осуществляется shell'ом. В системе нет самостоятельной программы, которая называется fg; вместо этого, fg, bg, &, jobs и kill все являются встроенными командами shell'а (на самом деле, иногда kill независимая программа, но она встроена в bash - один из shell'ов, используемый в Linux'е).

Логично было сделать именно так: поскольку каждый пользователь хочет иметь свое собственной пространство управления задачами, и каждый пользователь уже имеет свой собственный shell, это самый простой способ заставить shell следить за задачами пользователя. Поэтому, номер задачи пользователя имеет значение только для пользователя: мой номер задачи и ваш номер задачи, вероятно, являются совершенно различными процессами. На самом деле, если вы входили в систему больше, чем один раз, каждый из ваших shell'ов будет иметь уникальные данные управления задачами, поэтому вы, как пользователь, можете иметь две различных задачи с одним и тем же номером, которые исполняются в двух различных shell'ах.

Другой, более надежный способ - использовать номера идентификатора процесса PID. Они являются общесистемными - каждый процесс имеет свой собственный уникальный PID. Два различных пользователя могут ссылаться на PID и знать, что они имеют в виду один и тот же процесс. (Предполагая, что они работают на одной и той же машине!)

Давайте рассмотрим еще одну команду, чтобы понять, что же такое PID. Команда ps перечисляет все исполняющиеся процессы, включая и ваш shell. Попробуйте исполнить эту команду. Она имеет несколько опций, наиболее важные из которых (для большинства людей) 'a', 'u' и 'x'. Опция 'a' перечисляет процессы, принадлежащие любому пользователю, а не только ваши собственные. Опция 'x' перечисляет процессы, которые не имеют связанного с ними терминала. Это имеет смысл только для определенных системных программ, которые не общаются с пользователем в диалоговом режиме. Наконец, опция 'u' выдает дополнительную информацию о процессах, которая часто бывает полезна.

Для того, чтобы понять, что на самом деле делает ваша система, напишите все три опции вместе ps -aux. Вы можете найти процесс, который использует больше всех памяти, посмотрев на колонку %MEM, больше всех времени CPU, посмотрев в колонку %CPU. (В колонке TIME указано общее количество затраченного времени CPU.)

Еще одно небольшое замечание о PID. Команда kill, кроме того, что использует параметры вида %job#, может использовать и PID. Установите команду yes > /dev/null в фоновый режим, запустите ps, и посмотрите на PID команды yes. Затем напишите kill PID. (В общем случае, легче уничтожить задачу по номеру, а не используя PID.)

Если вы начнете программировать на C на вашей Linux системе, вы вскоре узнаете, что управление задачами shell'а - это просто диалоговая версия вызовов функций fork и execl. Сейчас это довольно сложно понять, но может оказаться полезным позже, когда программируя, вы захотите запустить много процессов из одной программы.

### 6.5 Виртуальная Консоль: Быть в Нескольких Местах Одновременно

Linux поддерживает виртуальные консоли. Таким образом можно представить вашу машину, как машину с несколькими терминалами, присоединенных к ядру Linux. К счастью, использование виртуальных консолей одна из самых простых вещей в Linux: есть "горячие клавиши" для быстрого переключения между консолями. Чтобы попробовать это, держите нажатой левую клавишу Alt, и нажмите F2 (это функциональная клавиша 2) (Убедитесь, что вы делаете это из текстовой консоли: если вы работаете под X или с другим графическим приложением, скорее всего это не будет работать, хотя ходят слухи, что в X Windows скоро будет предусмотрено переключение виртуальных консолей под Linux.)

Вы должны оказаться перед еще одним приглашением на вход в систему. Не волнуйтесь: вы сейчас находитесь на виртуальной консоли (VC) номер 2! Войдите в систему и что-нибудь сделайте, для того, чтобы убедиться, что это настоящий shell. Вы можете возвратиться на VC номер 1, держа нажатой левую клавишу Alt, и нажмав F1. Вы можете переключиться на третью виртуальную консоль очевидным способом (Alt-F3).

Системы Linux обычно имеют четыре VC по умолчанию. Вы можете увеличить это число до восьми; это должно быть описано в Руководстве Администратора Системы Linux. Для этого надо вносить изменения в файлы /etc. Однако, четырех VC должно быть достаточно для большинства людей.

Если вы однажды привыкли к ним, возможно, VC станут совершенно необхдимым инструментом для того, чтобы делать несколько действий одновременно. Например, я обычно запускаю Emacs на VC 1 (и делаю большую часть работы здесь), на VC 3 находятся программы связи (таким образом я могу передавать файлы по модему во время работы, или исполнять задачи на удаленной машине), на VC 2 находится shell, просто на случай, если я захочу выполнить что-либо не связанное с VC 1.

## 7. Небольшие, но мощные программы

### 7.1 Мощь Unix'а

Мощность Unix'а скрыта в маленьких командах, которые не кажутся уж очень полезными сами по себе, но в сочетании с другими командами (непосредственно или неявно) создают систему, более мощную и гибкую, чем большинство операционных систем. Команды, о которых я собираюсь рассказать sort, grep, more, cat, wc, spell, diff, head, tail. К сожалению, прямо сейчас не ясен смысл этих названий.

Давайте выясним, что каждая из этих утилит делает сама по себе, а затем, я объясню, как использовать их совместно.

Следует иметь ввиду, что краткое описание команд в этой главе не является исчерпывающим. Обращайтесь к справочному руководству man, если вам нужно узнать все опции.

## 7.2 Работа с файлами

В добавлении к командам cd, mv, rm, которые вы изучили в главе 4, есть и другие команды, которые работают с файлами (но не с данными внутри файлов). Эти команды touch, chmod, du и df. Все эти команды не работают с содержимым файла - они изменяют некоторые параметры файла, которые Unix запоминает.

Вот несколько вещей, которыми управляют эти команды:

- Установка даты обращения к файлу. Каждый файл имеет три даты, связанные с ним. Это время создания (когда файл был создан), время последней модификации файла (когда файл последний раз изменялся), и последнее время доступа (когда файл последний раз читали).

В старых файловых системах LINUX хранилась только одна дата, так как эти файловые системы наследовались от Minix. Если у вас одна из таких файловых систем - некоторая информация будет просто неприемлима, операции же по большей части не изменились.

- Владелец. У каждого файла в Unix'e есть один владелец.

- Группа. С каждым файлом связана группа пользователей этого файла. Наиболее общая группа для файлов пользователей называется users, в эту группу обычно включаются все пользователи системы.

- Права доступа. Каждый файл имеет права доступа (иногда его называют привелегии), которые сообщают, кто может читать файл, изменять его, или, в случае программы, исполнять ее. Каждая привелегия может быть изменена отдельно по отношению к владельцу, группе, или всем остальным пользователям.

```
touch file1  file2 ... fileN
```

touch будет изменять времена обращения к файлам, перечисленным в командной строке на текущее время (время, в которое команда была исполнена). Если файл не существует, touch создаст его. Также возможно задавать время явно, которое будет установлено для файлов - смотрите справочное руководство man для команды touch.

```
chmod [-Rfv] mode file1 file2 ... fileN
```

Команда, используемая для изменения прав доступа к файлу называется chmod, сокращенное change mode (изменить режим). Перед тем, как я расскажу как использовать эту команду, давайте обсудим какие права доступа есть в Unix. Каждый файл имеет связанную с ним группу

прав доступа. Эти права доступа сообщают Unix'у можно ли читать из файла, писать в файл, или исполнять файл, если он является программой. (В следующих нескольких параграфах я буду говорить о пользователях, которые выполняют некоторые действия. Естественно, что любым программам, которые исполняет пользователь, разрешено делать то же самое, что разрешено пользователю.

Однако, Unix различает три различных группы людей: во-первых, владелец файла (и тот, кому позволено применять к этому файлу chmod). Группа большей части ваших файлов может быть "users", то есть обычные пользователи системы. (Чтобы узнать группу конкретного файла, используйте команду ls -l file.) Третья группа людей это те, кто не является владельцем и членом группы файла.

Таким образом, файл может иметь права доступа чтения и записи по отношению к владельцу, права доступа на чтение для группы, и никаких прав для всех остальных. Или, в силу каких-то причин, файл может иметь права доступа чтения и записи для группы и всех остальных, но не иметь никаких прав доступа по отношению к владельцу.

Мы будем использовать команду chmod для изменения некоторых прав доступа. Сначала создайте новый файл, используя cat, emacs, или что-нибудь другое. По умолчанию вы будете иметь права доступа на чтение и запись для этого файла. (Пправа доступа, предоставленные всем остальным будут зависеть от установок системы и установок, касающихся вас как пользователя.) Убедитесь, что вы можете считать файл при помощи комады cat. Теперь, давайте отменим ваше право доступа на чтение, используя команду chmod u-r filename. (Параметр u-r расшифровывается как "user minus read" ("запретить чтение пользователю")). Если вы пытаетесь считать файл, то выдается ошибка "Permission denied" (нет прав доступа)! Установите назад право доступа на чтение, используя команду chmod u+r filename.

Директории, как и файлы, имеют права доступа чтения, записи, исполнения, но они действуют немного по другому. Привелегия чтения позволяет пользователю (или группе, или всем остальным) читать - смотреть имена файлов в директории. Право доступа на запись позволяет пользователю (или группе, или всем остальным) создавать или удалять

файлы. Привелегия исполнения дает право доступа к файлам в директории или поддиректориям. (Если пользователь не имеет право доступа на исполнение в директории, он не может даже перейти в эту директорию.

Используя chmod, измените права доступа пользователей, или группы, или всех остальных, или вообще для всех этих категорий, и укажите, как именно изменить привелегию. (То есть, используйте знак "+", для того, чтобы установить привелегию, и "-", чтобы убрать ее, или знак равенства, чтобы определить точные права доступа.) Также, возможно устанавливать права доступа при помощи "r" (read), "w" (write) и "x" (execute).

Флаг 'R' команды chmod изменяет права доступа директории, и всех файлов этой директории, и всех поддиректорий этой директории. ('R' означает recursive (рекурсивный)). Флаг 'f' заставляет chmod попытаться изменить права доступа, даже если пользователь не является владельцем файла. Если в качестве флага команде передается 'f', chmod не будет писать сообщение об ошибке, если не удастся изменить права доступа к файлу. Флаг 'v' заставляет команду chmod быть более информативной - она будет сообщать обо всем, что она делает.

### 7.3 Статистическая Информация о Системе

Команды в этой части показывают статистическую информацию об операционной системе или о ее частях.

#### du [-abs] [path1(путь1) path2 ... pathN]

du - сокращенное disk usage (использование диска). Эта команда

считает размер дискового пространства данной директории и всех ее поддиректорий. du перечисляет сколько места занимает каждая поддиректория текущей директории, и в самом низу, сколько места использует текущая директория (плюс сосчитанные поддиректории). Если вы передаете команде несколько параметров, она будет считать количество пространства, занимаемого этими файлами или директориями.

Флаг 'a' будет считать размер как директорий, так и файлов. Флаг 'b' будет отображать размер файлов в байтах, а не в килобайтах (1024 байт). Байт можно считать эквивалентным одному символу. Флаг 's' будет отображать все директории, указанные в командной строке без их поддиректорий.

df

df является сокращением disk free. df сообщает о количестве используемого пространства. Для каждой файловой системы (вспомните, различные файловые системы находятся на разных дисках или на разных логических дисках) эта команда показывает общее количество дискового пространства, используемого пространства, доступного пространства и общий объем, используемый файловой системой.

Вы можете встретиться с такими странными вещами: объем может превышать 100%, или используемое пространство плюс доступное не равно общему. Это происходит потому, что Unix резервирует некоторое пространство в каждой файловой системе для администратора системы (пользователя root). Таким образом, если пользователь случайно заполняет диск, в системе есть еще немного места для действий.

df не имеет никаких полезных для большинства людей опций.

uptime

Программа uptime делает именно то, что от нее ожидают. Она печатает время, которое система была в работе - время после последней загрузки.

uptime также выдает текущее время и среднюю загруженность. Средняя загруженность - это среднее число задач, ожидающих исполнения в определенный промежуток времени. uptime показывает среднюю загруженность в течение последней минуты, пяти минут, десяти минут. Средняя загруженность около нуля означает, что система почти незанята. Средняя загруженность около единицы означает, что используются почти все ресурсы системы, но нет перенагрузки. Высокая средняя загруженность - результат исполнения одновременно нескольких задач.

Удивительно, но uptime одна из немногих команд Unix, в которой вообще нет опций.

who

Команда who выводит список пользователей системы в настоящий момент, и то, когда они вошли в систему. Если команде передать параметр 'am i' (таким образом получится 'who am i'), она выведет информацию о текущем пользователе.

w [-f] [username]

Программа 'w' выводит список пользователей системы в настоящий момент, и что они делают. (Эта команда сочетает в себе действия команд uptime и who. Заголовок 'w' точно такой же, как и у uptime, и каждая строка показывает идентификатор пользователя, время, когда он вошел в систему и как долго он не предпринимал никаких действий. JCPU - общее количество времени CPU, которое занимал пользователь, а PCPU - общее количество времени CPU, которое занимает его текущая задача.

Если команде 'w' передается опция 'f', она показывает, с какой удаленной системы вошел пользователь. Необязательный параметр ограничивает действие команды 'w', показывая только имена пользователей.

7.4 Что находится в файле?

Есть несколько важных команд, которые используются в Unix'e для просмотра содержимого файлов, cat и more. Я рассказывал о них в главе

**6.**

cat [-nA] [file1 file2 ... fileN]

cat не очень дружественная команда - она не ждет, пока вы прочитаете файл, и в большинстве случаев используется вместе с каналами. Однако, cat имеет несколько полезных опций. Например, 'n' будет нумеровать все строки в файле, а 'A' покажет символы управления, как нормальные символы, вместо того, чтобы (возможно) делать всякие

странные вещи на экране. (Помните, чтобы посмотреть некоторые странные и, возможно, "менее полезные" опции, надо использовать команду man: man cat.) cat берет входные данные со стандартного ввода, если в командной строке не указано никаких файлов.

more [-l] [+ linenumber] [ file1 file2 ... fileN]

more более полезна, и эту команду лучше использовать при просмотре текстовых ASCII файлов. Единственная интересующая нас опция это l, которая сообщает more, что вы не хотите рассматривать Ctrl-L как символ "новой страницы". more начинает показывать файл, начиная с заданного номера строки.

Так как more это диалоговая программа, я приведу главные диалоговые команды:

- [Клавиша пробела] Показывает следующую страницу экрана.
- [d] Пролистывает 11 строк, или примерно половину обычного экрана из 25 строк.
- [/] Поиск регулярных выражений. Конечно, регулярное выражение может быть довольно сложным, но вы просто можете написать строчку, которую надо искать. Например, /toad и нажатие клавиши ввода будет искать следующее вхождение "toad" в вашем файле. При нажатии клавиши ввода после косой черты будет найдено следующее вхождение того, что вы ищете.
- [n] Также будет искать следующее вхождение регулярного выражения.
- [:][n] Если в командной строке вы указали более одного файла, эта команда переместит вас в следующий файл.
- [:][p] Переместит вас в предыдущий файл.
- [q] Выход из more.

head [-lines (строк)] [file1 file2 ... fileN]

head выводит первые десять строк перечисленных файлов, или первые десять строк стандартного ввода, если в командной строке не указано никаких файлов. Необязательное число представляет число строк, которые надо печатать. таким образом head -15 frog будет печатать первые 15 строк файла frog.

tail [-строк] [файл1 файл2 ... файлN]

Как и head, tail будет показывать только часть файла. Естественно, tail показывает только конец файла, или последние десять строк стандартного ввода. tail также имеет необязательную опцию, определяющую сколько строк надо показывать.

file [file1 file2 ... fileN]

Команда file пытается определить формат конкретного файла. Так как не все файлы имеют расширение или отличительные знаки, команда file выполняет некоторые элементарные проверки, чтобы попытаться выяснить, что это за файл.

Будьте осторожны, потому что возможно, что file неправильно определит формат файла.

## 7.5 Информационные команды

В этом разделе обсуждаются команды, которые изменяют файл, выполняют определенные действия над файлом, или показывают статистику файла.

grep [-nvwx] [-number (число)] expression (выражение) [file1 file2 ... fileN]

Одна из самых полезых команд в Unix - это grep, сокращенное (generalized regular expression parser - анализатор обобщенных регулярных выражений). Это имя утилиты, котороая может только искать текст в файле. Самый простой способ пользоваться grep:

```
/home/larry# cat animals
Animals are very interesting creatures. One of my favorite animals is
the tiger, a fearsome beast with large teeth.
I also like the lion---it's really neat!
/home/larry# grep iger animals
the tiger, a fearsome beast with large teeth.


/home/larry#
```

Один недостаток grep, это то, что хотя он показыает все строки, содержащие это слово, он не сообщает номер строки в файле. В зависимости от того, что вы делаете, это может оказаться достаточным. Например, если вы ищете 'error' в выходном потоке программы, попробуйте сделать так a.out | grep error, где 'a.out' имя вашей программы.

Если вам интересно знать, в каком именно месте расположены образцы, подходящие под шаблон, используйте опцию 'n', чтобы напечатать номера строк. Используйте опцию 'v', если вы хотите вывести все строки, в которых не содержатся образцы, подходящие под шаблон.

Другая отличительная черта grep, это то, что находятся

соответствия частям слов, как в примере выше, где iger'у соответствовл tiger. Для того, чтобы grep находил соответствия только целым словам, используйте опцию 'w', только целым строкам, используйте опцию 'x'.

Помните о том, что если вы не указали никакого файла, grep будет искать в стандартном вводе.

wc [-clw] [file1 file2 ... fileN]

wc - сокращенное word count (подсчет слов). Эта команда просто считает число слов, строк и символов в файле(ах). Если в командной строке не указано никаких файлов, команда работает со стандартным вводом.

Три параметра clw, сокращенное character (символ), line(строка), word(слово), соответственно, сообщают, что именно считать команде wc. Таким образом, wc -cw считает число символов и слов, но н считает число строк. По умолчанию, wc считает и слова, и строки, и символы.

Одно из самых красивых использований wc - выяснить, сколько файлов находится в текущей директории: ls | wc -w. Если вы хотите узнать, сколько файлов заканчивается на .c, сделайте так ls *.c | wc -w.

spell [file1 file2 ... fileN]

spell -это очень простая программа орфографической корректировки Unix, обычно используемая для американского английского. (Есть версии и для других европейских языков, но скорее всего на вашей машине находится версия для американского английского и только для него.) spell - это фильтр, как и большинство других программ, о которых мы говорили, которые берут текстовый ASCII файл, и выводит все слова, которые он считает неправильно написанными. spell работает с файлами, перечисленными в командной строке, или если не указано не одного файла, со стандартным вводом.

Более сложная программа орфографической корректировки, ispell, вероятно, тоже доступна на вашей машине. ispell предлагает возможное исправление орфографических ошибок и интерфейс с меню, если в командной строке указано имя файла, или исполнение программы, как фильтра, если не заданы имена файлов.

Работа с ispell должна быть достаточно очевидной; смотрите справочное руководство man в случае затруднений.

cmp file1 [file2]

Программа cmp сравнивает два файла. Первый файл должен быть указан в командной строке, второй файл может быть указан как параметр, или быть считан со стандартного ввода. cmp очень просто и ясно сообщает, где впервые различаются два файла.

diff file1 file2

Одна из самых сложных стандартных команд Unix называется diff. GNU версия diff имеет более двадцати опций! Это более мощная версия cmp, которая показывает вам различия, вместо того, чтобы просто сообщить, где находится первое отличие.

Так как разговор даже о части команды diff не запланирован в этой книге, я просто расскажу об основных операциях в diff. У diff есть два

файла, в качестве параметра, и команда показывает их различие построчно. Например:

```
/home/larry# cat frog
Animals are very interesting creatures. One of my favorite animals is
the tiger, a fearsome beast with large teeth.
I also like the lion---it's really neat!
/home/larry# cp frog toad
/home/larry# diff frog toad
/home/larry# cat dog
Animals are very nteresting creatures. One of my favorite animals is

the tiger, a fearsome beast with large teeth.
I also   like the lion---it's really neat!
/home/larry# diff frog dog
1c1,2
< Animals are very interesting creatures. One of my favorite animals is
---
> Animals are very nteresting creatures. One of my favorite animals is
>
3c4
< I also like the lion---it's really neat!
---
> I also   like the lion---it's really neat!
/home/larry#
```

Как вы можете видеть, diff ничего не выводит, когда два файла идентичны. При сравнении различных файлов, есть секция заголовков, 1c1,2 которая сообщает о том, что сравниваются строка 1 файла frog, расположенного слева и строки 1--2 файла dog, находящегося справа и какие различия найдены. Затем сравнивается строка 3 файла frog и строка 4 файла dog. Сначала может показаться странным, то что сравнивается строки с различными номерами, но это более эффективно, чем выводить строки файла до конца, если в одном из файлов лишний раз был нажат символ ввода.

## 8. Редактирование файлов с помощью Emacs.

### 8.1 Что такое Emacs?

Для того, чтобы что-либо сделать на компьютере, необходимо знать, как помещать текст в файлы и как изменять текст, уже находящийся там. Редактор как раз и является программой, служащей этим целям. Emacs - один из самых популярных редакторов, в частности, потому, что позволяет даже новичку получать конкретные результаты работы с ним. (Классический Unix'овский редактор, vi, описан в приложении C).

Чтобы изучить emacs вам нужно отыскать какой-нибудь файл с простым текстом (буквами, цифрами и т.п.), скопировать его в ваш домашний каталог (выполните, например, cp /usr/src/linux/README ~/README - мы не хотим менять исходный файл, поскольку он может содержать важную информацию), и запустить emacs для данного файла:

```
/home/larry# emacs README
```

(Разумеется, в случае, если вы решили скопировать файл /etc/rc, или /etc/inittab, или любой другой файл, подставьте имя этого файла

вместо README. Например, если вы копировали: cp /etc/rc ~/rc, то запускать emacs надо следующим образом: emacs rc.)

Emacs может работать по-разному в зависимости от того, как он запущен. При запуске с простой консоли, отображающей только текстовые символы, Emacs займет всю консоль под свое окно. Если вы запускаете его из X Windows, то Emacs откроет собственное окно. Я буду предполагать, что вы вызываете его с помощью текстовой консоли, но все это переносится и на случай использования X Windows - просто подставьте слово "окно" в тех местах, где я писал "экран". Также помните, что вам необходимо передвинуть курсор мыши в окно Emacs'а, чтобы набирать текст в нем!

Большая часть Вашего экрана после запуска Emacs содержит ваш текстовый документ, но последние две строки представляют особый интерес для начинающих изучать Emacs. Вторая из них (та, которая содержит длинную последовательность черточек) называется "строкой режима".

В моей строке режима вы видите слово "Top"("Вершина"). На его месте может быть слово "All"("Всё"), а также некоторые другие несущественные отличия. (Часто в строке режима отображается системное время.) Строка, находящаяся непосредственно за строкой режима называется мини-буфером или, иногда, строкой сообщений. Emacs использует её для того, чтобы отображать предназначенные вам сообщения, а также, в случае необходимости, для чтения вводимой вами информации. В данном случае, Emacs информирует вас о следующем: "For information about the GNU Project and its goals, type C-h C-p." ("Для получения информации о проекте GNU и его целях нажмите C-h C-p.") Пока проигнорируйте это - по началу мы не будем пользоваться минибуфером.

Прежде чем вы внесёте изменения в текст данного файла, вам необходимо научиться перемещаться по тексту. Курсор должен находиться в начале файла - в верхнем левом углу экрана. Чтобы переместить его вправо нажмите C-f (т.е. держите клавишу "Control" во время нажатия клавиши "f"("forward"). Курсор передвинется на один символ вправо, и, если вы будете продолжать нажимать эти клавиши одновременно, то примерно через пол-секунды автоматически произойдёт то же действие. Заметьте, что когда вы достигаете конца строки, то курсор автоматически переместится на начало следующей. Комбинация C-b ("backward") служит для перемещения в обратную сторону. Ну и, раз об этом зашла речь, комбинации C-n и C-p служат для перемещения курсора на следующую (next) и предыдущую (previous) строки соответственно.

Зачастую использование управляющих клавиш является наиболее быстрым способом перемещения курсора во время редактирования. Одна из целей Emacs'а - чтобы вы держали руки около алфавитно-цифровых клавиш клавиатуры, т.е. там, где и производится основная работа. Однако, если вы хотите, то вы можете перемещаться по тексту и с помощью клавиш-стрелок.

Когда вы используете X, для перемещения курсора в любом направлении вам достаточно переместить указатель мыши, нажав на её левую кнопку. Однако, это очень медленный способ - вам ведь каждый раз приходится подносить руку к мыши! Большинство пользователей Emacs'а для перемещения по тексту в первую очередь используют клавиатуру.

Всякий раз, когда захотите передвинуть курсор в левый верхний угол, используйте комбинации C-p и C-b. А теперь сохраните комбинацию C-b в нажатом состоянии немного дольше. Вы должны услышать сигнальный

звонок и увидите сообщение:"Beginning of buffer" ("Начало буфера"), которое появится в мини-буфере. Теперь вы конечно же поинтересуетесь: "Но что же такое буфер?"

Когда Emacs работает с каким-либо файлом, в действительности работы с самим этим файлом не происходит. Вместо этого он копирует содержимое данного файла в свою специальную рабочую область памяти, называемую буфером, где вы можете модифицировать его так, как вашей душе угодно. Когда работа проделана, вы с помощью Emacs'а сохраняете данный буфер - другими словами, записываете содержимое буфера в соответствующий файл. До тех пор, пока вы этого не сделали, файл остаётся неизменным, а содержимое буфера существует только внутри Emacs'а.

Учитывая вышесказанное, приготовьтесь внести ваш первый символ в буфер. До этого момента всё, что мы делали, не могло ничего испортить, и это очень существенный момент. Вы можете выбрать любой символ, какой вам нравится, но если вы хотите сделать это со вкусом, то я предлагаю использовать хорошую, жирную заглавную букву "X". Как только вы наберёте её, посмотрите на начало вашей строки режима внизу экрана. Как только вы измените буфер так, что его содержимое будет отличаться от содержимого файла на диске, Emacs отобразит две звёздочки в начале строки режима, чтобы дать вам знать, что буфер изменён:

```
 --**-Emacs: some_file.txt        (Fundamental)--Top-------------------
```

Эти две звёздочки отображаются сразу, как только вы измените содержимое буфера, и остаются видимыми до тех пор, пока вы не сохраните содержимое буфера. Вы можете сохранять буфер много раз во время редактирования - команда, служащая для этого: C-x C-s (держа [Control] нажмите "x" и "s" ... okay, итак, вероятно, вы научились и этому!). Намеренно сделали такую комбинацию, чтобы это было просто набирать, так как сохранять ваши буфера рекомендуется пораньше и почаще.

Я собираюсь предоставить вам список ещё нескольких команд, включая те, которые вы уже выучили. Теперь вы можете попрактиковаться с ними как только захотите. Я советую ознакомиться с ними прежде чем двигаться дальше:

```
  C-f         Передвижение вперёд (вправо) на один символ
  C-b         Передвижение назад  (влево)  на один символ
  C-n         Передвижение на следующую строку
  C-p         Передвижение на предыдущую строку
  C-a         Передвижение на начало строки
  C-e         Передвижение на конец строки
  C-v         Передвижение на следующую страницу/экран текста
  C-l         Перестроить экран так, чтобы текущая строка оказалась в
центре
  C-d         Стереть данный символ (попробуйте её)
  C-k         Стереть текст от текущей позиции и до конца строки
  C-x C-s     Записать буфер в соответствующий ему файл
  [Backspace] Стереть предыдущий символ (тот, который вы только что
набрали).
```

### 8.2 Как быстро начать работу в X

Если всё, что вас интересует, это быстрое редактирование сразу нескольких файлов, то пользователю X не требуется идти много дальше меню в верхней части экрана:

```
 Buffers  File  Edit  Help                                              ]
```

Эти меню недоступны в текстовом режиме.

Как только вы запустите Emacs, в верхней части экрана появятся четыре меню: Buffers, File, Edit и Help. Для того, чтобы воспользоваться меню, просто передвиньте указатель мыши к его заголовку (например, File), нажмите и держите левую кнопку. Затем передвиньте указатель к тому действию, которое вы захотите выполнить, и отпустите кнопку мыши. Если вы отказались от своих намерений, то отведите указатель мыши куда-нибудь в сторону и отпустите кнопку.

Меню Buffers содержит список различных файлов, которые вы редактируете в данном сеансе. Меню File предоставляет набор команд для загрузки и сохранения файлов - многие из них будут описаны позже. Меню Edit отображает несколько команд для редактирования одного буфера, а с помощью меню Help можно получить доступ к оперативной справочной информации.

Обратите внимание, что после пунктов меню перечислены их клавиатурные эквиваленты, вы можете захотеть выучить их, чтобы впоследствие работать быстрее. Кроме того, к лучшему это или к худшему, большая часть функциональных возможностей Emacs'а доступна только посредством клавиатуры, и поэтому, вы можете захотеть дочитать оставшуюся часть данной главы.

## 8.3 Редактирование сразу нескольких файлов

Emacs может работать одновременно с несколькими файлами. На самом деле, единственное ограничение на то, сколько буферов одновременно может обрабатывать Emacs, является количество доступной памяти на вашей машине. Команда, служащая для заведения буфера для нового файла: C-x C-f. Как только вы наберете ее, в мини-буфере появится запрос о имени файла:

```
Find file: ~/
```

Синтаксис здесь такой же, который используется для указания файлов в приглашениях shell'а; косые черты представляют подкаталоги, ~ означает ваш домашний каталог. Вы также получаете возможность "завершения имени файла", т.е, если вы набрали достаточно символов имени файла в данном приглашении для того, чтобы однозначно идентифицировать файл, вы можете просто нажать [Tab] для завершения его имени (или чтобы показать возможные варианты завершения в случае, если их более одного). Клавиша [Space] ("Пробел") также играет роль для завершения имени файла в мини-буфере, схожую с ролью клавиши [Tab], но я предложу вам проверить, в чем состоит разница. Как только в мини-буфере окажется полное имя файла, нажмите [Return] и Emacs предоставит буфер с содержимым данного файла. В Emacs'е этот процесс известен как поиск файла. Двигаемся дальше, сейчас найдите еще какой-нибудь не очень важный текстовый файл и загрузите его в Emacs'е (сделайте это из нашего начального буфера some_file.txt). Теперь вы имеете новый буфер. Я предположу, что он называется another_file.txt, т.к. я не могу видеть вашу строку режима.

Ваш первоначальный буфер внешне как-будто бы пропал, возможно вы заинтересовались: куда это он подевался? Он по-прежнему находится в

Emacs'е, и вы можете переключиться к нему с помощью комбинации C-x b. Когда вы напишите эту комбинацию, то увидите, что мини-буфер запрашивает вас о буфере, на который нужно переключиться, и содержит имя буфера, на который будет происходить переключение по умолчанию. Последнее означает, что если вы просто нажмете клавишу ввода [Return], не набирая имени буфера, то переключение произойдет на данный буфер. Этот буфер по умолчанию всегда является тем, который был переключен самым последним, т.е. если вы работаете с двумя буферами, комбинация C-x b всегда по умолчанию будет переключать на "другой" буфер (что сохраняет вас от необходимости набирания имени буфера). Даже если буфер по умолчанию и есть тот буфер, который вам нужен, всё же попробуйте ввести его имя.

Заметьте, что здесь также действует правило "завершения имени файла" во время его поиска: нажатие клавиши [Tab] завершает столько в имени буфера, сколько представляется возможным. Всякий раз, когда вы видите какое-нибудь приглашение в мини-буфере, полезно поверять, не может ли Emacs делать "завершение". Делая его каждый раз, когда это возможно, вы освободите себя от большого количества набирания на клавиатуре. Emacs обычно делает "завершение", когда вы выбрали один из пунктов некоторого предопределенного набора.

Всё, чему вы научились о передвижении по тексту, редактируя текст в первом буфере, применимо и ко второму, более новому. Будем двигаться дальше - измените текст в новом буфере, но не сохраняйте его (т.е. не нажимайте C-x C-s). Предположим, что вы не хотите сохранять внесенные вами изменения в файл. Команда, служащая этому - C-x k, которая "уничтожит" буфер. Наберите ее. Сначала вас спросят о том, какой же буфер уничтожить, при этом, текущий буфер есть "буфер по умолчанию", который почти всегда и является буфером, подлежащим уничтожению. Итак,

просто нажмите [Return]. Затем вас спросят о том, действительно ли вы хотите уничтожить данный буфер - Emacs всегда проверяет прежде, чем уничтожить какой-либо буфер, в который внесены изменения без их сохранения. Просто наберите "yes" и нажмите [Return] в случае, если вы хотите уничтожить его.

Двигайтесь дальше - попрактикуйтесь в загрузке, модифицировании, сохранении файлов, уничтожении соответствующих им буферов. При этом будьте уверены, что вы не портите какие-нибудь важные системные файлы, чтобы это не послужило причиной возможных повреждений, но все же попробуйте открыть по крайней мере пять буферов одновременно и, таким образом, вы сможете потренироваться в переключении между ними. (Если вы не зарегистрировались как администратор системы (пользователь root), то вы не сможете каким-нибудь образом повредить системе, но все равно, будьте внимательны.)

### 8.4 Завершение сеанса редактирования

Когда вы проделали всю работу с Emacs'ом, убедитесь, что все буфера, которые следовало сохранить, уже сохранены, и выйдите из Emacs'а с помощью комбинации C-x C-c. Иногда, после ее нажатия в мини-буфере появится один или два вопроса прежде, чем закончить работу с Emacs - не волнуйтесь, а просто ответьте на них. Если вы думаете, что можете вернуться в Emacs позднее, то не пользуйтесь комбинацией C-x C-c; наберите C-z, и она просто временно приостановит работу с Emacs'ом. После этого, вы можете вернуться к нему, используя команду Shell'a fg. Это более эффективный способ, чем завершение и перезапуск Emacs'а по нескольку раз, особенно, если вы позже будете редактировать те же самые файлы.

При работе в X нажатие C-z просто свернет окно в пиктограмму.

Смотрите раздел, посвященный этому в главе 5. Таким образом существует два способа сворачивания Emacs'а - обычный, который вам предоставляет ваш оконный администратор, и C-z. Помните, что когда вы свернули окно, то просто использование команды fg не развернет его - вам придется использовать ваш администратор окон.

## 8.5 Управляющая клавиша (мета-клавиша).

Вы уже знаете одну "клавишу-модификатор", используемую в Emacs'е, это клавиша [Control]. Существует и еще одна, называемая мета-клавишей, которая используется достаточно часто. Однако, не во всех клавиатурах она расположена в одном и том же месте, а в некоторых она может и вовсе отсутствовать. Первое, что вам необходимо сделать, это выяснить, где расположена ваша мета-клавиша. Возможно, что ваши клавиши [Alt] также являются мета-клавишами, если вы используете клавиатуру IBM PC или подобную, на которой имеются эти клавиши.

Способ проверить это - нажать клавишу, которая могла бы быть мета-клавишей, и нажать "x". Если после этого увидите, что в мини-буфере появится короткое сообщение (типа "M-x"), то вы нашли ее. Чтобы избавиться от этого сообщения и вернуться к вашему буферу наберите C-g.

Если же у вас не появилось сообщения, то все же остается еще одно решение. Вы можете использовать клавишу [Escape] как мета-клавишу. Но вместо того,чтобы держать ее в нажатом состоянии, пока вы нажимаете на следующую клавишу, нужно сразу же отпустить ее, а затем набрать следующий символ. Этот метод будет работать, даже если у вас нет настоящей мета-клавиши, таким образом, это самый простой способ двигаться дальше. Теперь попробуйте нажать [Escape], а затем клавишу "x". У вас должно появиться то же коротенькое сообщение. Чтобы оно исчезло, просто наберите C-g. Эта комбинация является общим способом в Emacs'е завершить работу с чем-либо, что в данный момент вас не интересует. Обычно она сопровождается звуковым сигналом, сообщая вам, что вы что-то прервали, но это и неплохо.

Обозначение M-x аналогично C-x (замените "x" любым другим символом). Если вы нашли настоящую мета-клавишу, то пользуйтесь ею, иначе просто используйте [Escape]. Я буду просто писать M-x, а вам придется пользоваться вашей собственной мета-клавишей.

## 8.6 Работа с блоками текста (вырезание, перемещение, удаление и вставка).

Emacs, как и любой другой хороший редактор, позволяет вам вырезать и перемещать блоки текста. Для того, чтобы делать это, вам нужно знать способ определения начала и конца блока. В Emacs'е вы делаете это, устанавливая в буфере два местоположения, называемах "отметка" (mark) и "точка" (point). Чтобы установить "отметку", переместите курсор к месту, с которого будет начинаться блок, нажмите C-SPC ("SPC" означает [Space], естественно). В мини-буфере должно появиться сообщение "Mark set" (установка отметки). Для некоторых терминалов C-SPC не работает, тогда вам придется пользоваться C-@. Теперь отметка установлена в данном месте. При этом не появится каких-либо особых выделений факта постановки отметки, ведь вы сами знаете, куда ее поставили.

А что насчет "точки"? На самом деле, вы устанавливаете ее всякий

раз, когда перемещаете курсор, т.к. "точка" просто привязывается к вашему текущему местонахождению в буфере. Говоря более формально, точка является тем местом, куда был бы вставлен текст, если бы вам нужно было что-нибудь набрать. Установив "отметку", а затем переместившись в конец текстового блока, вы фактически задали этот блок. Этот блок называется "регион" (region). Регион всегда будет означать область между отметкой и точкой.

Просто определение региона еще не позволяет его перемещать. Вы должны уметь говорить Emacs'у скопировать его, чтобы уметь перемещать его. Для того, чтобы скопировать регион, убедитесь, что отметка и точка правильно установлены, а затем нажмите M-w. Теперь Emacs записал его. Чтобы переместить его куда-либо в другое место, просто переместитесь туда и нажмите C-y. Эта операция называется вставкой (yanking) текста в буфер.

Если вы действительно хотите переместить текст региона куда-либо, нажмите C-w вместо M-w. Это "удалит" регион - весь текст внутри него исчезнет. На самом деле, он также был сохранен, как и при использовании M-w. Вы можете снова вставить его обратно с помощью C-y, как обычно. Место, в котором Emacs сохраняет весь такой текст называется "kill-ring" ("кольцо удалений"). В некоторых редакторах оно называется "clipboard" или "paste buffer".

Существует и другой способ вырезания и перемещения текста: всякий раз, когда вы пользуетесь C-k, чтобы уничтожить текст до конца строки, этот текст будет сохраняться в kill-ring. Если вы удалили более одной строки подряд, то все эти строки будут сохранены там вместе, так что очередное перемещение текста вставит эти строки все одновременно. По этой причине, часто быстрее несколько раз набрать C-k, чтобы чтобы уничтожить какой-то текст, нежели устанавливать отметку, точку, а затем нажимать C-w. Однако, оба этих способа будут работать. Это как раз случай личного предпочтения.

## 8.7 Поиск и замена текста.

Существует несколько способов поиска текста в Emacs'е. Некоторые из них достаточно сложные и не стоят того, чтобы здесь их касаться. Самый простой и приятный способ - это использование isearch. "Isearch" означает "incremental search" ("наращиваемый поиск"). Предположим, что вы хотите отыскать строку "gadfly" в буфере, содержащем следующий текст:

```
I was growing afraid that we would run out of gasoline, when
my passenger exclaimed ``Gadzooks!  There's a gadfly in here!''.
```

Вы переместились бы к началу буфера, или, по крайней мере, к той точке, о которой вы точно знаете, что она находится до первого вхождения целевого слова "gadfly", а затем нажмите C-s. Это переведет вас в режим isearch. Теперь начните набирать слово, которое вы ищете, "gadfly". Но как только вы наберете "g", вы увидите, что что Emacs переместил вас к первому вхождению "g" в данном буфере. Если представленный текст является всем содержимым буфера, то это будет первый символ в слове "growing". Теперь наберите букву "a", очередную букву слова "gadfly", и Emacs переместит вас к слову "gasoline", которое содержит первое вхождение "ga". Набрав еще "d", вы переместитесь к "gadzooks", и, наконец, "f" переместит вас к "gadfly", не заставляя вас набирать слово целиком.

Когда вы находитесь в режиме isearch, вы определяете строку для поиска. Каждый раз, когда вы добавляете новый символ к концу строки,

число ее вероятных вхождений в текст уменьшается, пока вы не набрали

достаточно символов, чтобы однозначно определить строку. Как только вы нашли интересующее вас вхождение, вы можете покинуть isearch, просто нажав [Return] или какую-нибудь из обычных команд перемещения. Если вы думаете, что строка, которую вы ищите находится выше, то вам следует пользоваться C-r, что заставляет isearch работать в обратном направлении.

Если вы нашли вхождение, но оно оказалось не то, что вы искали, то снова наберите C-s, все еще находясь при этом в режиме isearch. Это передвинет вас вперед к следующему вхождению, всякий раз, когда вы нажимате C-s. Если следующего вхождения не оказалось, то появится сообщение, что поиск завершился неудачно, но если вы снова наберете C-s, то он возобновится с самого начала буфера. Обратное верно для C-x - она продолжит поиск от конца к началу буфера.

Попытайтесь загрузить буфер с простым английским текстом и запустите isearch для строки "the". Предварительно вы можете вставить его сколько угодно раз, затем нажимайте C-s, чтобы перемещаться ко всем его вхождениям. Обратите внимание, что вы будете перемещаться ко всем словам, таким как "them", поскольку оно также содержит подстроку "the". Для того, чтобы искать только "the", вам следует добавить пробел к концу искомой строки. Вы можете добавить другие символы к этой строке в любой момент поиска, даже если вы подряд нажали несколько раз C-s, чтобы отыскать очередные вхождения. Вы также можете пользоваться [Backspace] или [Delete], чтобы удалить символы из строки в любой момент поиска, а нажатие клавиши [Return] приведет к завершению поиска, оставив вас у последнего вхождения целевой строки.

Emacs также позволяет вам замещать все вхождения какой-то строки некоторой новой строкой - это называется query-replace (замещение по запросу). Чтобы вызвать его, наберите query-replace и нажмите [Return]. Поскольку на имена команд действует механизм завершений, то как только вы наберете "query-re", можете просто нажать [Tab], чтобы завершить его. Допустим, вы хотите заменить все вхождения "gadfly" на "housefly". В ответ на приглашение "Query-replace:" наберите "gadfly" и нажмите [Return]. Затем снова появится запрос и вам нужно будет ввести "housefly". Emacs начнет передвигаться по буферу, останавливаясь на каждом вхождении слова "gadfly", и спрашивая вас,

хотите ли вы заменить это вхождение. Просто нажмите "y" или "n" при каждом запросе, говоря "Yes" или "No", соответственно, так до самого конца. Если вы не уловили смысл, пока читали это, то попробуйте проделать это сами.

## 8.8 Что же происходит в действительности.

На самом деле, все эти клавишные комбинации, которые вы изучаете, являются сокращениями функций Emacs'а. Например, C-p - короткий путь вызова встроенной функции Emacs'а "previous-line" (переход на предыдущую строку). Однако, все эти внутренние функции могут быть вызваны и при помощи их имен, используя комбинацию M-x. Если вы забыли, что функции previous-line соответствует комбинация C-p, то вы могли бы просто набрать: M-x previous-line [Return], и вы передвинетесь на одну строку вверх. Попробуйте сделать это, чтобы понять, что "M-x previous-line [Return]" и C-p в дейтвительности одно и то же.

Разработчик Emacs'а и начал с того, что сначала определил полный набор встроенных функций, поставив затем в соответствие наиболее часто используемым из них клавишные комбинации. Иногда легче просто вызвать функцию явно с помощью M-x, чем помнить, какая комбинация ей соответствует. Например, функции query-replace (перенос по запросу) в некоторых версиях Emacs'а соответствует комбинация M-%. Но кто может помнить такие сочетания? Если вы не используете функцию query-replace крайне часто, то легче просто вызвать ее с помощью M-x.

Большая часть того, что вы набираете, это символы, вставляемые в текст буфера. Т.е. каждой из этих клавиш соответствует функция self-insert-command, которая не делает ничего другого, кроме как вставляет данный символ в буфер. Комбинации, в которых участвует клавиша [Control] с каким-нибудь символом, обычно соответствуют функциям, которые производят различные действия, такие как перемещения по тексту. Например, комбинация C-v соответствует функции scroll-up, которая прокручивает (делает скролинг) буфер вверх на один экран (имеется ввиду, конечно, что ваша позиция при этом передвигается

вниз).

Допустим, вы хотите вставить какой-нибудь управляющий символ в буфер, как вы собираетесь делать это? Кроме всего прочего, управляющие символы являются ASCII символами, хотя и редко используемыми, но вы можете захотеть внести их в файл. Существует способ избежать того, что управляющие символы интерпретировались Emacs'ом как команды. Комбинация клавиш C-q соответствует функции quoted-insert. Данная функция читает очередную клавишу и вставляет ее символьное значение в буфер, не пытаясь интерпретировать ее как команду. Это был способ внесения управляющих символов в ваш файл, используя Emacs. Естественно, для того, чтобы вставить в файл C-q - это нажать C-q дважды!

Также Emacs имеет множество функций, которым не соответствуют какие-либо клавиши. Например, если вы пишите длинное сообщение, то вы не захотите нажимать клавишу ввода в конце каждой строки. Можете сделать это с помощью Emacs (вы можете сделать все, что угодно с помощью Emacs) - команда, используемая для этого auto-fill-mode, но для нее нет соотвествующей клавишной комбинации. Для того, чтобы вызвать эту функцию, вам следует набрать "M-x auto-fill-mode". Комбинация M-x используется для вызова функции по ее имени. Вы даже могли бы вызвать по имени такие функции, как next-line и previous-line, но это было бы крайне неэффективно, так как им уже соответствуют клавишные комбинации C-n и C-p!

Кстати, если вы после вызова функции auto-fill-mode посмотрите на вашу строку режима, вы заметите, что к ее первому краю добавилось слово "Fill". Пока это слово находится там, Emacs будет размещать текст (заворачивая строку по достижении правого края) автоматически. Вы можете отключить его, набрав "M-x auto-fill-mode" еще раэ - это команда с двумя состояниями.

Неудобства, связанные с набиранием в мини-буфере длинных имен функций, уменьшено за счет того, что Emacs делает завершение имени функции таким же образом, каким он делал это для имен файлов. Поэтому, вам редко придется набирать целиком имя функции, символ за символом. Если вы не уверены, можно или нет делать завершение, просто нажмите

клавишу [Tab]. Это никак вам не повредит: самое худшее, что может случиться, это то, что вы просто поставите символ табуляции, а если вам повезло, то окажется, что вы можете использовать завершение.

## 8.9 Использование подсказок (HELP) в Emacs'е.

Emacs имеет расширенные возможности помощи - настолько расширенные, что здесь мы можем их только слегка затронуть. Основные возможности системы подсказок доступны при помощи комбинации: C-h и одиночный символ. Например, C-h k дает информацию о любой клавише (она выдает приглашение нажать клавишу (или комбинацию клавиш), а затем сообщает, для чего она служит). Комбинация C-h t предоставляет небольшое средство обучения пользователя Emacs'у. Наиболее важно, что C-h C-h выдает подсказки о том, как пользоваться всей системой помощи (help on help) . Если вы знаете имя какой-нибудь функции Emacs'а (например, save-buffer), но не можете вспомнить, с помощью какой последовательности клавиш она вызывается, то наберите C-h w и имя данной функции. Или, если вы хотите более подробно знать, что делает данная функция, то пользуйтесь комбинацией C-h f, которая запрашивает имя интересующей вас функции.

Запомните, поскольку Emacs делает завершение имен функций, то на самом деле вам не обязательно помнить, как именно называется функция, о которой вы хотите получить подсказку. Если вы думаете, что можете угадать имя функции по ее начальным символам, то наберите ее и нажмите [Tab] и посмотрите, не произошло ли ее завершение. Если нет, то вернитесь назад и попробуйте какой-нибудь другой вариант. То же самое применимо и для имен файлов: если вы не можете вспомнить точного имени файла, к которому вы не обращались примерно три месяца, вы можете попробовать угадать его и применить "завершение". Таким образом, использование "завершения" не просто сокращает количество нажатий клавиш, но и представляет собой средство ответов на ваши вопросы.

Существуют и другие символы, которые вы можете набирать после C-h, и все они дают информацию о разных вещах. Наиболее часто используемые из них: C-h k, C-h w и C-h f. Как только вы глубже ознакомитесь с Emacs'ом, то попробуйте еще C-h a, которая запрашивает вас о строке, а затем сообщает вам о всех функциях, имена которых

содержат данную строку (например, "a" содержится в "apropos" или "about").

Еще одним источником информации является программа просмотра документации Info. Она слишком сложна, чтобы ее затрагивать здесь, но если вы сами захотите исследовать ее, то нажмите C-h f и прочитайте параграф в верхней части экрана. Это даст вам дополнительную информацию.

## 8.10 Специализированные буфера: Режимы

У буферов Emacs'а существуют режимы, ассоциированные с ними. Причиной этого является то, что ваши требования к подготовке документа, когда вы пишите почтовое сообщение, сильно отличаются от требований, скажем, к составляемой программе. Вместо того, чтобы пытаться работать с редактором, который реагировал бы на каждое отдельное требование(что было бы весьма сложно), разработчик Emacs'а решил заставить Emacs вести себя различным образом, в зависимости от того, что вы делаете в каждом выделенном буфере. Таким образом, буферы имеют режимы, каждый из которых спроектирован для некоторой специфической деятельности. Основные особенности, отличающие один режим от другого, - это комбинации клавиш, но если и другие, не менее важные отличия.

Основным режимом является "fundamental", который на самом деле не имеет никаких специфических команд. Вот то, что Emacs'у приходится

сообщать о нем:

Fundamental Mode:

Этот основной режим не предназначен для чего-либо конкретного. Другие основные режимы определяются путем сравнения с ним.

Я получил эту информацию таким образом: набрал C-x b, т.е. переключил буфер, и в ответ на приглашение ввести имя буфера, на который следует переключиться, я ввел "foo". Так как до этого момента не существовало буфера с таким именем, Emacs создал его и переключил меня на него. По умолчанию был установлен режим "fundamental", а если

бы это было не так, то я мог бы набрать "M-x fundamental mode", чтобы установить его. Для всех названий режимов имеется команда, называемая "<имя режима>-mode", переводящая текущий буфер в данный режим. Затем, чтобы получить больше информации об этом основном режиме, я набрал "C-h m", которая выдает вам информацию о текущем режиме буфера, с которым вы работаете.

Существует чуть более полезный режим, называемый "text-mode", в котором есть специальные команды M-S, для центрирования параграфа, и M-s, для центрирования строки. Кстати, обозначение M-S означает ровно то, что вы думаете, т.е. нажмите обе клавиши [Meta] и [Shift], а затем, держа их, нажмите "S".

Только не проделывайте сразу все сказанное - сначала создайте новый буфер, переведите его в текстовый режим "text-mode" и наберите C-h m. Возможно, что вы не поймете всей информации, которую выдаст Emacs, когда проделаете это, но все же вам следует попытаться хотя бы частично разобраться в ней.

Этот и следующий раздел являются введением в некоторые наиболее используемые режимы. Работая с ними, вы можете в любой момент нажать C-h m для получения дополнительной информации.

## 8.11 Режимы составления программ (Programming Modes).

### 8.11.1 Режим C-mode

Если вы используете Emacs для составления программ на языке C, то вы можете делать все отступы автоматически. Файлы, чьи имена оканчиваются на ".c" или ".h", автоматически редактирутся в этом режиме. Это означает, что становятся доступными определенные команды редактирования, полезные для составления программ на языке C. В режиме C-mode клавиша [Tab] соответствует команде "c-indent-command". Т.е. при нажатии [Tab] на самом деле не происходит вставки символа табуляции. Вместо этого, если вы нажимаете [Tab] в каком-либо месте строки, Emacs автоматически разместит данную строку соответсвующим образом (сделает отступ) в программе. Это говорит о том, что Emacs кое-что знает о синтаксисе языка C (хотя совсем ничего не знает о его

семантике, т.е. не может уберечь вас от возможных ошибок!)

При этом предполагается, что предыдущие строки размещены правильным образом. Т.е. если в предыдущей строке пропущена скобка, точка с запятой, фигурная скобка или что-нибудь другое, то Emacs разместит следующую строку несколько забавным образом. Если вы это заметите, то проверьте пунктуацию в предыдущей строке.

Вы можете использовать эту особенность для проверки пунктуации в

ваших программах, вместо того, чтобы просматривать всю программу целиком, высматривая ошибки, просто, начиная с самого верха, делайте отступы с помощью [Tab] и когда у какой-то строки получится неверный отступ, то проверьте строки, которые находятся непосредственно перед ней. Другими словами, дайте Emacs'у работать на вас!

### 8.11.2 Режим Scheme

Этот важный режим не даст вам никаких преимуществ до тех пор, пока у вас не появится компилятор или интерпретатор для языка программирования Scheme. Использование последнего не является настолько же обычным как использование, скажем, C-компилятора, однако, оно становится все более и более всеобщим, поэтому, я его также затрону. Большее из того, что подходит для схемного режима подходит для Lisp-режима в случае, если если вы предпочитаете писать на языке Lisp.

У Emacs'а есть два различных режима Scheme. Один из них, о котором я расскажу, называется "cmuscheme", а позже, в разделе о настройках Emacs'а, о том, как вообще может быть два различных схемных режима и как с ними работать. А сейчас не бейспокойтесь о том, если не все, о чем я говорю здесь, подходит к вашему Emacs'у. Настраиваемый редактор значит непредсказумый редактор, и от этого никуда не деться!

Вы можетете запустить интерактивный схемный режим в Emacs'е с помощью "M-x run-scheme". Это создаст буфер с именем "*scheme*", который содержит в себе обычный схемное приглашение на ввод. Вы можете набирать в нем схемные выражения и нажать клавишу ввода. После

обработки введенного будет выдан результат. Таким образом, для того, чтобы работать интерактивно со схемным процессом, вы могли бы просто набирать все определения и описания функций через данное приглашение. Возможно, что у вас уже есть ранее написанный исходный схемный код в некотором файле и было бы легче проделывать всю работу в этом файле, а затем по необходимости пересылать определения в буфер схемного процесса.

Если этот исходный файл оканчивается на ".ss" или на ".scm", то он автоматически будет загружен в схемном режиме, когда вы откроете его с помощью C-x C-f. Если, по каким-либо причинам, он оказался в другом режиме, то вы можете сделать это, набрав "M-x scheme-mode". Этот схемный режим scheme-mode не тот же самый, когда запускается буфер со схемным процессом; более того, буфер с исходным кодом, будучи в режиме scheme-mode, имеет специальные команды для взаимодействия с буфером схемного процесса.

Если вы находитесь внутри определения функции в буфере исходного схемного файла и наберете C-c C-e, то это определение будет "передано" в буфер процесса - в точности так, как если бы вы сами его туда ввели. C-c M-e перешлет определение и, затем, переключит вас в буфер схемного процесса для совершения некоторой интерактивной работы. C-c C-l загружает файл схемного кода (последняя команда работает и для буфера схемного процесса, и для буфера с исходным схемным кодом). И также, как и в других режимах составления программ, нажатие клавиши [Tab] произведет корректный отступ очередной строки.

Если вы находитесь внутри приглашения в буфере схемного процесса, вы можете использовать команды M-p и M-n, чтобы пролистать ваши предыдущие команды (т.е. input history). Т.е. если вы отлаживаете функцию 'rotate' и уже применяли ее к аргументам в буфере схемного процесса, например, так:

```
> (rotate (a b c d e))
```

то вы можете снова запустить эту команду, набрав M-p в приглашении. Нет нужды заново набирать длинные выражения в приглашении схемного процесса - приобретя привычку использовать "input history", вы

сэкономите большое количество времени.

Emacs "знает" достаточно мало языков программирования: C, C++, Lisp and Scheme - только некоторые из них. Вообще говоря, он "знает", как размещать текст во время составления программ, только "интуитивно".

### 8.11.3 Режим "Mail" (почтовый)

Кроме всего прочего с помощью Emacs'а вы можете редактировать и посылать почтовые сообщения. Для того, чтобы открыть почтовый буфер, наберите C-x m. Вы должны заполнить два поля: "To:" ("Кому:") и "Subject:" ("Назначение:"), а затем нажмите C-n, чтобы внизу появилась разделительная черта в теле сообщения (которая сначала будет пуста). Не изменяйте и не стирайте ее, т.к. иначе Emacs будет не в состоянии послать ваше собщение - он использует эту строку для отделения заголовка сообщения, который содержит информацию о том, куда послать сообщение, от его дейсвительного содержимого.

Ниже этой разделительной черты вы можете набирать все, что угодно. Когда будете готовы отправить сообщение, просто наберите C-c C-c, и Emacs отправит его, а после этого закроет почтовый буфер.

## 8.12 Как работать еще более эффективно

Опытные пользователи Emacs'а очень щепетильны в отношении эффективности. По сути дела, все заканчивается тем, что им приходится тратить достаточно много времени на поиск более эффективных способов. Мне не хотелось бы, чтобы это случилось и с вами, поскольку, есть несколько простых советов, следуя которым вы станете более квалифицированным пользователем Emacs'а. Иногда опытные пользователи ставят в глупое положение новичков, которые, по какой-то причине, не знают всех этих трюков - люди начинают заботиться о том, как "правильно" пользоваться Emacs'ом. Мне следовало бы осудить этот вид элитизма, если бы у меня самого не было "рыльцо в пушку". Итак:

Когда вы передвигаетесь по тексту, используйте наиболее быстрые доступные средства. Зная, что C-f это "вперед на символ", вы можете догадаться, что M-f это "вперед на слово". C-b это "назад на символ". Догадайтесь, что такое M-b? Однако, это не все: вы можете передвинуться вперед на одно предложение, когда после его конца стоит два пробела (Иначе Emacs будет не в состоянии разобраться, где кончается предыдущее и начинается следующее предложение). M-a означает "назад на предложение".

Если вы застали себя за повторным нажатием C-f для того, чтобы добраться до конца строки, то вам должно быть стыдно, убедитесь, что во вместо этого вы пользуетесь C-e и C-a, чтобы добраться до начала строки. Если вы многократно пользуетесь C-n, чтобы передвинутся вниз на один экран, то вам должно быть не менее стыдно, и далее всегда пользуйтесь C-v. То же самое касается перемещения на экран вверх -

вместо C-p используйте M-v.

Если вы находитесь вблизи конца строки и обнаруживаете, что у вас где-то ранее в строке есть опечатка или пропущено слово, не пользуйтесь [Backspace] или [Delete], чтобы добраться до этого места. Это потребует перепечатывания целых порций хорошего текста. Вместо этого пользуйтесь комбинациями M-b, C-b и C-f, чтобы добраться до нужного места, исправьте ошибку, а затем используйте C-e, чтобы вернутся к концу строки.

Когда вам надо вписать имя файла, никогда не пишите полное имя. Напечатайте столько, чтобы его можно было автоматически однозначно идентифицировать, и и примените механизм "завершения" нажатием [Tab] или [Space]. Незачем лишний раз нажимать клавиши, если можно воспользоваться циклами ЦПУ.

Если вы печатаете какой-нибудь простой текст, и ваша функция автопереноса была отключена, используйте M-q, что означает "заполнить абзац" в обычных текстовых режимах. Это расположит абзац, в котором вы находитесь так, как если бы был произведен перенос в каждой строке, не заставляя вас это делать вручную. M-q будет работать одинаково независимо от того, где вы находитесь - в середине, в начале или в конце.

Иногда удобно пользоваться функцией C-x u ("Undo"), которая попытается убрать все последние изменения. Emacs догадается, насколько "убрать", обычно он делает это очень деликатно. Повторяя это многрактно, изменений будет убираться все больше и больше, пока Emacs будет "понимать", что делать.

## 8.13 Настройка Emacs

Emacs настолько большой и сложный, что имеет даже собственный язык программирования! Я не шучу, чтобы настроить Emacs в соотвествие с вашими требованиями, вам пришлось бы писать программы на этом языке. Он называется Emacs Lisp, и это диалект Lisp'а. Поэтому, если вы имели предыдущий опыт работы на Lisp'е, он вам покажется достаточно знакомым. Если нет - не волнуйтесь, я не собираюсь значительно углубляться, так как он лучше познается в работе. Чтобы действительно узнать о программировании Emacs'а, вам пришлось бы обратиться к справочной информации по Emacs Lisp и просмотреть множество программ на нем.

Большинство функциональных возможностей Emacs'а определено в файлах, написанных на Emacs Lisp. Большая их часть распространяется вместе с Emacs'ом и все это в совокупности называется "Emacs Lisp Library". Расположение этой библиотеки зависит от того, каким образом Emacs был инсталирован в вашей системе, обычно располагается в /usr/lib/emacs/lisp, /usr/lib/emacs/19.19/lisp и т.п.. "19.19" - номер версии Emacs'а, и он может отличаться в вашей системе.

Вам не нужно просматривать вашу файловую систему в поисках этой библиотеки, потому что Emacs хранит информацию об этом в переменной load-path. Для того, чтобы выяснить значение этой переменной, ее необходимо "вычислить"; т.е. довести до интерпретатора Emacs Lisp ее значение. Существует специальная функция для вычисления выражений Lisp'а в Emacs'е, называемая lisp-interaction-mode (режим взаимодейсвия с Lisp'ом). Обычно в этом режиме имеется буфер, называемый "*scratch*". Если вы не можете найти его, создайте новый буфер под любым именем и наберите "M-x lisp-interaction-mode" в нем.

Теперь у вас появилось рабочее пространство для взаимодействия с интерпретатором Emacs Lisp'а. Наберите следующее:

```
load-path
```

а затем в конце нажмите C-j. В этом режиме lisp-interaction-mode комбинация C-j связывается с eval-print-last-sexp. "Sexp" означает S-expression (S-выражение), т.е. просто "сбалансированная" группа круглых скобок, ничего не содержащая. Пожалуй, это слишком просто звучит, но вы почуствовали бы, что это означает, если бы работали с Emacs Lisp. В любом случае, вычисление load-path даст вам примерно следующее:

```
 load-path C-j
("/usr/lib/emacs/site-lisp/vm-5.35" "/home/kfogel/elithp"
 "/usr/lib/emacs/site-lisp" "/usr/lib/emacs/19.19/lisp")
```

Разумеется это не будет выглядеть одинаково в разных системах, поскольку, это зависит от того, как Emacs был установлен. Пример, приведенный выше, взят с моего 386 PC, на которой работает LINUX. Как показано выше, load-path - это некий список строк. Каждая строка в этом списке представляет собой директорию, которая может содержать файлы на Emacs Lisp'е. Когда Emacs'у требуется загрузить файл с Lisp'овским кодом, он ищет его во всех этих директориях по очереди. Если директория указана, а фактически не существует, то Emacs просто игнорирует ее.

Когда Emacs начинает работу, он автоматически пытается загрузить файл .emacs, содержащийся в вашем личном каталоге. Соответственно, если вы хотите иметь собственные настройки Emacs'а, то вам следует поместить их в файл .emacs. Наиболее часто используемые настройки - это реакции на различные комбинации клавиш, вот как их задают:

```
 (global-set-key "\C-c l" 'goto-line)
```

global-set-key - это функция двух аргументов: клавишная комбинация и соответствующее ей действие. Слово "global" означает, что

эта комбинация будет действовать во всех основных режимах (сущесвует и другая функция, local-set-key, которая задает комбинацию, действующую в пределах одного буфера). Выше я поставил в соответствие C-c l функции goto-line. Эта комбинация описана с помощью некоторой строки. Специальный синтаксис "\C-<char>" означает нажатую клавишу [Control] во время нажатия клавиши <char>. Аналогично, "M-<char>" для мета-клавиши.

Все это хорошо, но как я мог знать, что имя этой функции это "goto-line"? Я могу знать, что хочу сопоставить комбинацию C-c l некоторой функции, которая запрашивает номер строки, а затем перемещает курсор на эту строку, но но как же я узнаю имя этой функции?

Вот здесь как раз самый подходящий случай воспользоваться средствами встроенной помощи. Как только вы определились с тем, какую функцию ищете, вы можете использовать Emacs, чтобы узнать ее точное имя. Вот самый быстрый и "грязный" способ: поскольку Emacs делает завершение имен функций, просто наберите C-h f (что, как вы помните, описывает функцию), а затем просто нажмите [Tab]. Это является запросом для Emacs'а сделать завершение для пустой строки - другими

словами, это завершение будет соответствовать любой простой функции! Это может занять немного времени, поскольку Emacs имеет так много встроенных функций, но он отобразит их столько, сколько позволит ваш экран.

Сейчас нажмите C-g, чтобы выйти из "описывателя" функций. Появится буфер, называемый "*Completions*" ("*Завершения*"), который содержит только что сгенерированный список завершений. Переключитесь в этот буфер. Теперь вы можете использовать C-s для поиска похожих функций. Например, можно смело предположить, что функция, которая запрашивает о номере строки и переводит курсор, содержит в своем имени слово "line". Таким образом, просто найдите строку "line", и, в конце концов, найдете то, что искали.

Если вы предпочитаете другой метод, пользуйтесь C-h a, команда apropos, чтобы показать все функции, чьи названия содержат данную строку. На мой взгляд, результат работы команды apropos немного

труднее упорядочить, чем просто поиск в списке завершений, но вам может показаться иначе. Попробуйте оба метода.

Всегда существует возможность, что у Emacs'а нет предопределенной функции, делающей именно то, что вам нужно. В данном случае вам придется написать функцию самому. Я не собираюсь рассказывать о том, как это делать, вам придется обратиться к библиотеке Emacs Lisp Library за примером определения функции и прочитать справочную информацию Info pages о Emacs Lisp. Если вы знаете местного "Guru" по Emacs'у, спросите его, как это делается. Определение собственных функций в Emacs'е - не такое уж трудное дело, кстати, в прошлом году я написал сто тридцать одну, или около того. Это требует небольшой практики, но кривая обучения совсем не является крутой.

Следующее, что люди делают в их собственном .emacs'е, это придание определенным переменным предпочтительных значений. Например, запустите заново Emacs, предварительно поместив в ваш файл .emacs следующее:

```
(setq inhibit-startup-message t).
```

Emacs проверяет значение переменой inhibit-startup-message, для того, чтобы решить, выводить ли при запуске определенную информацию по версии и сообщение об отсутствии всяческих гарантий. Выражение на Lisp'е, приведенное выше, использует команду setq для присвоения данной переменной значения "t", что означает логичение значение "истина" в Lisp'е. Противоположное к "t" это "nil", что означает значение "ложь" в Emacs Lisp'е. Вот две вещи из моего .emacs, которые вы можете счесть полезными:

```
(setq case-fold-search nil); поиск осуществляется в любом случае
;; делает отступ в программах на C таким, каким нужно:
(setq c-indent-level 2)
```

Первое выражение делает поиск независимым от случая, включая isearch, т.е. поиск будет производиться как для верхнего, так и для нижнего регистров, даже если искомая строка содержит только символы в нижнем регистре. Второе выражение устанавливает отступ по умолчанию для выражений на языке C немного меньшим, чем обычно (это кому как

нравится), я нахожу, что это облегчает чтение C-кода.

Символ комментария в Lisp'е это ";". Emacs игнорирует все, что следует за ней, если только он не встречается в строке символов

следущим образом:

```
;; these two lines are ignored by the Lisp interpreter, but the
;; s-expression following them will be evaluated in full:
(setq some-literal-string "An awkward pause; for no purpose.")
```

Это неплохая идея, отмечать комментариями ваши изменения в файлах на Lisp'е, поскольку через пол года вы не будете помнить, что вы имели в виду, модифицируя их. Если комментарии появляются в строке сами по себе, то им должны предшествовать ";;". Это помогает Emacs'у правильно осуществлять отступ в файлах на Lisp'е.

Вы можете узнать о внутренних переменных Emacs'а теми же способами, что и о функциях. Пользуйтесь C-h v, describe-variable, для составления списка завершений, или C-h C-a, apropos. Apropos отличается от C-h a, command-apropos, тем, что он показывает не только функции, но и переменные.

Расширение по умолчанию в файлах на Emacs Lisp'е - ".el", как в "c-mode.el". Однако, для того, чтобы программы на Lisp'е работали быстрее, Emacs позволяет ее побайтно откомпилировать, и данные откомпилированные файлы оканчиваются на ".elc" вместо ".el". Исключение составляет файл .emacs, который не требует расширения ".el", так как Emacs итак знает, что именно его следует искать при запуске.

Для того, чтобы загрузить файл с кодом на Lisp'е в интерактивном режиме, наберите M-x load-file. Это приведет к выдаче запроса о имени файла. Чтобы загружать Lisp'овские файлы из других Lisp'овских файлов, делайте следующее:

```
(load "c-mode") ; заставить Emacs загрузить код из файла c-mode.el или
                ;; c-mode.elc
```

Сначала Emacs добавит к указанному в команде имени расширение .elc и попытается найти файл с таким именем по путям, указанным в переменной load-path. В случае неудачи он пробует то же самое проделать с расширением .el; если же и это не принесло результатов, то он пропускает эту символьную строку. Вы можете побайтно откомпилировать файл с помощью команды M-x byte-compile-file, но если вы часто модифицируете файл, то, возможно, этого не стоит делать. Вам никогда не следует побайтно компилировать ваш .emacs, как, впрочем, и давать ему расщирение .el.

После того, как был загружен ваш .emacs, Emacs пытается загрузить файл под именем default.el. Обычно он располагается в одной из директорий, хранимых в переменной load-path, которая называется site-lisp, или local-elisp, или нечто вроде этого (см. выше пример для load-path). Пользователи, которые овладевают Emacs'ом в многопользовательских системах, пользуются default.el для внесения изменений, которые влияют на Emacs всех пользователей, поскольку, их Emacs каждого загружает этот файл после их персонального .emacs. Также не стоит побайтно компилировать default.el, т.к. он достаточно часто подвергается модификациям.

Если ваш .emacs содержит некоторые ошибки, Emacs вместо попытки загрузить default.el просто остановится, высвечивая сообщение "Error in init file" ("Ошибка в файле инициализации"). Если вы увидели подобное сообщение, то, вероятно, что-то не в порядке с вашим .emacs.

Есть еще один выражения, который часто используется в .emacs.

Emacs Lisp Library иногда предлагает многофункциональные пакеты для осуществления одних и тех же вещей различными способами. Это означает то, что вы должны определить, какой вы хотите использовать, иначе, вы получите пакет по умолчанию, который далеко не всегда наиболее соответствует всем вашим целям. Одна из областей, где случается подобное, это Scheme-режим Emacs'а (смотрите выше). Существует два различных интерфейса, распространяемых вместе с Emacs'ом (по крайней мере, в 19-ой версии) : x-scheme и cmuscheme.

```
prompt> ls /usr/lib/emacs/19.19/lisp/*scheme*
/usr/lib/emacs/19.19/lisp/cmuscheme.el
/usr/lib/emacs/19.19/lisp/cmuscheme.elc
/usr/lib/emacs/19.19/lisp/scheme.el
/usr/lib/emacs/19.19/lisp/scheme.elc
/usr/lib/emacs/19.19/lisp/xscheme.el
/usr/lib/emacs/19.19/lisp/xscheme.elc
```

Мне гораздо больше понравился интерфейс, предлагаемый cmuscheme, чем x-scheme, но Emacs по умолчанию пользуется последним. Как я могу заставить Emacs работать согласно моим требованиям? Я помещаю в мой .emacs следующее:

```
;; Заметьте, как выражение может быть разбито на две строки. Lisp просто
;; игнорирует пустые места:
( autoload 'run-scheme "cmuscheme"
"Run an inferior Scheme, the way I like it." t)
```

Функция autoload берет имя функции(отмеченной знаком " ' ",в связи с соглашениями, принятой в Lisp'е) и сообщает Emacs'у, что данная функция описана в определенном файле. Файл является вторым аргументом, который представлен строкой (без расширений ".el" и ".elc"), являющейся именем данного файла, который следует искать по путям, указанным в переменной load-path.

Остальные аргументы являются альтернативными, необходимыми лишь в случае, когда третий аргумент - документационная строка для этой функции, так что, если вы вызываете для нее describe-function, то получаете полезную информацию. Четвертый аргумент сообщает Emacs'у, что данная автозагружаемая функция может быть вызвана в интерактивном режиме (т.е. через M-x). Это очень важно в данном случае, потому что должна быть возможность начать схемный процесс, работающий под Emacs'ом, просто набрав M-x run-scheme.

Тогда после того, как run-scheme была определена как автозагружаемая функция, что произойдет, если я наберу M-x run-scheme. Emacs смотрит на функцию run-scheme, видит, что она начинает автозагружаться, и загружает файл, проименованный автозагрузкой (в данном случае "cmuscheme"). Поскольку, побайтно откомпилированный файл cmuscheme.elc существует, то Emacs будет его загружать. Данный файл ОБЯЗАН определить функцию run-scheme, иначе будет автозагрузочная ошибка. К счастью, он в самом деле определяет run-scheme, поэтому все идет гладко, и я получаю предпочитаемый схемный интерфейс. ((( 10: между прочим, cmuscheme как раз тот интерфейс, о котором я говорил ранее в разделе по работе в схемном режиме, поэтому, если вы хотите пользоваться тем, о чем там шла речь, то вам необходимо убедиться, что вы запустили cmuscheme)))

Автозагрузка - это своего рода обещание Emacs'у того, что когда наступит время, то он сможет найти означенную функцию в файле, который вы ему укажите. В свою очередь, вы получаете некоторый контроль, над тем, что загружается. Также автозагрузки помогают уменьшить объем памяти, занимаемой Emacs'ом за счет того, что определенные настройки не будут загружены до тех пор, пока они не потребуются. Многие команды в дейтвительности не определяются как функции, когда запускается Emacs. Более того, они просто автозагружаются из определенного файла. Если вы никогда не обращаетесь к команде, она никогда не загружается. Эта экономия пространства жизненно необходима для функционирования Emacs'а: если бы он загружал каждый доступный файл в Lisp библиотеке, то Emacs'у потребовалось двадцать минут для того, чтобы просто запуститься, а после этого он занял бы всю доступную память на вашей машине. Не беспокойтесь, вам не придется устанавливать все эти автозагрузки в вашем .emacs; о них позаботились при создании Emacs'а.

## 8.14 Узнавая нечто большее.

Я не сказал вам всего, что можно узнать об Emacs'е. На самом деле, я не думаю, что сказал вам хотя бы один процент. В то время, как вы знаете достаточно для элементарного пользования, существует большое количество удобств и приемов, экномящих время, с которыми вам следовало бы познакомиться. Лучший способ это сделать - дождаться, когда вам что-либо понадобится самому, затем поискать функцию, которая это делает.

Даже нельзя выразить, насколько важно удобство системы подсказок в Emacs'е. Например, допустим, вы хотите, чтобы было возможно

поместить содержимое некоего файла в буфер, в котором загружен другой файл так что, данный буфер будет содержать оба этих файла. Если бы вы догадались, что есть команда insert-file, то были бы правы. Для того, чтобы проверить вашу гениальную догадку, наберите C-h f. В ответ на приглашение в мини-буфере введите имя функции, о которой вам нужна подсказка. Так как вы знаете, что дейсвует механизм завершений по именам функций, и вы можете догадаться, что команда, которую вы ищете, начинается с "insert", то наберите insert и нажмите [Tab]. Это покажет вам все функции, имена которых начинаются с "insert", в частности "insert-file".

Итак, вы завершаете имя функциии читаете о том, как она работает, а затем пользуетесь M-x insert-file. Если вам интересно, соответсвует ли этой функции какая-нибудь клавишная комбинация, вы набираете C-h w insert-file [Return] и выясняете. Чем больше вы узнаете о возможностях системы подсказок (help) Emacs'а, тем легче вам распросить Emacs о нем самом. Умение проделывать это вместе с духом исследования и желанием открывать новые пути может увенчаться экономией нажатия клавиш.

Для того, чтобы заказать копию руководства пользователя по Emacs'у и/или руководства по программированию на Emacs Lisp, пишите по адресу:

Free Software Foundation
675 Mass Ave
Cambridge, MA 02139
USA

Оба этих руководства распространяются в месте с Emacs'ом в виде файлов в формате, распознаваемым Info Documentation Reader (C-h i). В некоторый момент вам следует набрать C-h C-c, чтобы почитать условия распространения по Emacs'у. Это интересней, чем вам может показаться и поможет лучше прочуствовать понятие свободного программного

обеспечения. Если вы думаете, что термин "free software" означает только то, что данная программа ничего не стоит, то пожалуйста прочитайте эти условия распространения, как только у вас появится время!

## 9. Я - это я

Если бы бог считал, что нам необходим дар предвидения, то он даровал бы его нам.

### 9.1 Настройка bash.

Одна из отличительных черт философии Unix'а - это то, что системные проектировщики не пытались предусмотреть всех нужд, которые появлялись у пользователя, вместо этого они постарались сделать ее инструментарий легко подстраиваемым под конкретные нужды пользователя. Это, в основном, делается благодаря файлам конфигурации. Они часто известны, как init files (файлы инициализации), "rc-files" (файл управления выполнением), а также "dot files", поскольку имена этих файлов часто начинается с точки. Если вы помните, имена файлов, начинающиеся с ".", не высвечиваются командой ls.

Наиболее важными конфигурационными файлами являются те, которые использует shell. В LINUX'е shell'ом по умолчанию является bash, и это тот самый shell, о котором будет идти речь. Перед тем, как мы перейдем к тому, как настроить bash, нам следует знать, что нужно для bash'овских файлов.

#### 9.1.1 Запуск Shell

Существует несколько различных типов работы bash. Он может быть запущен как login shell (оболочка, назначенная в момент регистрации пользователя). Так происходит, когда вы первый раз входите в систему. Login shell является, как правило, первым shell'ом, с которым вы имеете дело.

Другой тип работы bash - это работа в качестве интерактивного shell'а. Примером интерактивного shell'а может служить любая оболочка, выдающая приглашение пользователю и ожидающая затем ввода. Следует заметить, что Login shell также является интерактивным shell'ом. Примером non-login интерактивного shell'а может служить shell, запущенный внутри xterm. Вообще говоря, любой shell, создаваемый не во-время регистрации, является non-login shell'ом.

И, наконец, существуют неинтерактивные shell'ы. Они используются для выполнения командного файла (shell script), весьма похожего на .bat-файлы в MS DOS. Такие скрипты функционируют как мини-программы. Несмотря на то, что они, как правило, гораздо медленнее обычных (откомпилированных) программ, писать их часто намного легче.

В зависимости от типа shell'а при его запуске используются различные файлы:

| Тип Shell'а | Действие |
|---|---|
| Интерактивный login | Читается и выполняется файл .bash_profile |
| Интерактивный | Читается и выполняется файл .bashrc |
| Неинтерактивный | Читается и выполняется shell script |

---

### 9.1.2 Файлы инициализации

Поскольку большинство пользователи хотят иметь одно и тоже окружение, то не имеет значения, какой именно тип shell'а они используют, и независимо от того, является ли этот shell login shell'ом, мы начинаем нашу конфигурацию с очень простой команды в файле .bash_profile: "source ~/.bashrc". По команде "source" shell интерпретирует ее аргумент как shell script. Для нас это означает, что всякий раз, когда запускается .bash_profile, файл .bashrc также запускается.

Теперь остается только добавить в наш .bashrc еще несколько команд. Если вы хотите, чтобы какая-либо команда исполнялась только тогда, когда вы входите в систему, то добавьте ее в ваш .bash_profile.

### 9.1.3 Синонимы (Aliasing)

Итак, что именно вы можете пожелать сделать? На мой взгляд, 90% пользователям следует добавить в свой .bashrc такую строчку:

```
alias ll="ls -l"
```

Эта команда определяет shell'овские синоним, называемый ll, который "расширяется" до обычной команды shell'а "ls -l", когда пользователь вызывает его. Таким образом, предположив, что Bash прочитал эту команду из файла .bashrc, вы можете просто набрать ll и получить тот же результат, что и при вызове команды "ls -l". Когда вы набираете ll и нажимаете [Return], bash перехватывает эту команду, для того, чтобы попробовать отыскать определение синонима с таким именем. Находя синоним, bash заменяет его на "ls -l" и передает на исполнение. В системе не существует программы с именем ll, но shell автоматически преобразует его в корректную программу.

Несколько незамысловатых синонимов приведены в таблице 9.1.3. Вы можете добавить их в ваш собственный .bashrc. Довольно дюбопытен первый синоним. Если его определить, то когда бы вы не набрали ls, то к нему автоматически присоединяется флаг -F. (Синоним не расширяется рекурсивно.) Это - обычный способ добавления опции, используемый при каждом вызове программы.

Обратите внимание на комментарии, начинающиеся с символа #, в таблице 9.1.3. Shell игнорирует следующую за # часть строки.

Вы, должно быть, заметили несколько странных особенностей. Во-первых, я опустил кавычки для некоторых синонимов, например, для pu. Кавычки не являются необходимыми, когда справа от знака равенства стоит только одно слово. Тем не менее, ставить их можно и даже желательно. Кавычки обязательно в случае, если вы намерены присвоить синоним команде с опциями и/или аргументами:

```
alias ls="ls -F"          # выдать символы в конце листинга
alias ll="ls -l"          # специальный ls
alias la="ls -a"
```

```
alias ro="rm *~; rm .*~"  # удаляет резервные копии, созданные Emacs'ом
alias rd="rmdir"          # так меньше набирать!
alias md="mkdir"
alias pu=pushd            # pushd, popd, and dirs не описываются в данном
alias po=popd             # руководстве --- вы можете узнать о них
alias ds=dirs             # с помощью bash manpage
# все, что ниже - просто сокращения
alias to="telnet cs.oberlin.edu"
alias ta="telnet altair.mcs.anl.gov"
alias tg="telnet wombat.gnu.ai.mit.edu"
alias tko="tpalk kold@cs.oberlin.edu"
alias tjo="talk jimb@cs.oberlin.edu"
alias mroe="more"         # правка правописания!
alias moer="more"
alias email="emacs -f rmail" # мой mail reader
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
                          # один из способов вызова emacs'а
```

Таблица 9.1.3. Некоторые простейшие синонимы для bash.

У последний синонима сложные кавычки:

```
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
```

Как вы, должно быть, догадались, я хотел обойти двойные кавычки в самих опциях, поэтому мне пришлось заключить их в кавычки с помощью символа \, чтобы bash не посчитал, что они означают конец синонима.

В конце я присвоил синонимы двум часто возникающим опечаткам: "mroe" и "moer" вместо команды more, которая имеется в виду. Синонимы не оказывают влияния на аргументы, с которыми запускается программа. Следующее хорошо работает:

```
/home/larry# mroe hurd.txt
```

На самом деле, умение как определить собственные синонимы является, по меньшей мере, половина всей настройки shell'а, которую вам придется делать. Поэкспериментируйте немного, посмотрите какие длинные команды вам приходится часто набирать, и введите для них синонимы. Вы убедитесь, что это делает работу в bash гораздо приятней.

### 9.1.4 Переменные окружения

Другая важная вещь, которую нужно сделать в .bashrc - это установить переменные окружения. А что такое переменные окружения? Давайте подойдем к этому с другой стороны: предположим, что вы читаете документацию по программе fruggle и наталкиваетесь на следующее предложение:

> Fruggle обычно ищет свой конфигурационный файл .frugglerc в личном каталоге пользователя. Однако, если переменная окружения FRUGGLEPATH установлена в другое имя файла, то он будет искать там.

Каждая программа действует в окружении, которое определяется shell'ом, который вызывает программу. (Теперь вы видите, насколько важны shell'ы. Представьте, что вам пришлось бы каждый раз при вызове программы вручную устанавливать все окружение!)

Можно сказать, что окружение существует "внутри" оболочки. У программистов есть специальная процедура для обращения к окружению, и программа fruggle этой процедурой пользуется. Она проверяет значение переменной окружения FRUGGLEPATH. Если эта переменная оказывается неопределенной, то она просто использует файл .frugglerc из ванего личного каталога. А если она определена, fruggle воспользуется значением данной переменной (которое должно быть именем файла, используемого fruggle) вместо .frugglerc.

Здесь показано, как вы можете изменить ваше окружение в bash:

```
/home/larry# export PGPPATH=/home/larry/secrets/pgp
```

Вы можете считать, что команда export означает "пожалуйста, экспортируйте эту переменную в окружение, где я буду вызывать

программы так, чтобы ее значение было для них видимым". В дальнейшем вы увидете, что есть причины называть ее export.

Мало известная программа шифрования Фила Циммермана использует специальную переменную pgp. По умолчанию pgp использует вашу личную директорию в качестве места для поиска определенных файлов, которые ей необходимы (содержащие ключи шифрования), а также, как места для хранения временных файлов, которая она создает во время работы. Присваивая переменной PGPPATH данное значение, я сказал ей, вместо этого пользоваться директорией /home/larry/secrets/pgp. Мне пришлось читать руководство по pgp, чтобы выяснить точное имя этой переменной и что она делает, но это все-таки достаточно общепринято, писать имя программы в заглавных буквах, предшествующих слову "PATH".

Также полезно иметь возможность обратиться к окружению:

```
/home/larry# echo $PGPPATH
/home/larry/.pgp
/home/larry#
```

Обратите внимание на "$"; вы ставите его перед переменной окружения для того, чтобы извлечь значение переменной. Если бы вы напечатали его без $, то
echo просто бы переписало свои аргумент(ы):

```
/home/larry# echo PGPPATH
PGPPATH
/home/larry#
```

Символ $ служит для вычисления значений переменных окружения, но он только в контексте shell'а, т.е. кокда его интерпретирует shell. А когда shell его интерпретирует? Ну, скажем, когда вы набираете команды в приглашении, или, когда bash читает команды из файла типа .bashrc, тогда можно говорить, что он интерпретирует команды.

| Имя переменной | Что содержит | Пример |
|---|---|---|
| HOME | ваш личный каталог | /home/larry |
| TERM | тип вашего терминала | xterm,vt100, or console |
| SHELL | путь к вашему shell'у | /bin/bash |

| | | |
|---|---|---|
| USER | ваше имя входа в систему | larry |
| PATH | список для поиска программ | /bin:/usr/bin:/usr/local /bin:/usr/bin/X11 |

Таблица 9.1.4: Некоторые важные переменные окружения.

Существует другая команда, которая очень полезна при обращении к окружению: env. env просто печатает все переменные окружения. Возможно, особенно когда вы используете X, что список этих переменных вылезет за пределы экрана. Если это случится, просто составьте конвейер команд env и more: env | more.

Некоторые из переменных окружения могут быть очень полезны, так что я коснусь их здесь. Посмотрите на таблицу 9.1.4. Эти четыре переменных определяются автоматически, когда вы входите в систему: вы не устанавливаете их значения в ваших файлах .bashrc или .bash_login.

Давайте более внимательно рассмотрим переменную TERM. Чтобы понять ее смысл, давайте вернемся назад к истории Unix'а: Данная операционная система нуждается в определенных данных о вашей консоли для того, чтобы предоставлять элементарные функции, такие как отображение символа на экране, перемещение курсора на следующую строку и т.п. Вначале производители компьютеров добавляли новые характеристики терминалам так, что они проявлялись постоянно: это и первое инверсное изображение, затем установка европейских символов и, в конце концов, даже примитивные графические функции (помните, было время, когда не было еще мышей и оконных систем). Однако, все эти новые функции породили проблему программистам: как они могут узнать, чем оснащен данный терминал, а чем нет? И как они могут добавить новые возможности так, чтобы не сделать старые терминалы ни к чему непригодными?

В системе Unix ответом на все эти вопросы является файл /etc/termcap. /etc/termcap - это список всех терминалов, о которых знает ваша система, и как они управляют курсором. Если системный администратор завел новый терминал, все, что нужно будет сделать, это добавить точку входа для этого терминала в файл /etc/termcap вместо перестройки всего Unix'а. Иногда это даже проще. Кстати, терминал Digital Equipment Corporation (DEC) vt100 стал неявным стандартом, и многие новые терминалы были сконструированы таким образом, что они могут эмулировать его, т.е. вести себя так, как-будто бы они и есть vt100.

В Linux'е значением переменной TERM иногда является console (консоль), которая является vt100-подобным терминалом с некоторыми дополнительными возможностями.

Другая переменная, PATH, также важна для функционирования shell'а. Вот пример:

```
/home/larry# env | grep ^PATH
PATH=/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:
                                                    /usr/TeX/bin
/home/larry#
```

Ваша переменная PATH - это список разделенных между собой каталогов, в которых shell'у следует искать программу, когда вы набираете ее имя для запуска. Когда вы набираете ls и нажимаете [Return], то, согласно примеру, сначала Bash будет искать ее в

каталоге /home/larry/bin, т.е. в директории, которую я создал для хранения программ, которые я написал. Однако, я ведь не писал программы ls (на самом деле, она могла быть написана еще до моего рождения). Не обнаружив ее там, Bash продолжает поиск в директории, следующей по списку, /bin, а там она уже есть! Файл /bin/ls действительно существует и является исполняемым, таким образом, Bash прекратит поиск программы ls и запустит ее. Возможно, что существовала другая программа под именем ls, расположенная в каталоге /usr/bin, но bash никогда бы не запустил ее, пока я не попросил бы этого, уточнив ее полное имя:

```
/home/larry# usr/bin/ls
```

Переменная PATH нужна для того, чтобы у нас не было необходимости набирать полные имена для всех команд. Когда вы набрали команду, Bash ищет ее в директориях, проименованных в PATH по порядку и, если нашел, запускает ее. Если нет, то получается грубая ошибка:

```
/home/larry# clubly
clubly: command not found (команда не найдена)
```

Заметьте, что моя переменная PATH не содержит текущую директорию, ".". Если бы она содержала ее, то она могла бы выглядеть следующим образом:

```
/home/larry# echo $PATH
.:/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
/home/larry#
```

Это является спорным моментом в Unix'овских кругах (теперь и вы их участник, хотите вы этого или нет). Проблема заключается в том, что включение в вашу переменную PATH текущего каталога может быть дырой в системе защиты. Предположим, что вы переходите (командой cd) в директорию, где кто-то оставил "троянского коня", программу под именем ls, и вы запускаете ls, что было бы естественно при входе в новую директорию. Поскольку текущая директория, ".", идет первой а вашей переменной PATH, то shell нашел бы именно эту версию программы ls и запустил бы ее. Какие только пакости нельзя поместить в эту программу, а вы буквально только что сами взяли и запустили ее. Типу, который написал ее, даже не нужно предоставлять какие-либо привилегии для этого, просто нужно выдать разрешение на директорию, где расположена "ложная" ls. Это даже может быть его домашняя директория, если он знает, что вы будете работать где-то около.

Вашей собственной системе очень не повезет, если с ней работают люди, оставляющие такие ловушки для других. Возможно, что все пользователи - ваши друзья или коллеги. Однако, в больших многопользовательских системах (например, в университетских) легко может оказаться несколько недружелюбных программистов, которых вы

никогда и не встречали. Так это или нет, хотите ли вы иметь такую возможность, используя "." в вашей PATH, зависит от ситуации; я не собираюсь быть догматичным, а просто хочу предупредить вас о возможном риске. (Помните, что вы всегда можете запускать программы из текущей директории, задав их более точно, например, "./foo".) Многопользовательские системы вообще представляют собой сообщества, где люди могут делать друг другу различные "подарки" всевозможными способами.

То, как я на самом деле установил свою переменную PATH, включает

в себя многое из того, что вы уже узнали о переменных окружения. Вот, что действительно содержится в моем файле .bashrc:

```
export PATH=${PATH}:.:${HOME}/bin/:bin:/usr/bin:/usr/local/bin:
                                    /usr/bin/X11:/usr/TeX/bin
```

Здесь я использую тот факт, что переменная HOME установлена ранее, чем Bash начнет читать мой .bashrc, и я могу использовать ее значение при установке моей PATH. Фигурные скобки представляют собой дальнейший уровень "закавычивания"; они выделяют то, что именно должен вычислить "$", так что shell не придет в замешательство от текста, который немедленно следует за ним ("/bin" в данном случае). Вот другой пример того, как это действует:

```
/home/larry# echo ${HOME}foo
/home/larryfoo
/home/larry#
```

Без этих кавычек у меня ничего бы не получилось, поскольку не существует переменной окружения с именем HOMEfoo.

```
/home/larry# echo $HOMEfoo

/home/larry#
```

Позвольте мне прояснить еще одно обстоятельство, связанное с переменной PATH: значение "$PATH". С помощью этого в мою новую PATH добавляется значение переменной PATH, которое было установлено ранее..

Но где могла быть установлена та, старая, переменная? Файл /etc/profile служит, как бы, в качестве глобального .bash_profile, который является общим для всех пользователей. Имея такой централизованный файл, системному администратору становится гораздо проще добавить новую директорию в переменную PATH каждого пользователя, или что-нибудь подобное, без необходимости проделывать это индивидуально каждому. Если включаете старое значение в вашу новую переменную, вы никогда не потеряете те директории, которые система уже установила для вас.

Вы также можете управлять видом вашего приглашения. Это делается с помощью установки переменной окружения PS1. Лично я хочу, чтобы приглашение показывало текущую рабочую директорию - вот, как я сделал это в своем .bashrc:

```
export PS1='$PWD# '
```

Как вы заметили, на самом деле здесь используются две переменные. Одна, та, которая устанавливается, это PS1, а другая, значение которой используется, это PWD, которую можно расшифровать как "Print Working Directory" или как "Path to Working Directory". Но взятие значения PWD происходит внутри одинарных кавычек. Они служат для того, чтобы вычислить выражение внутри них, которое само вычисляет значение переменной PWD. Если бы мы просто выполнили команду export PS1=$PWD, то приглашение все время высвечивало бы тот каталог, в котором произошла установка PS1. Это все, конечно, немного запутано и не так уж важно. Просто имейте в виду, что вам необходимо поставить '', если вы хотите, чтобы текущая директория постоянно отображалась в вашем приглашении.

Возможно, что вы предпочтете команду export PS1='$PWD>' или даже имя вашей системы: export PS1=`hostname`'>'. Позвольте мне ниже проанализировать последний пример.

В этом последнем примере используется новый тип закавычивания, с помощью обратных одинарных кавычек ``. Это не служит для предохранения чего-либо - на самом деле, вы увидите, что слово "hostname" так не появится в приглашении, когда вы запустите это. В действительности

происходит то, что команда внутри этих обратных кавычек будет вычислена, и результат ее работы будет выведен в то место, где должна была бы стоять закавыченное имя команды.

Попробуйте выполнить команды echo `ls` или wc `ls`. По мере того, как вы будете становиться более опытным пользователем shell'а, эта техника будет становиться все более и более мощной.

Есть еще очень много вещей, подлежащих конфигурации в вашем .bashrc, и здесь не хватит места, чтобы рассказать о них. Вы можете почитать bash man page для получения дополнительной информации, или просто пораспрашивать более опытных пользователей. Вот вам для изучения законченный вариант .bashrc; он полностью стандартен, хотя путей для поиска, установленных в переменной path немного больше обычного.

```
# всякие мелочи:
ulimit -c unlimited
export history_control=ignoredups
export PS1='$PWD>'
umask 022

# пути, нужные приложениям:
export MANPATH=/usr/local/man:/usr/man
export INFOPATH=/usr/local/info
export PGPPATH=${HOME}/.pgp

# установка основной переменной PATH:
homepath=${HOME}:~/bin
stdpath=/bin:/usr/bin:/usr/local/bin:/usr/ucb/:/etc:/usr/etc:/usr/games
pubpath=/usr/public/bin:/usr/gnusoft/bin:/usr/local/contribs/bin
softpath=/usr/bin/X11:/usr/local/bin/X11:/usr/TeX/bin
export PATH=.:${homepath}:${stdpath}:${pubpath}:${softpath}
# фигурные скобки не обязательны, так как двоеточие является
# разделителем; тем не меннее, фигурные скобки - это хороший тон,
# и они ничем не могут повредить.

# синонимы


alias ls="ls -CF"
alias fg1="fg %1"
alias fg2="fg %2"
alias tba="talk sussman@tern.mcs.anl.gov"
alias tko="talk kold@cs.oberlin.edu"
alias tji="talk jimb@totoro.bio.indiana.edu"
alias mroe="more"
alias moer="more"
alias email="emacs -f vm"
alias pu=pushd
alias po=popd
alias b="~/.b"
alias ds=dirs
alias ro="rm *~; rm .*~"
alias rd="rmdir"
alias ll="ls -l"
```

```
alias la="ls -a"
alias rr="rm -r"
alias md="mkdir"
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""

function gco
{
  gcc -o $1 $1.c -g
}
```

### 9.2 Файлы инициализации системы X Window System

Большинство людей предпочитают работать внутри графического окружения, и для Unix'овских машин это обычно означает использование системы X. Если вы привыкли к Macintosh или Microsoft Windows, то X Window System может вам чем-то напомнить их, особенно способом настройки.

В Macintosh или Microsoft Windows вы настраиваете окружение как бы изнутри самого окружения: если вы хотите изменить фон, например, вы делаете это, выбирая с помощью мыши новый цвет в накоторой специальной графической программе. В системе X ее настройки "по умолчанию"

хранятся в текстовых файлах, которые вы редактируете непосредственно - иначе говоря, вам следует набрать название цвета в файле для того, чтобы установить цвет фона.

Никто не возражает, что этот метод не является таким же быстрым, как в некоторых коммерческих оконных системах. Мне кажется, что тенденция системы оставаться текстовой, даже в графическом окружении, связана с тем, что X создавалась группой программистов, которая не пыталась написать программное обеспечение для чайников. Эта тенденция может измениться в дальнейших версиях X (по крайней мере, я так надеюсь), но сейчас вам придется научиться иметь дело с большим числом текстовых файлов. Это, по крайней мере, дает вам гибкий и прочный контроль над вашей конфигурацией.

Здесь приведены наиболее важные файлы для конфигурирования X.

.xinitrc Скрипт, запускаемый X'ом в начале работы.
.twmrc Читается X window manager, twm.
.fvwmrc Читается X window manager, fvwm.

Все эти файлы, если они существуют, должны лежать в вашем личном каталоге.

.xinitrc - это обычный скрипт shell'а, который запускается при вызове X. Он может делать то же самое, что и любой другой скрипт shell'а, но, конечно, наиболее разумно использовать его при запуске различных X-программ и установке параметров window system. Последняя команда, .xinitrc, это обычно название запускаемого window manager, например, /usr/bin/X11/twm.

Какого рода вещи хотели бы вы поместить файл .xinitrc? Наверное несколько обращений к программе xsetroot, необходимых для того, чтобы ваше корневое (фоновое) окно и курсор мышь выглядели бы так, как вам хочется. Обращения к xmodmap, которые сообщают серверу, как интерпретировать сигналы от клавиатуры. Прочие программы, которые вы хотели бы запускать каждый раз при запуске X (например, xclock).

"Сервер" означает основной X процесс на вашей машине, тот с которым все остальные X программы взаимодействуют, чтобы пользоваться дисплеем. Эти остальные программы называются клиентами, а все целиком называется системой "клиент-сервер".

Здесь приведена часть моего .xinitrc; ваш почти наверняка будет выглядеть иначе, поэтому, это - все лишь пример:

```
# /bin/sh
# Первая строка сообщает операционной системе, каким shell'ом пользоваться
# при интерпретировании данного скрипта. Сам скрипт должен быть помечен
# как выполняемый; вы можете это сделать с помощью "chmod +x ~/.xinitrc".
# xmodmap - это программа, сообщающая серверу, как интерпретировать
# сигналы с вашей клавиатуры. Она определенно стоит того, чтобы
# ее изучить. Вы можете запустить "man xmodmap", "xmodmap -help"
# "xmodmap - grammer" и т.д. Я не гарантирую, что выражение внизу
# будет значить что-либо для вашей системе (я даже не гарантирую,
# что будет что-либо значить для моей)

xmodmap -e 'clear Lock'
xmodmap -e 'keycode 176 = Control_R'
xmodmap -e 'add control = Control_R'
xmodmap -e 'clear Mod2'
xmodmap -e 'add Mod1 = Alt_L Alt_R'

# xset - программа для установления некоторых прочих параметров для
# X сервера:
xset m 3 2 &       # параметры мыши
xset s 600 5 &     # параметры заставки
xset s noblank &   # ditto
xset fp+ /home/larry/x/fonts # for cxterm
# для того, чтобы узнать больше, воспользуйтесь "xset -help".

# Приказывает X-серверу скомбинировать fish.cursor с fish.mask и
# использовать получившийся образец в качестве курсора мыши.
xsetroot -cursor /home/lab/larry/x/fish.cursor /home/lab/larry/x/fish.mask
&




# Приятная палитра фона и цвет:
xsetroot -bitmap /home/lab/larry/x/pyramid.xbm -bg tan

# todo: xdrb here? What about .Xdefaults file?

# Вам следует воспользоваться "man xsetroot" или "xsetroot help" для
# получения большей информации о программе, представленной выше.

# Программа-клиент Jim'а Blandy появления часов с цветным циферблатом:
/usr/local/bin/circles &

# Возможно вы захотите появления часов на вашем экране все время?
/usr/bin/X11/xclock -digital &

# Позволяет программам-X-клиентам, работающим в
# occs.cs.oberlin.edu воспроизводить себя на дисплеях;
# то же самое для juju.mcs.anl.gov:
xhost occs.cs.obrlin.edu
xhost juju.mcs.anl.gov

# Вы можете приказать X-серверу, дать возможность клиентам
```

```
# на любом другом узле (узлом является удаленная машина)
# отображать свою работу здесь. Эта является дырой в системе
# безопасности -- эти клиенты могут быть запущены кем-то другим,
# и следить за нажатием клавиш, когда вы # печатаете ваш пароль
# или что-нибудь в этом роде. Однако, если вы все-таки # хотите
# это сделать, вы можете поставить "+" вместо всевозможных имен
# узлов, а не специальное имя узла, например:
# xhost +

 # И, наконец, запускаем window manager:
/usr/bin/X11/twm
# Некоторые люди предпочитают другие window manager'ы. Я
# пользуюсь twm, но вместе с Linux часто распространяет также
# fvwm:
# /usr/bin/X11/fvwm
```

Заметим, что некоторые команды запускаются в фоновом режиме (т.е. за ними следует "&"), а некоторые нет. Отличие состоит в том, что некоторые программы запускаются вместе с X'ом и продолжают работу до вашего выхода - они и помещаются в фоновый режим. Остальные однажды выполняются и немедленно заканчивают работу, одна из таких xsetroot; она только устанавливает корневое окно, или курсор, или что-нибудь еще, а затем завершаются.

Как только запускается window manager, он начинает обрабатывать свой собственный инициализационный файл, в котором содержится информация о том, как установлено ваше меню, на какие позиции вынесены окна, контроль над иконками и прочие не менее важные вещи. Если вы пользуетесь twm, то инициализационным файлом является файл .twmrc вашего личного каталога. Если вы, пользуетесь fvwm, то это, соответственно, .fvwmrc и т.д. Я работаю только с этими двумя, поскольку это window manager'ы, самые подходящие для Linux'а.

### 9.2.1 Конфирация twm

.twmrc - это не скрипт shell'а; он написан на языке, специально сделанным для twm!^4, хотите, верьте, хотите - нет. (Самое неприятное в файлах инициализации, это то, что, как правило, у каждого есть свой идиосинкратичный командный язык. Это означает, что пользователи начинают быстро обучаться командным языкам. Я полагаю, что это было бы прекрасно, если бы первые Unix'овские программисты бы согласились на некоторый стандартный формат файлов инициализации, так чтобы нам не пришлось каждый раз изучать новый синтаксис, но, если быть честным, трудно предсказать, какого рода информация понадобится программам.) Больше всего люди любят забавляться в своем .twmrc с оконным стилем (т.е., цвета и т.п.) и с изготовлением различных меню, поэтому, здесь приведен пример .twmrc, который делает это.

```
# Установка цветов для различных частей окон. Это очень помогает
# "чуствовать" ваше окружение

Color
```

```
{
    BorderColor "OrangeRed"
    BorderTileForeground "Black"
    BorderTileBackground "Black"
    TitleForeground "black"
    TitleBackground "gold"
    MenuForeground "black"
    MenuBackground "LightGrey"
    MenuTitleForeground "LightGrey"
    MenuTitleBackground "LightSlateGrey"
    MenuShadowColor "black"
    IconForeground "DimGray"
    IconBackground "Gold"
    IconBorderColor "OrangeRed"
    IconManagerForeground "black"
    IconManagerBackground "honeydew"
}


# Я надеюсь, что у вас не монохромная система, но если уж так ...
Monochrome
{
    BorderColor "black"
    BorderTileForeground "black"
    BorderTileBackground "white"
    TitleForeground "black"
    TitleBackground "white"
}

# Я создал baifang.bmp с программой "bitmap". Здесь я приказываю twm
# использовать ее по умолчанию в заголовках окон
Pixmaps
{
    TitleHighlight "/home/larry/x/beifang.bmp"


}

# Не беспокойтесь по поводу этой чепухи, она только для
# продвинутых пользователей.
BorderWidth     2
TitleFont       "-adobe-new century schoolbook-bold-r-normal--14
                                    -140-75-75-p-87-iso8859-1"
MenuFont        "6x13"
IconFont        "lucidasans-italic-14"
ResizeFont      "fixed"
Zoom 50
RandomPlacement


# Эти программы не получают заголовок окна по умолчанию:
NoTitle
{
  "stamp"
  "xload"
  "xclock"
  "xlogo"
  "xbiff"
  "xeyes"
  "oclock"
  "xoid"
}
```

```
# AutoRaise означает, что окно выносится вперед, где бы его не
# открыл мышиный указатель. Меня это раздражает, поэтому я это
# выключил.  Как вы можете заметить, я унаследовал свой .twmrc
# от людей, которым тоже не нравился AutoRaise.
AutoRaise
{
  "nothing"     # I don't like auto-raise  # Me either  # nor I
}

# Здесь определяются функции кнопки мыши. Обратите внимание на
# образец: если кнопка мыши нажата в корневом окне, если не нажата


# клавиша модификации, то всегда выводится меню. Прочее расположения
# курсора обычно используются для различных оконных манипуляций,
# клавиши модификации используются вместе с кнопка мимыши, чтобы
# добиться еще сложных манипуляций.
#
# Вы не обязаны следовать этому образцу в вашем .twmrc.

# Кнопка мыши = Клавиша : Контекст : Функция
# ------------------------------------------
Button1 =      : root    : f.menu "main"
Button1 =      : title   : f.raise
Button1 =      : frame   : f.raise
Button1 =      : icon    : f.iconify
Button1 = m    : window  : f.iconify

Button2 =      : root    : f.menu "stuff"
Button2 =      : icon    : f.move
Button2 = m    : window  : f.move
Button2 =      : title   : f.move
Button2 =      : frame   : f.move
Button2 = s    : frame   : f.zoom
Button2 = s    : window  : f.zoom

Button3 =      : root    : f.menu "x"
Button3 =      : title   : f.lower
Button3 =      : frame   : f.lower
Button3 =      : icon    : f.raiselower

# Вы можете написать ваши собственные функции; эта используется
# в меню "windowops" у конца этого файла:
Function "raise-n-focus"
{
    f.raise
    f.focus
}

# Итак, внизу приведены те меню, на которые делались ссылки
# в разделе о кнопках мыши. Заметьте, что многие из пунктов


# в этих меню называются подменю. Вы можете создать столько
# уровней меню, сколько захотите, но помните, что рекурсивные
# меню не действуют. Я это проверил.
menu "main"
{
"Vanilla"       f.title
"Emacs"         f.menu "emacs"
"Logins"        f.menu "logins"
"Xlock"         f.menu "xlock"
"Misc"          f.menu "misc"
```

```
}

# Это позволяет мне вызвать Emacs на нескольких различных
# машинах. Чтобы узнать побольше о том, как это работает,
# посмотрите раздел о файлах .rhosts:
menu "emacs"
{
"Emacs"      f.title
"here"       !"/usr/bin/emacs &"
""           f.nop
"phylo"      !"rsh phylo \"emacs -d floss:0\" &"
"geta"       !"rsh geta \"emacs -d floss:0\" &"
"darwin"     !"rsh darwin \"emacs -d floss:0\" &"
"ninja"      !"rsh ninja \"emacs -d floss:0\" &"
"indy"       !"rsh indy \"emacs -d floss:0\" &"
"oberlin"    !"rsh cs.oberlin.edu \"emacs -d
                        floss.life.uiuc.edu:0\" &"
"gnu"        !"rsh gate-1.gnu.ai.mit.edu \"emacs -d
                        floss.life.uiuc.edu:0\" &"
}

# Это позволяет мне вызвать xterm'ы на нескольких различных
# машинах. Чтобы узнать побольше о том, как это работает,
# посмотрите раздел о файлах .rhosts:
menu "logins"
{
"Logins"     f.title
"here"       !"/usr/bin/X11/xterm -ls -T `hostname` -n `hostname` &"


"phylo"      !"rsh phylo \"xterm -ls -display floss:0 -T phylo\" &"
"geta"       !"rsh geta \"xterm -ls -display floss:0 -T geta\" &"
"darwin"     !"rsh darwin \"xterm -ls -display floss:0 -T darwin\" &"
"ninja"      !"rsh ninja \"xterm -ls -display floss:0 -T ninja\" &"
"indy"       !"rsh indy \"xterm -ls -display floss:0 -T indy\" &"
}

# Xlock'овский скринсейвер, вызываемый с различными опциями,
# (каждыя дает свою приятную картинку):
menu "xlock"
{
"Hop"   !"xlock -mode hop &"
"Qix"   !"xlock -mode qix &"
"Flame" !"xlock -mode flame &"
"Worm" !"xlock -mode worm &"
"Swarm" !"xlock -mode swarm &"
"Hop NL"   !"xlock -mode hop -nolock &"
"Qix NL"   !"xlock -mode qix -nolock &"
"Flame NL" !"xlock -mode flame -nolock &"
"Worm NL" !"xlock -mode worm -nolock &"
"Swarm NL" !"xlock -mode swarm -nolock &"
}

# Разнообразные программы, которые я запускаю от случая
# к случаю:
menu "misc"
{
"Xload"         !"/usr/bin/X11/xload &"
"XV"            !"/usr/bin/X11/xv &"
"Bitmap"        !"/usr/bin/X11/bitmap &"
"Tetris"        !"/usr/bin/X11/xtetris &"
"Hextris"       !"/usr/bin/X11/xhextris &"
"XRoach"        !"/usr/bin/X11/xroach &"
```

```
"Analog Clock"  !"/usr/bin/X11/xclock -analog &"
"Digital Clock" !"/usr/bin/X11/xclock -digital &"
}



# Вот то, что я сопоставил средней кнопке мыши:
menu "stuff"
{
"Chores"       f.title
"Sync"         !"/bin/sync"
"Who"          !"who | xmessage -file - -columns 80 -lines 24 &"
"Xhost +"      !"/usr/bin/X11/xhost + &"
"Rootclear"    !"/home/larry/bin/rootclear &"
}

# X функции, которые иногда бывают полезны:
menu "x"
{
"X Stuff"              f.title
"Xhost +"              !"xhost + &"
"Refresh"              f.refresh
"Source .twmrc"        f.twmrc
"(De)Iconify"          f.iconify
"Move Window"          f.move
"Resize Window"        f.resize
"Destroy Window"       f.destroy
"Window Ops"           f.menu "windowops"
""                     f.nop
"Kill twm"             f.quit
}

# Вот подменю, упомянутое выше:
menu "windowops"
{
"Window Ops"           f.title
"Show Icon Mgr"        f.showiconmgr
"Hide Icon Mgr"        f.hideiconmgr
"Refresh"              f.refresh
"Refresh Window"       f.winrefresh
"twm version"          f.version
"Focus on Root"        f.unfocus
"Source .twmrc"        f.twmrc
"Cut File"             f.cutfile


"(De)Iconify"          f.iconify
"DeIconify"            f.deiconify
"Move Window"          f.move
"ForceMove Window"     f.forcemove
"Resize Window"        f.resize
"Raise Window"         f.raise
"Lower Window"         f.lower
"Raise or Lower"       f.raiselower
"Focus on Window"      f.focus
"Raise-n-Focus"        f.function "raise-n-focus"
"Destroy Window"       f.destroy
"Kill twm"             f.quit
}
```

Поверьте мне, это не самый сложный .twmrc, который я когда-либо видел. Достаточно вероятно, что некоторые полезные файлы-примеры

.twmrc работают в вашей системе X. Загляните в директорию /usr/lib/X11/twm/ или /usr/X11/lib/X11/twm и просмотрите, что там есть.

Ошибка, которая выявляется с помощью файла .twmrc, это непоставленный символ "&" после команды о меню. Если вы заметили, что X зависает, когда вы запускаете определенные команды, то, возможно, это происходит именно по этой причине. Выйдите из X с помощью [Ctrl][Alt][Backspace], отредактируйте ваш .twmrc и попробуйте снова.

### 9.2.2 Конфигурация fvwm

Если вы пользуетесь fvwm, то директория /usr/lib/X11/fvwm/ (или /usr/X11/lib/X11/fvwm/) также содержат примеры конфигурационных файлов.

## 9.3 Другие файлы инициализации.

Вот некоторые из них :

.emacs обрабатывается текстовым редактором Emacs при запуске
.netrc дает входные имена и пароли по умолчанию для FTP.
.rhosts делает возможной работу на удаленной машине
.forward для автоматического перенаправления данных в mail

### 9.3.1 Файл инициализации Emacs'а

Если вы используете Emacs в качестве своего основного текстового редактора, то файл .emacs является достаточно важным. Мы с ним имели дело в главе 8.

### 9.3.2 Умолчания в FTP

Файл .netrc позволяет определить некоторые умолчания в FTP до запуска FTP. Здесь приведен небольшой пример .netrc:

```
machine floss.life.uiuc.edu login larry password fishSticks
machine darwin.life.uiuc.edu login larry password fishSticks
machine geta.life.uiuc.edu login larry password fishSticks
machine phylo.life.uiuc.edu login larry password fishSticks
machine ninja.life.uiuc.edu login larry password fishSticks
machine indy.life.uiuc.edu login larry password fishSticks

machine clone.mcs.anl.gov login fogel password doorm@
machine osprey.mcs.anl.gov login fogel password doorm@
machine tern.mcs.anl.gov login fogel password doorm@
machine altair.mcs.anl.gov login fogel password doorm@
machine dalek.mcs.anl.gov login fogel password doorm@
machine juju.mcs.anl.gov login fogel password doorm@

machine sunsite.unc.edu login anonymous password larry@cs.oberlin.edu
```

Каждая строка вашего .netrc специфицирует имя машины, входное имя, используемое по умолчанию для данной машины и пароль. Это очень удобно, если вам, скажем, часто приходится пользоваться FTP, и вы устали от постоянного набирания в разных местах имени пользователя и пароля. Программа FTP попытается произвести автоматический login, используя найденную в вашем файле .netrc информацию, если вы соединяетесь по FTP с одной из машин, содержащихся в данном файле.

Вы можете приказать FTP игнорировать ваш .netrc и не пытаться совершить auto-login, вызывая его с опцией -n: "ftp -n".

Вы должны убедиться, что ваш файл .netrc можете прочесть только вы. Используйте программу chmod для установки прав на чтение файла. Если другие люди могут его прочесть, это означает, что они могут узнать ваш пароль в различных узлах. FTP и прочие программы, использующие файл .netrc, откажутся работать, если права на чтение неподходящие. Для получении дополнительной информации о файле .netrc, когда у вас будет возможность, наберите "man .netrc" или "man ftp".

### 9.3.3 Упрощение работы на удаленной машине

Если в вашем личном каталоге есть файл .rhosts, это позволит вам запускать программы на вашей машине с удаленного узла. То есть, вы и так могли бы войти в систему на машине cs.oberlin.edu, но с правильно конфигурированным файлом .rhost, находящим на floss.life.uius.edu, вы могли бы запустить программу на floss.life.uius.edu и получать результаты на cs.oberlin.edu без login'а.

Файл .rhost выглядет так:

```
frobnozz.cs.knowledge.edu jsmith
aphrodite.classics.hahvaahd.edu wphilps
frobbo.hoola.com trixie
```

Формат файла очень прост: имя машины, за ним следует имя пользователя. Предположим, что этот пример - это на самом деле мой файл .rhost на floss.life.uius.edu. Это означает, что я могу запускать программы на floss'е с результатом передающимся любой из перечисленных

машин так, как будто я вошел туда как соответствующий пользователь данной машины.

Обычно для запуска удаленной программы используется программа rsh. Она действует как "удаленный shell" - запускает shell на удаленной машине и выполняет определенную команду, например:

```
frobbo$ whoami
trixie
frobbo$ rsh floss.life.uiuc.edu "ls ~"
foo.txt    mbox   url.ps    snax.txt
frobbo$ rsh floss.life.uiuc.edu "more ~/snax.txt"
[snax.txt comes paging by here]
```

Пользователь trixie на floss.life.uius.edu, у которого был вариант файла .rhosts, приведенный выше, позволяет trixie из frobbo.hoola.com запускать программы как trixie с floss'а.

Нет необходимости иметь одинаковое имя пользователя на всех машинах, чтобы .rhost правильно работал. Пользуйтесь опцией "-l" для rsh, чтобы сообщить удаленной машине, каким именем пользователя вы хотели бы пользоваться для login'а. Если это имя на удаленной машине

существует и имеет файл .rhost с вашими текущимими (т.е. локальными) именами машины и пользователя, rsh будет работать.

```
frobbo$ whoami
trixie
frobbo$ rsh -l larry floss.life.uiuc.edu "ls ~"
[ Вставьте здесь листинг моей директории на floss'е ]
```

Это будет работать, у пользователя на floss.life.uiuc.edu есть файл .rhost, который позволяет пользователю trixie из frobbo.hoopla.com запускать команды на своей машине. Являются ли оба этих пользователя одним и тем же лицом не имеет значения: единственное, что важно, это имена пользователей, машин и содержимое файла .rhost у пользователя larry на машине floss. Заметьте, что файл .rhost у пользователя trixie на машине frobbo не играет роли, поскольку, важно только содержимое файла на той удаленной машине

(floss).

Есть и другие комбинации, которые могут встретиться в файле .rhost - например, вы можете опустить имя пользователя, которое следует за именем удаленной машины, чтобы позволить пользователю с той машины запускать программы также, как вы на локальной машине! Разумеется, здесь вы рискуете безопасностью: кто-то может запустить программу, которая будет удалять ваши файлы, просто пользуясь тем, что он имеет доступ к определенной машине. Если вы собираетесь делать подобные вещи, то вы должны быть уверены в том, что ваш файл .rhost доступен только вам и больше никому.

### 9.3.4 Переадресация почтовых сообщений

Вы можете также создать файл .forward, который, строго говоря, не является файлом инициализации. Если он содержит адрес электронной почты, то все сообщения, предназначенные вам, будут переадресованы на этот адрес. Это полезно, если вы имеете доступ к различным системам, но хотите читать почту только в одном месте.

Существуют и другие файлы инициализации. Их точное число варьируется от системы к системе и зависит от программного обеспечения, установленного в вашей системе. Чтобы узнать побольше, посмотрите файлы в вашем личном каталоге, имена которых начинаются с ".". Нет гарантии, что все эти файлы - файлы инициализации, но, скорей всего, большинство из них являются таковыми.

## 9.4 Просмотр некоторых примеров

В качестве интересного и содержательного примера, я могу рассказать вам, про запуск Linux-систем. Итак, если вы имеете доступ к Internet, войдите telnet'ом в floss.life.uiuc.edu под именем "guest" с паролем "explorer", и покопайтесь там. Большинство файлов с примерами, приведенных здесь, можно найти в /home/kfogel, а также в других каталогах. Вы можете копировать все, что вам удалось прочесть. Пожалуйста, будьте осторожны: floss не имеет надежной системы защиты,

и вы, конечно, можете получить к ней доступ как администратор, если хорошо постараетесь. Я предпочитаю доверять людям.

# 10. Поговорим о других вещах

Unix-ные сети очень хороши. Две Unix-компьютера могут обмениваться информацией с помощью массы различных способов. В этой главе я попытаюсь рассказать о об обширнейших возможностях, предоставляемых сетью.

В этой главе мы изучим электоронную почту, службу новостей Usenet и несколько других утилит Unix, используемых для передачи сообщений.

## 10.1 Электронная Почта

Электронная почта - это одно из самых популярных стандартных средств Unix С ней вам не надо будет искать конверт, листок бумаги, ручку, марку, и пользоваться услугами почтовой службы.

### 10.1.1 Отправление Почты

Все что вам нужно сделать, это написать mail username (имя_пользователя) и ваше сообщение.

Например, я хочу послать почту пользователю sam:

```
/home/larry# mail sam
Subject: The user documentation
Just testing out the mail system.
EOT
/home/larry#
```

mail - очень простая программа. Как и cat, она берет данные со стандартного ввода по одной строке, пока в строке не встретится символ конца текста Ctrl-d. Поэтому для того, чтобы отправить сообщение мне нужно нажать клавишу ввода и затем Ctrl-d.

mail - самый быстрый способ отправить почту, эту программу хорошо использовать в сочетании с каналами и перенаправлениями ввода/вывода. Например, если я хочу послать файл report1 пользователю "Sam", мне надо написать mail Sam < report1, можно даже послать результаты работы программы "sort report1 | mail Sam".

Однако, в mail есть и плохие стороны. mail очень плохой редактор. вы не можете изменить строку, после того, как нажали клавишу ввода! Поэтому я советую вам отправлять почту (когда не надо использовать каналы и перенаправления ввода/вывода) при помощи Emacs'а. Как это делать описано в части 8.10.

### 10.1.2 Чтение Почты

Программа mail предлагает немного неуклюжий способ чтения почты. Если вы напишите mail без параметров, то увидите следущее:

```
/home/larry# mail
No mail for larry
/home/larry#
```

Я собираюсь отослать почту самому себе, таким образом я могу потренироваться читать почту:

```
/home/larry# mail larry
Subject: Frogs!
and toads!
EOT
```

```
/home/larry# echo "snakes" | mail larry
/home/larry# mail
Mail version 5.5 6/1/90.  Type ? for help.
"/usr/spool/mail/larry": 2 messages 2 new
>N  1 larry                 Tue Aug 30 18:11  10/211   "Frogs!"
N  2 larry                 Tue Aug 30 18:12   9/191
&
```

Приглашением на ввод в командной строке программы mail является амперсанд ("&"). В командную строку можно вводить несколько простых

команд. Если вы напишите ? и затем нажмете клавишу ввода, будет выдана подсказка.

Основные команды программы mail следующие:

t message-list (список_сообщений) - показать сообщения на экране.
d message-list - удалить сообщения.
s message-list file(файл) - сохранить сообщения в файле.
r message-list - ответить на сообщения - то есть, начать составлять новое сообщение тому, кто прислал вам сообщение, находящееся в списке.
q - завершить работу и сохранить все сообщения, которые вы не удалили, в файле mbox вашего домашнего каталога.

Что такое список_сообщений? Он состоит из целых чисел, разделенных пробелами, (или даже интервалов целых чисел, как например, 2-4 (то же самое, что "2 3 4"). Вы также можете ввести имя отправителя, по команде t Sam будут напечатаны все сообщения от Sam'а. Если список сообщений не указан, предполагается, что будет показано последнее сообщение.

Есть некоторые проблемы при чтении с помощью программы mail. Во-первых, если сообщение больше длины вашего экрана, программа mail не остановится после вывода первой страницы! Вам придется сохранить это сообщение и прочитать его позже при помощи команды more. Во-вторых, в программе нет хорошего интерфейса для старой почты - если вы захотите сохранить сообщение и прочитать его позже.

В emacs также есть возможность чтения файлов, при помощи rmail, но об этом не рассказывается в этой книге. Кроме того, большинство Linux-систем имеют несколько других программ для чтения почты, таких как elm или pine.

10.2 Новостей больше, чем достаточно

10.3 Поиск людей

10.4 Использование Систем с Удаленного Терминала

Если вы используете X, давайте создадим новый xterm для других систем, с которыми мы работаем. Используйте команду " xterm -title "lionsden" -e telnet lionsden &". Эта команда создаст новое xterm окно, которое автоматически запускает telnet. (Если вы делаете это часто, вы можете создать для этого синоним или скрипт shell'а.)

10.5 Передача Файлов на Лету

11. Смешные Команды

Большинство людей, которым приходилось иметь дело с командами Unix, описанными в этой главе не согласятся с таким заголовком. "Что за черт! Только что было показано, что интерфейс Linux очень стандартен, а сейчас оказывается что есть несколько команд, каждая из которых работает по-своему. Я никогда не запомню всех тех опций, а вы говорите, что команды смешные?" Да, вы только что видели пример хакерского юмора. Посмотрите на это с другой стороны: в MS-DOS нет эквивалента этим командам. Если они нужны вам, вам придется купить их, и вы не знаете, каким будет интерфейс. Здесь эти команды очень полезны и недороги - "в нагрузку". Так что радуйтесь жизни!

Команды, которые описаны в этой главе, включают find, которая позволяет пользователю искать в дереве директорий заданную группу файлов; tar, для создания архивов, которые надо отправить куда-либо или просто сохранить; dd, которая производит низкоуровневое копирование; и sort, которая ... правильно, сортирует файлы. И последняя оговорка: эти команды никаким образом не стандартизованы, и если ядро общих опций можно найти во всех *IX системах, GNU версия, изложенная ниже, и которую вы можете найти в вашей Linux системе, имеет гораздо больше возможностей. Поэтому, если вы планируете использовать другие UNIX-подобные операционные системы, не забудьте проверить их справочное руководство на целевой машине, чтобы изучить, возможно, небольшие отличия.

## 11.1 find, Команда, Осуществляющая Поиск Файлов

### 11.1.1 Общие сведения

Среди множества команд, рассмотренных ранее есть некоторые, которые позволяют пользователю рекурсивно спускаться по дереву директорий, для того, чтобы выполнить некоторые действия. Канонический пример - это ls -R и rm -R. Аналогичным образом, find - рекурсивная программа. Когда вы думаете "Ну что, мне придется сделать то-то и то-то во всех таких типах файлов моего логического диска", вам стоит подумать об использовании find. В определенном смысле тот факт, что find находит файлы - это просто побочный эффект; его настоящяя работа - вычисление.

Основная структура команды такая:

find путь (path) [...] выражение (expression) [...]

По крайней мере, это так в GNU версии; другие версии не позволяют задавать более одного пути, хотя не так уж часто приходится делать такое. Грубое объяснение синтаксиса команды довольно просто: вы говорите, где вы хотите начать поиск (путь; в команде find GNU вы можете не указывать это, по умолчанию будет взята текущая директория.), и какой тип поиска вы хотите исполнить (выражение).

Стандартное поведение команды немного хитрое, поэтому на него стоит обратить внимание. Давайте предположим, что в вашей базовой директории есть директория garbage, в которой содержится файл foobar. Вы пишите find -name foobar (что, как вы можете догадаться, ищет все файлы с именем foobar), и вы ничего не получаете, кроме приглашения на ввод в командной строке. Проблема в том, что find по умолчанию - "неразговорчивая" команда; она просто возвратит 0 при завершении поиска (найдя или не найдя то, что искала) и ненулевое значение, если произошла ошибка. Это происходит по-другому в той версии, которую вы

можете найти в Linux, но, в любом случае, это полезно запомнить.

### 11.1.2 Выражения

Выражения могут быть разделены на четыре группы по ключевым словам: опции, тесты, действия и операторы. Каждая из них возвращает истинное или ложное значение и имеет побочный эффект.
Рассмотрим каждую группу подробнее.

опции - влияют на операцию find в целом, а не на обработку одного файла. Примером является -follow, который указывает find следовать символическим ссылкам, а не рассматривать их как самостоятельный файл. Опции всегда возвращают истину.

проверки - это действительно проверки (например, -empty проверяет, пуст ли файл) и возвращает истину или ложь.

действия - имеют также побочный эффект - имя рассматриваемого файла. Возвращает истину или ложь.

операторы - на самом деле, не возвращают значение (по соглашению они могут рассматриваться как истина) и используются для построения сложных выражений. Примером является -or, логическое "или" двух подвыражений. Обратите внимание, когда два выражения находятся рядом, подразумевается -and - логическое "и".

Обратите внимание, что find полагается на то, что shell проанализирует командную строку; это означает, что все ключевые слова должны быть отделены пробелами и не содержать управляющих символов, т.к. shell преобразует их по своему разумению. Приемлимы все известные способы избежать преобразования управляющих символов (обратная косая черта, кавычки, двойные кавычки); в примерах односимвольному ключевому слову будет обычно предшествоать обратная косая черта, потому что это самый простой способ (по крайней мере, по-моему).

### 11.1.3 Опции

Здесь приведен список опций, распознаваемых GNU версией команды find. Помните, что все они всегда возвращают истину.

-daystart измеряет время работы, начиная не с 24 часов назад, а с последней полуночи. Настоящий хакер не поймет предназначение этой опции, но обычный человек, кто программирует от восьми до пяти оценит это.

-depth обрабатывает содержимое директории перед тем, как обработать саму директорию. Сказать по правде, я не знаю как использоовать эту опцию, кроме как для эмуляции команды rm -F (конечно, вы не можете удалить директорию перед тем, пока не будут удалены все файлы директории.)

-follow различает символические ссылки. Эта опция подразумевает

опцию -noleaf; смотрите ниже.

-noleaf означает, что при определении количества поддиректорий, содержащихся в некотором директории не следует использовать счетчик ссылок. Обычно, количество поддиректориев вычисляется как счетчик ссылок минус 2. Дело в том, что на каждую директорию ссылаются собственно по имени директории (имя файла в директории, в котором находится данная), имя '.', которое определено в каждой директории для ссылки на себя, и имя '..' в каждой поддиректории, ссылающееся на родительскую директорию.

Таким образом, на каждую директорию имеется по крайней мере две ссыли и по одной ссылке из каждой поддиректории. Однако, из этого правила могут быть исключения, например в случае символических ссылок и распределенных файловых систем. (Распределенные файловые системы позволяют файлам выглядеть локальными по отношению к машине, в то время как они расположены где-то в другом месте.)

-maxdepth levels (уровни), -mindepth levels, где уровень - это неотрицательное целое число, которое говорит, соответственно, что надо искать не больше чем, или не меньше чем levels уровней директорий. Два

примера надо привести обязательно.
maxdepth 0 означает, что эту команду следует выполнять только для аргументов, указанных в командной строке, то есть без рекурсивного спуска по дереву директорий; -mindepth 1 означает, что команда выполнится для всех файлов, уровень которых ниже, чем уровень аргументов командной строки.

-version печатает текущую версию программы.

-xdev, (название не очень соответствует смыслу), заставляет find не переходить к обработке других устройств, то есть не переходить из одной файловой системы в другую. Это очень удобно, когда вам приходится искать что-то в корневой файловой системе; на многих машинах это довольно небольшой раздел, однако в противном случае find/ будет искать во всей структуре.

### 11.1.4 Тесты

Первых два теста понять очень легко:
-false всегда выдает ложь, -true всегда выдает истину.
Другие тесты, которые не нуждаются в спецификации значения - это -empty, который возвращает истину, когда файл пустой, и пара -nouser / -nogroup, которые возвращают истину в случае, если в /etc/passwd или /etc/group нет записи, соответствующей id пользователя/группы владельца файла. В многопользовательских системах достаточно часто случается так, что пользователь был удален, но файлы, которыми он владел, остались в файловых системах, и согласно закону Мерфи занимают большую часть дискового пространства.

Конечно, можно искать определенного пользователя или группу. Для этого используются -uid nn и -gid nn. К сожалению, невозможно прямо задать имя пользователя, а необходимо использовать численные id и nn.

Разрешается использовать форму +nn, что означает "значение строго больше чем nn", и -nn, что означает "значение строго меньше чем nn" Это довольно глупо в случае UID, но удобно в других тестах.

Еще одна полезная опция - это -type c, которая возвращает истину в случае если тип файла - c. Мнемоника для возможных параметров аналогична используемым в команде ls; таким образом, используется тип файла b, если файл является файлом блочного устройства; c для файлов символьных устройств; d для директорий; p для именованых каналов; l для символических ссылок, и s для socket'ов. Обычные файлы обозначаются f. Тест, близкий по семантике -xtype, он аналогичен -type, кроме случая символических ссылок. Если опция -follow не была передана, проверяется указанный файл, а не сама ссылка. Тест, абслолютно не связанный с предыдущими - это fstype. В этом случае проверяеся тип файловой системы. Распознаются типы nfs, tmp, msdos и ext2.

-inum nn и -links nn проверяют, что файл имеет номер inode, равный nn, или nn ссылок, а -size nn выдает истину, если в файле есть по крайней мере nn 512-байтовых блоков. (это не совсем точно: для неплотных файлов неразмещенные блоки тоже считаются). Так как теперь результат ls -s не всегда измеряется в 512-байтовых кусках (Linux, например, использует 1k как основную единицу), возможно добавить к nn символ b, который означает, что надо считать в байтах, или k, в килобайтах.

Биты доступа проверяются с помощью теста -perm mode. Если перед mode нет знака, тогда биты файла должны точно соответствовать mode. Знак "-" означает, что все биты доступа должны быть установлены, но не делается никакого допущения относительно остальных. - perm +mode выдает истину, если хотя бы один из битов установлен. Режим пишется в восьмеричной системе или символьно, точно также как вы использовали их в chmod.

Следущая группа проверок работает со временем последнего использования файла. Это удобно, если у пользователя заполнено все дисковое пространство, так как обычно есть много файлов, которые он не использовал целую вечность и чье назначение он позабыл. Их трудно обнаружить, и единственная возможность найти их с помощью команды find.

-atime nn - истина, если к файлу последний раз обращались nn дней назад, -ctime nn - если статус файла был последний раз изменен nn дней назад, - например, командой chmod, и -mtime nn - если файл был последний раз изменен nn дней назад. Иногда вам нужен более точный тест -newer file (файл), он выполнен, если рассматриваемый файл был изменен позже чем file. Таким образом, вам просто придется использовать touch с желаемой датой. В GNU find кроме того, есть тесты -anewer и -cnewer, которые ведут аналогично; и тесты -amin, -cmin и -mmin, которые считают время в минутах вместо дней.

Последний по порядку, но не последний по значению тест, который я использую чаще всего. -name pattern (шаблон) выдает истину, если имя файла в точности соответствует шаблону. В стандартной команде ls вы почти всегда будете использовать этот тест. Почему `почти всегда'? Потому что, конечно, вам придется запомнить, что все параметры обрабатываются shell'ом, и при этом все метасимволы "расшиваются". Таким образом, тесты подобные -name foo* не возвращают то, что вы хотите, и вам придется писать -name foo или -name "foo*". Это, наверное, одна из наиболее частых ошибок, которые делают небрежные пользователи, так что пишите это большими буквами на своем экране. Еще одна проблема в том что, как и с ls, точки в начале имен файлов не распознаются. Чтобы избежать этого, вы можете использовать тест -path pattern который не беспокоится о точках и косых чертах при сравнении

пути рассматриваемого файла с шаблоном.

### 11.1.5 Действия

Я уже говорил, что действия - это те, кто действительно что-то делают. Итак, -prune скорее что-то делает, чем не делает, а именно, спускается по дереву директорий (если -depth не передан). Она обычно работает вместе с -fstype, чтобы выбирать среди различных файловых систем, которые надо проверить.

Остальные действия могут быть разделены на две большие категории;

- Действия, которые что-то печатают. Наиболее простое из них - действие по умолчанию команды find - это -print, которое просто печатает имя файла(ов) в соответствии с другими условиями в командной

строке, и возвращает истину. Простыми вариантами -print является -fprint file, который использует file вместо стандартного вывода; -ls печатает текущий файл в том же формате, что и ls -dils; -printf format ведет себя более или менее также как функция C printf(), таким образом, вы можете задавать формат вывода; -fprintf file format делает то же самое, но записывет в файл. Эти действия тоже возвращают истину.

- Действия, которые что-то исполняют. Их синтаксис немного странный, но они широко используются, так что, пожалуйста, обратите на них внимание.

-exec command (команда) \; команда выполняется, и действие возвращает истину, если ее код возврата 0, то есть выполнилась нормально. Причина, чтобы писать '\;' - скорее логическая: find не знает, где заканчивается команда, и хитрость с помещением действия exec в конец команды, тут не проходит. Самый хороший способ сообщить конец команды - использовать символ ';', но, конечно, сама точка с запятой в командной строке будет "съедена" shell'ом и не будет передана find. Также надо запомнить, как задавать имя текущего файла в команде. Это делается при помощи строки {}. В некоторых старых версиях find требуется чтобы {} были окружен пробелами - не очень то удобно, когда вам нужен, например, путь целиком, а не только имя файла - но в find'е GNU {} может быть в любом месте строки, составляющей команду. И не должно быть опущено или заключено в кавычки, конечно, вы спросите? Забавно, мне никогда не приходилось делать это ни под tcsh, ни под bash (sh не рассматривает { и } как специальные символы, поэтому это не является проблемой). Идея в том, что shell "знает", что {} не является опцией, имеющей какой-то смысл, поэтому shell не пытается ее "расшить", к счастью для find, который получает {} нетронутой.

-ok command \; действует также как и -exec, с той разницей, что для каждого выбранного файла у пользователя запрашивается подтверждение команды; если ответ начинается с y или Y, она исполняется, в противном случае не исполняется, и действие возвращает ложь.

### 11.1.6 Операторы

Существует большое число различных операторов; здесь приведен их список, в порядке уменьшения приоритета.

\(expr\)

Задает порядок старшинства. Скобки, конечно, должны быть заключены в кавычки, так как они распознаются и для shell'ом тоже.

! expr
-not expr

Меняет истинное значение выражения, то есть, если expr истина, выдает ложь. Восклицательный не нужно выделять символом \, так как за ним следут пробел.

expr1 expr2
expr1 -a expr2
expr1 -and expr2

Все это соответствует логической операции И, которая и подразумевается в большинстве случаев. expr2 не вычисляется, если expr1 ложно.

expr1 -o expr2
expr1 -or expr2

Соответствует логической операции ИЛИ. expr2 не вычисляется, если expr1 истинно.

expr1, expr2

Это оператор списка; expr1 и expr2 вычисляются (конечно, со всеми сторонними эффектами!) и конечное значение выражения то же, что и expr2.

### 11.1.7 Примеры

У find слишком много опций. Но есть много изящно написанных примеров, которые стоит запомнить, так как они используются слишком часто. Давайте рассмотрим некоторые из них.

```
% find . -name foo\* -print
```

Отыскивает все имена файлов, которые начинаются с foo. Если строка входит как подстрока в имя, возможно, более разумно написать что-то вроде "*foo*", а не foo.

```
% find /usr/include -xtype f -exec grep foobar \
          /dev/null {} \;
```

grep выполняется рекурсивно, начиная с каталога usr/include. В данном случае нас интересует как обычный файл, так и символические ссылки, которые указывают на обычные файлы, поэтому применяется тест -xtype. Очень часто этого можно просто не задавать, особенно, если вы уверены, что искомая строка не содержится в бинарных файлах. А почему в команде содержится /dev/null? Эта нужно для того, чтобы заставить grep записывать имя файла, в котором было найдено соответствие. Команда grep применяется ровно к одному файлу, и каждый раз запускается заново, поэтому она не думает, что нужно выводить имя файла. Но теперь есть два файла, то есть текущий файл и /dev/null! По другому это, наверное, можно сделать так: перенаправить вывод команды посредством канала в xargs и дать ей возможность исполнить grep. Я только что попытался сделать это, и полностью разрушил свою файловую систему (вместе с этими записями, которые я теперь пытаюсь

восстановить вручную :-().

```
% find / -atime +1 -fstype ext2 -name core \
             -exec rm {} \;
```

Это классическая задача для программы crontab. Команда удаляет все файлы в файловой системе типа ext2, называющиеся core, и к которым не обращались в течение последних 24 часов. Возможно, что кто-то захочет использовать файл ядра для того, чтобы сделать дамп "после

смерти", но вряд ли кто помнит, что он делал более 24 часов назад.

```
% find /home -xdev -size +500k -ls > piggies
```

Полезно знать владельцев файлов, засоривших файловую систему. Обратите внимание, что мы используем -xdev; так как мы хотим осуществить поиск только в одной файловой системе, не обязательно спускаться в другие файловые системы, монтированные под /home.

### 11.1.8 Последнее слово

Помните, что find - это команда с большими временными затратами, так как ей надо обращаться к каждому inode системы для того, чтобы выполнить действие. Поэтому разумно объединять все нужные вам операции в едином вызове find, особенно в работе по 'наведению порядка' в файловой системе, которые обычно запускаются с помощью crontab. Приведем поясняющий пример: предположим мы хотим удалить файлы, оканчивающиеся на .BAK и изменить защиту всех директорий на 771 а всех файлов, оканчивающихся на .sh на 755. И может быть мы подсоединили файловую систему NFS по модему и мы не хотим проверять файлы там. Зачем писать три различные команды? Наиболее эффективный способ выполнить эту задачу - следующий:

```
% find . \( -fstype nfs -prune \) -o \
        \( -type d       -a -exec chmod 771 {} \; \) -o \
        \( -name "*.BAK" -a -exec /bin/rm {}   \; \) -o \
        \( -name "*.sh"  -a -exec chmod 755 {} \; \)
```

Это кажется неизящным (и с злоупотреблением обратных косых черт!), но при более внимательном рассмотрении оказывается, что все не так уж и запутано. Вспомните, что на самом деле выполняются вычисления истинности; а команды - это только побочный эффект. Но это означает, что команда будет выполнена, только если find вычислит exec часть выражения, а это будет только тогда, когда левая часть выражения дает истину. Таким образом, например, если рассматриваемый в данный момент файл является директорией, то вычисляется первый exec и доступ к inode изменяется на 771; иначе он пропускает это подвыражение и переходит к следующему. Возможно, это легче увидеть на практике, чем написать; но

после некоторого промежутка времени это становиться довольно естественным.

## 11.2 tar, the tape архиватор

### 11.2.1 Введение

### 11.2.2 Основные опции

### 11.2.3 Модификаторы

### 11.2.4 Примеры

## 11.3 dd, команда копирования данных

Легенда гласит, что давным-давно, когда был создан первый UNIX, его разработчикам была нужна низкоуровневая команда для копирования данных между устройствами. Они торопились, и поэтому решили одолжить синтаксис, используемый в IBM-360 машинах, а позднее разработать интерфейс, согласующийся с другими командами. Прошло время, и все так привыкли к странному способу использования dd, что было решено ничего не изменять. Я не знаю, правда это или нет, но это забавная история.

### 11.3.1 Опции

По правде говоря, dd не так уж непохожа на остальные команды Unix: это фильтр, который считывает по умолчанию со стандартного ввода и пишет на стандартный ввод. Таким образом, если вы просто напишите dd на экране, ничего не изменится, и команда будет ожидать ввода, и самое разумное написать после этого ctrl-C.

Синтаксис команды следующий:

```
dd [if=file] [of=file] [ibs=bytes] [obs=bytes]
    [bs=bytes] [cbs=bytes] [skip=blocks] [seek=blocks]
    [count=blocks] [conv={ascii,ebcdic,ibm,block,
    unblock,lcase,ucase,swab,noerror,notrunc,sync}]
```

Таким образом, все опции имеют вид option=value (опция=значение). Не разрешаются пробелы ни перед, ни после знака равенства. Также важно запомнить, что за всеми численными значениями (байтами и блоками выше) может следовать множитель. Возможный выбор b для блока, множитель 512, k для килобайт (1024), w для слов (2), и xm, множитель m.

Значение опций if объясняется ниже.

if=filein и of=fileout сообщает dd о том, что надо считывать, соответственно, с filein и писать в fileout. В последнем случае файл вывода обрезается до значения, выданного seek, или, если ключевое слово отсутствует, до 0 (то есть удаляется), перед исполнением операции. Однако, бросьте взгляд ниже, на опцию notrunc.

ibs=nn и obs=nn задает, сколько байтов нужно считывать или записывать за раз. Я думаю, что по умолчанию это 1 блок, то есть 512 байт, но я не уверен в этом абсолютно: но можно быть уверенным, так происходит с простыми файлами. Эти параметры очень важны при использовании специальных устройств в качестве ввода или вывода; например, при считывания с сети надо установить ibs в 10k, а у 3.5 дюймовых дискет естественный размер блока 18k. Неправильная установка этих значений может привести не только к длительному исполнению команды, но и к ошибкам таймаута, поэтому будьте внимательны.

bs=nn и считывает и записывает nn байт за раз. Переопределяет ключевые слова ibs и obs.

cbs=nn устанавлавает буферы преобразования в nn байт. Эти буферы используются при переводе из ASCII в EBCDIC, или из неблокового устройства в блоковое. Например, файлы, созданные под VMS часто имеют размер блока 512, поэтому вам придется установить cbs в 1b при считывании лент, записанных в формате VMS. Надеюсь, что вам не придется сталкиваться с этим!

skip=nbl и seek=nbl сообщают программе о том, что надо пропустить nbl блоков, соответственно, с начала ввода или с начала вывода. Конечно, последний случай имеет смысл, если указано преобразование

notrunc, см. ниже. Любой размер блока является значением ibs (obs). Будьте осторожны: если вы не установите ibs и напишите skip=1b вы действительно пропустите 512 раз по 512 байт, что будет 256KB. Это будет не совсем то, то вы хотели?

count=nbl означает, что надо копировать только nbl блоков с входа, каждый размером, указанным в ibs. Эта опция, вместе с предыдущей, становиться полезной, если, например, у вас есть испорченный файл и вы хотите восстановить как можно больше информации из него. Вы просто пропускаете часть, которую невозможно считать и получаете то, что осталось.

conv=conversion,[conversion...] преобразует файл, как указано в аргументе. Возможными преобразованиями являются ascii, которое преобразует из EBCDIC в ASCII; ebcdic и ibm, они обе выполняют обратное преобразование (не существует единственного преобразования из EBCDIC в ASCII! Первое преобразование стандартное, а второе лучше работает при печати файлов на IBM принтере); block, который дополняет записи, прерванные переходом к новой строке до размера, указанного в cbs, заменяя символ перехода к новой строке на хвостовые пробелы; unblock, который выполняет обратное действие (убирает хвостовые пробелы, и заменяет их символом перехода к новой строке); case and ucase, задает преобразование перевода на нижний регистр и верхний регистр; swab, перестанавливает каждую пару входных байтов (например, чтобы использовать файл, написанный на 680x0, содержащий короткие целые, на машине с Intel'овской архитектурой вам необходимо сделать такое преобразвание); noerror, для продолжения обработки после чтения ошибки; sync, который дополняет входной блок до размера ibs хвостовыми NUL'ами.

### 11.3.2 Примеры

Каноническим примером является пример, с которым, возможно, вам пришлось столкнуться, когда вы пытались создать свою первую Linux дискету: как записать на флоппи диск, без файловой системы MS-DOS. Решение просто:

```
% dd if=disk.img of=/dev/fd0 obs=18k count=80
```

Я решил не использовать ibs, потому что я не знаю, какой размер блока лучше для жесткого диска, но в этом случае не будет вреда, если вместо obs я использую bs - может быть это даже будет немного быстрее. Обратите внимание на явное указание количества секторов для записи (18KB это размер сектора, поэтому count устанавливается в 80) и использование низко-уровневого имени флоппи-устройства.

Другое полезное применение dd относится к резервному копированию в сетях. Давайте предположим, что мы находимся на машине альфа и что

на машине бета есть лентопротяжка /dev/rst с файлом, запакованным при помощи tar, который мы хотим получить. У нас одинаковые права на обеих машинах, но на бете нет дискового пространства, чтобы записать этот tar файл. В этом случае, мы можем написать

```
% rsh beta 'dd if=/dev/rst0 ibs=8k obs=20k' | tar xvBf -
```

чтобы сделать за один раз операцию целиком. В этом случае, нам придется использовать возможности rsh для чтения с ленты. Входные и выходные размеры устанавливаются по умолчанию для этих операций, это 8KB для чтения с tape и 20KB для записи в ethernet; с точки зрения tar на другой машине, есть тот же самый поток байтов, который мы можем получить с ленты, но он приходит в достаточно своеобразной манере, и необходима опция B.

Да, я забыл: Я совсем не думаю, что dd это акроним (аббревиатура по первым буквам) для "duplicator данных", но по крайней мере это хороший способ запомнить его смысл...

### 11.4 sort, сортировщик данных

#### 11.4.1 Введение

#### 11.4.2 Опции

#### 11.4.3 Примеры

## 12. Опечатки, Ошибки и Другие Неприятности

Unix не был спроектирован так, чтобы защищать от глупостей, так как такая стратегия не позволяет делать и умные вещи.

Doug Gwyn

### 12.1 Как Избежать Ошибок

Многие пользователи время от времени сообщают о сбоях в работе операционной системы Unix и часто это происходит из-за их собственных действий. Пользователи довольны операционной системой Unix, когда все идет хорошо, и ненавидят ее после работы поздней ночью, так как случается много неприятностей из-за того, что так мало команд требуют подтверждения. Когда пользователь хорошо выспался, он редко думает об этом, неразговорчивые команды позволяют ему работать более гладко.

Однако, есть некоторые недостатки. rm и mv никогда не запрашивают подтверждения и это часто приводит к нежелательным последствиям. Давайте изучим небольшой список советов, который поможет вам избежать подобных проблем:

- Храните резервные копии! Особенно это относится к системным администраторам - они должны регулярно делать резервные копии своих систем! Раз в неделю будет достаточно для того, чтобы спасти много файлов. Смотрите Руководство Системного Администратора LINUX для более подробного изучения.

- Каждый пользователь должен хранить свои собственные резервные копии, если это возможно. Если вы используете не одну файловую систему, постарайтесь хранить обновленные копии всех ваших файлов на

каждой системе. Если возможно использовать дисковод, лучше хранить

наиболее ценную информацию на дискетах. В крайнем случае, храните дополнительные копии ваших наиболее важных материалов в отдельном каталоге!

- Задумывайтесь о применении особенно опасных команд, таких как mv, rm, и cp, перед тем как выполнить действие. Так же следует быть осторжным при перенаправлении вывода ( > ) - таким образом, если вы не обратите внимания, можно перезаписать ваши файлы. Даже самые безобидные команды могут оказаться пагубными:

```
/home/larry/report# cp report-1992 report-1993 backups
```

может стать катастрофой:

```
/home/larry/report# cp report-1992 report-1993
```

- Автор также рекомендует, по собственному опыту, не делать сопровождение файла поздно ночью. Не кажется ли вам, что в структуре вашего каталога небольшой беспорядок? Остановитесь,- небольшой беспорядок никогда не повредит.

- Следите за текущей директорией. Иногда используемое вами приглашение на ввод не отображает название текущей директории, и это может послужить причиной ошибки. Грустно читать почту по адресу comp.unix.admin (Это группа обсуждения Usenet, в которой обсуждается вопрос администрирования Unix машин) о root-пользователе, который находится в /, а не в /tmp!  Например:

```
mousehouse> pwd
/etc
mousehouse> ls /tmp
passwd
mousehouse> rm passwd
```

Вышеперечисленные команды только сильно огорчили бы пользователя, который наблюдал бы за тем, как только что был удален файл пароля для данной системы. Без этого файла в систему будет нельзя войти!

## 12.2 В этом не ваша вина.

К несчастью, для программиств всего мира ошибки пользователя не являются единственной проблемой. Unix и Linux - это сложные системы и все их известные версии имеют ошибки. Иногда эти ошибки трудно обнаружить, и они проявляются только при определенных условиях.

Прежде всего, что же такое ошибка исполнения (bug)? Примером такой ошибки, может послужить ответ компьютера "7" на запрос о вычислении "5+3". Хотя, это тривиальный пример того, что может работать неправильно, большинство подобных ошибок в ваших программах возникает при использовании арифметики неким чрезвычайно странным образом.

### 12.2.1 Когда появляется ошибка

Когда компьютер дает неправильный ответ (удостоверьтесь, что ответ неправильный!) или ломается - это ошибка. Если какая-нибудь программа заканчивается аварийно или выдает сообщение об ошибке операционной системы - это ошибка.

Если команда не завершает исполнение, это может быть ошибка, но вы должны убедиться, что не заставили ее делать какие-нибудь действия,

которые требуют больших временных ресурсов. Попросите помощи у специалиста, если вы не знаете, что делает эта команда.

Некоторые сообщения предохранят вас от возможных ошибок. Некоторые сообщения не относятся к ошибкам исполнения. Получив подозрительное сообщение проверьте в разделе 3.3 и в какой-нибудь другой документации, что оно не является обычным инфомационным сообщением. Например, такие сообщения, как "disk full" ("диск заполнен") или "lp0 on fire" не являются проблемами программного обеспечения, но что-то не в порядке с вашей аппаратурой - недостаточно дискового пространства или плохой принтер.

Если вы не можете найти какой-нибудь информации о программе - это ошибка в документации, вам следует связаться с автором программы и

самим предложить восполнить недостаток. Если что-то неправильно в существующей документации (особенно, в этой!), это ошибка справочного руководства. Если что-то кажется неполным или непонятным в документации - это ошибка.

Если вы не можете выиграть gnuchess в шахматы - это просто недостаток вашего алгоритма игры, и совсем не обязательно "ошибка" в вашей голове.

### 12.2.2 Сообщение об ошибке

Если вы уверены, что нашли ошибку, важно убедиться, что ваша информация попадет по адресу. Постарайтесь понять в какой программе ошибка - если у вас не получается это сделать, можно обратится за помощью по адресу comp.os.linux.help или comp.unix.misc. После того, как вы нашли программу, в которой обнаружена ошибка, почитайте справочное руководство, чтобы выяснить, кто ее написал.

Самый хороший способ послать сообщение об ошибке в Linux - по электронной почте. Если у вас нет доступа к электронной почте, вы можете обратиться к тому, кто поставил вам Linux - в конце концов, обратитесь к тому, у кого есть электронная почта, или кто купил коммерческий Linux, и поэтому хочет, чтобы в Linux было как можно меньше ошибок. Помните, однако, что ни у кого нет обязательств исправлять ошибки, если это не оговорено в контракте!

Когда вы отсылаете сообщение об ошибке, включите как можно больше полезной информации. В том числе:

- Описание того, что, по вашему мнению, неправильно. Например, "Я получил 5 при сложении 2+2" или "было выдано сообщение segmentation violation -- core dumped." Важно точно сообщить, что произошло, так чтобы тот, кто сопровождает программу мог исправить вашу ошибку!

- Включите все относящиеся к делу переменные окружения.

- Версию вашего ядра (смотрите файл /proc/version) и ваших системных библиотек (смотрите директорию /lib - если вы не можете

разобраться с этим, отошлите список /lib).

- Как вы запустили программу, или, если это была ошибка ядра, что вы делали в это время.

- Вся периферийная информация. Например, команда w может не отображать текущий процесс для определенных пользователей. Не

говорите, "w не работает для некоторых пользователей". Ошибка может произойти, потому что имя пользователя состоит из восьми символов или пользователь вошел в систему по сети. Вместо этого сообщите, что "w не отображала текущий процесс пользователя greenfie когда он вошел в систему по сети"

- И помните о том, что нужно быть вежливым. Многие люди производят свободно распространяемое программное обеспечение просто для забавы, и потому, что у них доброе сердце. Не обижайте их, ведь так много разработчиков разочаровались в обществе Linux, а ведь это только начало жизненного пути Linux!

## Приложение A. Общая лицензия GNU

### Общая лицензия GNU
### Версия 2, Июнь 1991



#### Преамбула

Лицензии для большей части программного обеспечения составлены так, чтобы лишить вас свободы распространять или изменять его. Напротив, лицензия GNU General Public License гарантирует вам свободу распространения и модификации нашего программного обеспечения - одинаковую свободу для всех. Эта лицензия применима к большей части программного обеспечения Free Software Foundation и к любой другой программе, авторы которой решили воспользоваться данной лицензией. (Часть программного обеспечения Free Software Foundation охвачена лицензией GNU Library General Public License, а не этой лицензией.) Эту Лицензию вы можете также применить и для своих программ.

Когда мы говорим о свободном программном обеспечении, то обращаем внимание именно на свободу, а не на стоимость. Наши лицензии General Public Licenses составлены так, чтобы вы были уверены, что вы можете распространять копии свободного программного обеспечения (если хотите,

то не бесплатно), что вы получили исходные тексты или можете получить их, если захотите, что вы можете менять содержимое данного программного обеспечения или использовать его части для новых свободно распространяемых программ; мы хотим, чтобы вы знали, что вы можете делать все это.

Чтобы защитить ваши права, необходимо сделать ограничения, которые не позволят кому-либо лишить вас этих прав или потребовать у

вас отказаться от них. Эти ограничения накладывают определенную ответственность на вас, если вы распространяете копии данного программного обеспечения или модифицируете его.

Например, если вы распространяете копии таких программ за плату или бесплатно, вы должны передать покупателям все права, которые есть у вас. Вы должны убедиться, что они также получили или могут получить исходные тексты. Вы также обязаны показать им данные соглашения, чтобы они знали свои права.

Мы защищаем ваши права по двум аспектам: (1) авторское право на программное обеспечение, (2) предлагаем вам данную лицензию, которая дает вам официальное разрешение копировать, распространять и/или модифицировать программное обеспечение.

Кроме того, для защиты каждого автора и нашей собственной защиты, мы считаем своим долгом разъяснить, что относительно данного свободно распространяемого программного обеспечения не дается никаких гарантий. Если программное обеспечение модифицируется кем-либо еще, мы хотим, чтобы его получатели знали, что у них не оригинал, так чтобы какие-либо проблемы, возникшие по вине других, не отразились на репутации авторов оригинала.

И наконец, любой свободной распространяемой программе постоянно угрожают патенты на программное обеспечение. Мы хотели бы избежать проблем, когда вторичные распространители свободно распространяемой программы сами получат права на патент, фактически присваивая себе права данную программу. Чтобы это предотвратить, мы поясняем, что каждый патент должен быть лицензирован для всеобщего свободного пользования, либо не лицензирован вовсе.

Далее следуют конкретные условия копирования, распространения и модификации.

Соглашения и условия.

0. Данные условия лицензии касаются любой программы или прочей работы, содержащей в себе пометку, помещенную туда владельцем авторского права, и сообщающую о том, что данная программа может распространяться в соответствии с требованиями данной Лицензии (General Public License). Далее слово "Программа" относится к любой подобной программе или работе, а "работа, основанная на Программе" означает как Программу, так и любую производную работу, подчиняющуюся закону об авторском праве, т.е: работу, содержащую Программу или часть ее, дословно или с модификациями, и/или транслированную на другой язык. (Здесь и далее, понятие трансляции включается без ограничений в термин 'модификация'.) Лицо, приобретающее лицензию, обозначается в

тексте, как "вы".

Мероприятия, отличные то копирования, распространения, модификации не охвачены данной Лицензией; они находятся вне сферы ее рассмотрения. На запуск Программы не накладывается ограничений, а результат работы программы рассматривается только, если его содержимое составляет работу, основанную на Программе (независимо от того, был ли он получен при запуске программы). Верно ли это, или нет, зависит от того, что делает Программа.

1. Вы можете копировать и распространять точные копии исходных текстов Программы в таком виде, в котором вы его получили, на любом носителе, снабжая каждую копию соответствующей пометкой о правах на копирование и отказ от гарантии; оставляйте неизмененными все те пометки, которые относятся к данной Лицензии и к отсутствию какой-либо гарантии; и предоставьте любому другому получателю программы копию данной Лицензии, наряду с данной Программой. Вы можете установить плату за физический акт передачи копии, и можете, на ваше усмотрение, предложить гарантию за определенную плату.

2. Вы можете модифицировать ваши копии данной Программы или какой-либо ее части, таким образом, создавая работу, основанную на Программе, а также копировать или распространять подобные модификации или работать по соглашениям Раздела 1, при условии, что вы ознакомились со всеми данными условиями:

a. В ваших модифицированных файлах должны содержаться пометки, сообщающие о том, что вы изменили данные файлы, и содержащие дату каждого изменения.

b. Каждая работа, распространяемая или публикуемая вами, целиком или частично содержащая Программу или основанная на ней или какой-либо ее части, должна быть снабжена лицензией, позволяющей бесплатно передавать ее третьим лицам на условиях данной Лицензии.

c. Если измененная программа является интерактивной, вы должны при запуске такой программы печатать сообщение, которое включает соответствующую ссылку на авторское право, пометку об отсутствии гарантии (или, наоборот, сообщающее, что вы предоставляете гарантию), и что пользователи могут в свою очередь распространять программу при выполнении данных соглашений, а также где можно найти копию данной Лицензии. (Исключение: если первоначальная Программа является диалоговой, но не печатает такое сообщение, то и в вашей работе, основанной на данной Программе не требуется печатать такое сообщение.)

Эти требования применимы к работам, связанным с модификацией вцелом. Если некоторые части работы не основаны на Программе, и могут по разумным соображениям считаться отдельными и независимыми программами сами по себе, то данная Лицензия и ее требования не применимы к этим частям, если они будут распространяться, как отдельные работы. Но при распространении тех же самых частей как частей работы, основанной на Программе, распространение должно быть согласовано с положениями данной Лицензии, чьи ограничения относятся ко всем другим лицам, имеющим лицензию, и распространяется на всю работу целиком, и таким образом на любую ее часть, независимо от того, кем она была написана.

Этот раздел лицензии введен не для того, чтобы претендовать или оспаривать права на работу, написанную целиком вами, а для того, чтобы следить за соблюдением прав при распространении производных или коллективных программ, основанных на Программе.

В дополнение следует сказать, что объединение другой работы, не основанной на Программе с Программой (или с работой, основанной на Программе) на одном носителе или в одном томе памяти не переносит действие лицензии на эту работу.

3. Вы можете копировать или распространять Программу (или работу, основанную на Программе, согласно положениям раздела 2) в объектном коде или исполнимом виде при выполнении условий, изложенных в разделах 1 и 2, обеспечивая выполнения одного из следующих пунктов:

a. Сопровождайте его полными соответствующими исходными текстами которые должны распространяться в соответствии с положениями разделов 1 и 2, на носителе, обычно используемым при предоставлении программного обеспечивания; или

b. Сопровождайте его письменным предложением, действующим по крайней мере в течение 3 лет, о предоставлении получившему данный программный продукт, полных исходных текстов на соответствующем машинном носителе, за плату не превышающей стоимость поставки.

c. Сопровождайте его информацией, которую вы получили в качестве предложения о предоставлении соответствующих исходных текстов. (Эта альтернатива допускается только для некоммерческого распространения и только, если вы получили программу в в объектном коде или исполнимом виде с таким предложением, в соответствии с пунктом b.)

Для работ по модификации наиболее предпочтительны исходные тексты программы. Для исполнимой программы, полные исходные тексты означает исходные тексты всех модулей, которые содержит программа, плюс все соответствующие программе файлы определения интерфейса, плюс скрипты, используемые для управления компиляцией и инсталляцией исполняемой программы. Однако, в виде исключения, предоставленные исходные тексты не должны включать что-либо обычно предоставляемое (в исходных текстах

или в двоичном виде) вместе с главными компонентами (компилятором, ядром и так далее) операционной системы, на которой программа выполняется, если только эти компоненты не сопровождают исполняемые файлы.

Если распространение исполняемых или объектных кодов реализуется предоставлением возможности копирования из указанного места, то предоставления возможности копирования исходных текстов из того же места считается распространением исходных текстов, даже если третью сторону не заставляют их копировать.

4. Вы не имеете права копировать, изменять, распространять Программу, способом, отличным от того, как было специально оговорено в данной Лицензии. Любая попытка копировать, изменять, распространять Программу, не оговоренная в лицензии, неправомерна, и автоматически повлечет за собой аннуллирование ваших прав, утвержденных в данной Лицензии. Однако, стороны, получившие копии или права от вас, согласно данной Лицензии не будут лишены прав, утвержденных в данной Лицензии, если они в свою очередь не сделают каких-либо нарушений.

5. Вам не требуется выполнять соглашения данной Лицензии до тех пор, пока вы не подписали ее. Однако, ничто другое не дарует вам прав на прав на изменение или распространение Программы или производных работ, основанных на Программе. Эти действия запрещены законом, если вы не приняли данную Лицензию. Поэтому при изменении или распространении Программы (или любой другой работы, основанной на Программе) вы должны указывать то, что вы принимаете все условия и соглашения данной Лицензии относительно копирования, распространения или изменения Программы (или любой другой работы, основанной на Программе).

6. Каждый раз, как вы перераспространяете Программу (или любую другую работу, основанную на Программе), получателю автоматически должна быть передана лицензия от лицензора, позволяющая копировать, распространять, изменять, подчиняясь условиям и соглашениям лицензии. Вы можете не налагать каких-либо ограничений на права, дарованные получателю. Вы не ответственны за выполнение третьими сторонами условий данной Лицензии.

7. Если вследствие судебного решения, заявления о нарушении патента или по каким-то другим причинам (не ограниченным соглашениям о патенте) условия, наложенные на вас (в судебном порядке, по соглашению или каким-то другим причинам) противоречат условиям данной Лицензии, они не освобождают вас от выполнения соглашений данной Лицензии. Если у вас не получается распространять Программу так, чтобы выполнять одновременно обязательства по отношению к данной Лицензии и какие-то другие обязательства, тогда, как следствие,вы не можете распространять Программу совсем. Например, если патентная лицензия не позволяет бесплатное распространение Программы всеми получившими Программу непосредственно от вас или через вас, единственный способ удовлетворить обоим: и этому условию, и данной Лицензии - это полностью отказаться от распространения Программы.

Если какая-нибудь часть этого раздела становиться недействительной при каких-то конкретных обстоятельствах, то считается, что оставшиеся части, равно как и весь раздел, применимы при других обстоятельствах.

Цель этого раздела не в том, чтобы подтолкнуть вас к нарушению каких-либо патентных прав или прав собственности или оспаривать справедливость подобных требований; единственная его цель - защита целостности системы свободного распространения программного обеспечения, которая реализуется практикой общедоступных лицензий. Многие люди внесли большой вклад в широкий спектр программного обеспечения, распространяемого посредством этой системы, полагаясь на ее последовательное использование; от автора/дарителя решает, будет ли он или она распространять программное обеспечение посредством какой-то другой системы, и мы не можем навязывать этот выбор.

Целью данного раздела является уточнение того, что является следствием последующего содержания лицензии.

8. Если распространение и/или использование Программы ограничивается в определенных странах патентами, исходный владелец патента поместивший Программу под действие данной Лицензии, может добавить ограничение на географическое распространение, исключив

соответствующие страны, так что распространение будет разрешено только среди стран, которые не были исключены. В таком случае, данная

Лицензия включает это ограничение в число своих соглашений, как если бы оно было одним из основных.

9. Фонд свободно распространяемого программного обеспечения (Free Software Foundation) может время от времени публиковать пересмотренную и/или новую версию Общего Положения Лицензии GNU. Новые версии будут похожи на данную, но могут отличаться в деталях, связанных с вновь возникшими проблемами.

Каждая версия имеет уникальный номер. Если в Программе определяется версии лицензии, который применим к ней и ко всем более поздним версиям, вы можете следовать соглашениям и условиям этой версии или какой-нибудь более поздней версии, выпущенной Free Software Foundation. Если в Программе не определяется версии данной Лицензии, вы можете выбирать любую версию, когда либо выпущенную Free Software Foundation.

10. Если вы хотите включить части Программы в другие свободно распространяемые программы, чьи условия распространения отличаются от наших, напишите автору, чтобы спросить разрешение. Относительно программного обеспечения, защищенного copyright Free Software Foundation, авторское право принадлежит Free Software Foundation, напишите в Free Software Foundation; мы иногда делаем подобные исключения. Наше решение будет руководствоваться двумя целями: сохранения статуса свободно распространяемого программного обеспечения для всех производных программ нашего свободного программного обеспечения и содействием в распределении и повторном использовании программного обеспечения в целом.

БЕЗ ГАРАНТИИ.

11. ПОСКОЛЬКУ ПРОГРАММА ЛИЦЕНЗИРУЕТСЯ КАК СВОБОДНО РАСПРОСТРАНЯЕМАЯ, ДЛЯ НЕЕ НЕ ПРЕДОСТАВЛЯЕТСЯ НИКАКОЙ ГАРАНТИИ, ЗА ИСКЛЮЧЕНИЕМ ПРЕДУСМАТРИВАЕМОЙ СООТВЕТСТВУЮЩИМ ЗАКОНОМ. ЗА ИСКЛЮЧЕНИЕМ СЛУЧАЕВ, КОГДА ПИСЬМЕННО УКАЗАНО ОБРАТНОЕ, ВЛАДЕЛЬЦЫ АВТОРСКИХ ПРАВ И/ИЛИ ДРУГИЕ СТОРОНЫ, ПОСТАВЛЯЮТ ПРОГРАММУ "КАК ЕСТЬ", БЕЗ КАКИХ-ЛИБО

ГАРАНТИЙ, ЯВНЫХ ИЛИ НЕЯВНЫХ ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОДРАЗУМЕВАЕМЫМИ ГАРАНТИЯМИ ВОЗМОЖНОСТИ ПРОДАЖИ И ПРИГОДНОСТИ ДЛЯ КАКИХ-ТО КОНКРЕТНЫХ НУЖД. ВЕСЬ РИСК, СВЯЗАННЫЙ С КАЧЕСТВОМ И ПРОИЗВОДИТЕЛЬНОСТЬЮ ПРОГРАММЫ ЛОЖИТСЯ НА ВАС. В СЛУЧАЕ, ЕСЛИ ДОКАЗЫВАЕТСЯ НАЛИЧИЕ В ПРОГРАММЕ ОШИБКА, ВСЕ РАСХОДЫ ПО НЕОБХОДИМОМУ ОБСЛУЖИВАНИЮ, ИСПРАВЛЕНИЯМ ИЛИ ВОССТАНОВЛЕНИЮ ВЫ ПРИНИМАЕТЕ НА СЕБЯ.

12. НИ В КАКИХ СЛУЧАЯХ, ЗА ИСКЛЮЧЕНИЕМ СЛУЧАЕВ ИСПОЛНЕНИЯ ЗАКОНА ИЛИ СОГЛАСИЯ В ПИСЬМЕННОЙ ФОРМЕ, ЛЮБОЙ ВЛАДЕЛЕЦ АВТОРСКОГО ПРАВА ИЛИ ДРУГОЕ ЛИЦО, ИМЕЮЩЕЕ ПРАВО ИЗМЕНЯТЬ И/ИЛИ РАСПРОСТРАНЯТЬ ПРОГРАММУ, КАК ИЗЛОЖЕНО ВЫШЕ, НЕ НЕСЕТ ОТВЕТСТВЕННОСТЬ ЗА УЩЕРБ, ПРИНЕСЕННЫЙ ЛЮБЫМИ ОШИБКАМИ, ВКЛЮЧАЯ ОБЩИЕ, ЧАСТНЫЕ, СЛУЧАЙНЫЕ ИЛИ ПЕРМАНЕНТНЫЕ ПОВРЕЖДЕНИЯ, ПРОИЗОШЕДШИМИ ПРИ ИСПОЛЬЗОВАНИИ ПРОГРАММЫ ИЛИ ВЫТЕКАЮЩИЕ ИЗ НЕВОЗМОЖНОСТИ ЕЕ ИСПОЛЬЗОВАНИЯ (ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОТЕРЕЙ ИЛИ НЕВЕРНОМУ ПРЕДСТАВЛЕНИЮ ДАННЫХ, ИЛИ УБЫТКАМИ, ПОНЕСЕННЫМИ ВАМИ ИЛИ ТРЕТЬЕЙ СТОРОНОЙ ИЛИ НЕВОЗМОЖНОСТИ СОВМЕСТНОГО ИСПОЛЬЗОВАНИЯ ПРОДУКТА СОВМЕСТНО С ДРУГИМ ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ), ДАЖЕ В СЛУЧАЕ, ЕСЛИ ВЛАДЕЛЕЦ БЫЛ ПРЕДУПРЕЖДЕН О ВОЗМОЖНОСТИ ПОДОБНЫХ ПРОБЛЕМ.

Как применять данные соглашения к вашим новым программам

Если вы разрабатываете новую программу, и хотите, чтобы она нашла наиболее широкое применение, то самый лучший способ достичь этого - сделать ее свободно распространяемой, чтобы каждый мог распространять и изменять ее, подчиняясь определенным соглащениям.

Если вы хотите сделать программу свободно распространяемой, добавьте следующие замечания к программе. Лучше написать их в начале каждого исходного файла, добавив, что программа предоставляется без гарантии; и каждый файл должен, как минимум, содержать строку "copyright" и ссылку на то, где находится полное сообщение.

> Одна строка сообщает имя программы и краткое изложение того, что она делает.
>
> Copyright @ 19yy имя автора
>
> Эта программа является свободно распространяемой; вы можете перераспространять и/или модифицировать ее, в соответствии с условиями GNU General Public License, опубликованной Free Software Foundation; версии 2, или (по вашему выбору) более поздней версии.
>
> Эта программа свободно распространяется в надежде, что она будет полезной, но БЕЗ КАКОЙ-ЛИБО ГАРАНТИИ, даже без гарантии того, что программа применима для ваших целям. Для получения дополнительной информации смотрите общие положения лицензии GNU.
>
> Вы должны были получить копию GNU General Public License вместе с программой; если этого не произошло, напишите в Free Software Foundation, Inc.,685 Mass Ave, Cambridge, MA 02139,

Напишите также, как связаться с вами по электронной и по обычной почте.

Если программа диалоговая, сделайте краткое сообщение наподобие этого в начале работы диалоговой программы:

> Gnomovision version 69, Copyright @ 19yy имя автора
> Gnomovision предоставляется АБСОЛЮТНО БЕЗ ВСЯКОЙ ГАРАНТИИ;
> для более подробной информации напишите 'show w'.
> Это свободно распространяемая программа, и вы можете
> перереаспространять ее, подчиняясь некоторым условиям;
> для более подробной информации напишите 'show c'.

Гипотетические команды 'show w' и 'show c' будут показывать соответствующую часть лицензии GNU. Конечно, команды, которые вы будете использовать могут называться как-нибудь по-другому, а не 'show w' и 'show c'; это может быть даже нажатие кнопки мыши или выбор пункта меню - то, что больше подходит вашей программе.

Кроме того, в любом случае, вам нужно указать вашего работодателя (если вы работаете как программист), и "отказ от авторских прав" на программу, сли это необходимо. Ниже приведен пример (измените имена):

> Voyodyne, Inc., настоящим отказывается от всех своих авторских прав на программу 'Gnomovision' (которая осуществляет проход компилятора), автор программы James Hacker.

подпись Ty Coon, 1 апреля 1989
Ty Coon, президент.

Общие положения лицензии GNU не позволяют включения вашей программы в программы, являющиеся частной собственностью. Если ваша программа - это библиотека подпрограмм, то более полезно разрешить подключение ее к патентованым приложениям. Если это то,что вам нужно, используйте общие положения лицензии библиотеки GNU вместо этой лицензии.

Далее приводится оригинальный англоязычный текст этого документа.

# GNU GENERAL PUBLIC LICENSE
## Version 2, June 1991



### Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software---to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid

anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the

source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions
~~~~~ ~~~ ~~~~~~~~~~

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The ``Program'', below, refers to any such program or work, and a ``work based on the Program'' means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term ``modification''.) Each licensee is addressed as ``you''.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License
~~~~~

along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that

users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent

issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and ``any later version'', you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by

the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our

free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM ``AS IS'' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the ``copyright'' line and a pointer to where the full notice is found.

> one line to give the program's name and an idea of what it does.
> Copyright (C) 19yy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139,

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) 19yy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items---whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a ``copyright disclaimer'' for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

## Приложение B. Общая лицензия библиотеки GNU

### Общая лицензия библиотеки GNU
Версия 2, Июнь 1991

Разрешается копировать и распространять точные копии этого лицензионного документа, но изменение его содержания запрещено.

[Это первая версия GNU Library General Public License (Общей лицензии библиотеки GNU). Она имеет номер 2, для того, чтобы соответствовать второй версии GNU General Public License.]

Преамбула

Лицензии для большей части программного обеспечения составлены так, чтобы лишить вас свободы распространять или изменять его. Напротив, лицензия GNU General Public License гарантирует вам свободу распространения и модификации нашего программного обеспечения - одинаковую свободу для всех.

Данная Лицензия, Library General Public Licence, применима к специально разработанному Free Software Foundation програмному обеспечению и к некоторым другим библиотекам, чьи авторы решили воспользоваться ей. Вы можете ее использовать и для своих библиотек.

Когда мы говорим о свободном программном обеспечении, то обращаем внимание именно на свободу, а не на стоимость. Наши лицензии General Public Licenses составлены так, чтобы вы были уверены, что вы можете распространять копии свободного программного обеспечения (если хотите, то не бесплатно), что вы получили исходные тексты или можете получить их, если захотите, что вы можете менять содержимое данного программного обеспечения или использовать его части для новых свободно распространяемых программ; мы хотим, чтобы вы знали, что вы можете

делать все это.

Чтобы защитить ваши права, необходимо сделать ограничения, которые не позволят кому-либо лишить вас этих прав или потребовать у вас отказаться от них. Эти ограничения накладывают определенную ответственность на вас, если вы распространяете копии данного программного обеспечения или модифицируете его.

Например, если вы распространяете копии этой библиотеки за плату или бесплатно, вы должны передать покупателям все права, которые есть у вас. Вы должны убедиться, что они также получили или могут получить исходные тексты. Если вы линкуете программу с библиотекой, вы должны предоставить все объектные файлы покупателю, чтобы он мог перелинковать их с библиотекой, после внесения изменения в библиотеку и ее перекомпиляции. Вы также обязаны показать им данные соглашения, чтобы они знали свои права.

Мы защищаем ваши права по двум аспектам: (1) авторское право на программное обеспечение, (2) предлагаем вам данную Лицензию, которая дает вам официальное разрешение копировать, распространять и/или модифицировать программное обеспечение.

Кроме того, с целью защиты распространителей, мы считаем своим долгом разъяснить, что относительно данного свободно распространяемого программного обеспечения не дается никаких гарантий. Если библиотека модифицируется кем-либо еще, мы хотим, чтобы ее получатели знали, что у них не оригинал, так чтобы какие-либо проблемы, возникшие по вине других, не отразились на репутации авторов оригинала.

И наконец, любой свободной распространяемой программе постоянно угрожают патенты на программное обеспечение. Мы хотели бы избежать проблем, когда вторичные распространители свободно распространяемой

программы сами получат права на патент, фактически присваивая себе права данную программу. Чтобы это предотвратить, мы поясняем, что каждый патент должен быть лицензирован для всеобщего свободного пользования, либо не лицензирован вовсе.

Большая часть программного обеспечения GNU, включая некоторые библиотеки, охвачена GNU General Public License (Общей лицензией GNU), которая была написана для программ-утилит. Данная Лицензия, GNU Library General Public License, предназначена для определенных библиотек. Эта лицензия сильно отличается от GNU General Public License; прочтите ее полностью, и не думайте, что все ее положения совпадают с положениями GNU General Public License.

Нам пришлось написать отдельную лицензию для нескольких библиотек из-за того, для библиотек стирается различия между изменением или добавлением чего-либо в программу и простым ее использованием. Линковка программы с библиотекой, без изменения библиотеки, в некотором смысле просто использование библиотеки, и аналогично исполнению программной утилиты или программы-приложения. Однако, в буквальном и юридическом смысле, собранный исполнимый модуль - это совместная работа, производная от исходной библиотеки, и GNU General Public License рассматривает ее как таковую.

Из-за этого различия, использование GNU General Public License для библиотек недостаточно эффективно способствовало распросранению программного обеспечения, так как большинство разработчиков не использовали эти библиотеки. Мы решили, что ослабление условий может содействовать распространению программного обеспечения.

Однако, ничем не ограниченная линковка программ, которые не являются свободно распространяемыми, будет лишать их пользователей всей выгоды от свободного статуса самих библиотек. Цель данной Library General Public Licenses - позволить разработчикам коммерческого программного обеспечения использовать свободно распространяемые библиотеки, сохраняя ваше право как пользователя таких программ вносить изменения в свободно распространяемые библиотеки, встроенные в эти программы. Мы надеемся, что это приведет к ускорению развития библиотек.

Далее следуют точные соглашения и условия по копированию, распространению и модификации. Обратите особое внимание на разницу между "программой, основанной на использовании библиотеки" и "программой, использующей эту библиотеку". Первая содержит код, являющийся производным от библиотеки, в то время как вторая работает совместно с библиотекой.

Обратите внимание, что может случиться так, что библиотека подчиняется условиям обычной General Public License, а не специальной лицензии.

Соглашения и условия о копировании, распространении и изменении.

0. Нижеследующие соглашения лицензии относятся к любой библиотеке, содержащей пометку, помещенную владельцем авторского права, говорящей о том, что данная программа может распространяться в соответствии с требованиями данной Лицензии (General Public License) (также называемой 'данная Лицензия'). Лицо, приобретающее лицензию, в тексте обозначается как "вы".

Библиотека - это набор функций и/или данных, объединенных удобным образом для линковки с программами-приложениями (которая использует некоторые из этих данных и функций) с целью создания исполнимого модуля.

Далее слово "Библиотека" относится к любой подобной библиотеке или работе, распространяемой в соответствии с соглашениями данной Лицензии. "Работа, основанная на Библиотеке" означает как библиотеку, так и любую производную работу, подчиняющуюся закону об авторском праве, т.е: работу, содержащую Библиотеку или часть ее, дословно или с модификациями, и/или транслированную на другой язык. (Здесь и далее, понятие трансляции включается без ограничений в термин 'модификация'.)

Работа с исходными текстами наиболее удобна при внесении изменений в программу. Для библиотеки полный исходный текст означает все "исходники" для всех модули, которые она содержит, плюс все соответствующие файлы, определяющие интерфейсы, плюс скрипты, используемые для управления компиляцией и инсталляцией библиотеки.

Мероприятия, отличные то копирования, распространения, модификации не охвачены данной Лицензией; они находятся вне сферы ее рассмотрения. На запуск программы, использующей Библиотеку, не

накладывается ограничений, а результат работы программы рассматривается только, если его содержимое составляет работу, основанную на Библиотеке (независимо от использования Библиотеки в качестве инструмента для написания программы). Верно ли это, или нет, зависит от того, что делает Библиотека, и что делает программа, использующая Библиотеку.

1. Вы можете копировать и распространять точные копии полных исходных текстов Библиотеки в таком виде, в котором вы его получили, на любом носителе, снабжая каждую копию соответствующей пометкой о правах на копирование и отказ от гарантии; оставляйте неизмененными все те пометки, которые относятся к данной Лицензии и к отсутствию какой-либо гарантии; и предоставьте любому другому получателю программы копию данной Лицензии, наряду с данной Библиотекой. Вы можете установить плату за физический акт передачи копии, и можете, на ваше усмотрение, предложить гарантию за определенную плату.

2. Вы можете модифицировать ваши копии данной Библиотеки или какой-либо ее части, таким образом, создавая работу, основанную на использовании Библиотеки, а также копировать или распространять подобные модификации или работать в соответствии с соглашениями Раздела 1, при условии, что вы ознакомились со всеми данными условиями:

a. Модифицированная работа сама должна быть Библиотекой.

b. В ваших модифицированных файлах должны содержаться пометки о том, что вы изменили файл, содержащие дату каждого изменения.

c. Каждая работа, должна быть лицензированна как единое целое по соглашениям данной Лицензии, и бесплатно предоставлена любой третьей стороне.

d. Если некая процедура в измененной Библиотеке нуждается в функции или таблице данных, предоставляемой прикладной программой, использующей эту процедуру, и эта функция или данные не передаются процедуре в качестве аргумента, то вы должны гарантировать, что библиотечная процедура корректно работат в любом случае, вне

зависимости от того, предоставляет ли ей приложение нужный ресурс.

(Например, библиотечная функция, вычисляющая квадратный корень, полностью корректно определена независимо от приложения. Поэтому, в разделе 2d требуется, чтобы любая предоставляемая приложением функция или таблица, используемая этой функцией имела право отсутствовать: в случае, когда приложение не предоставляет эту функцию или таблицу, функция вычисления квадратного корня должна, тем не менее, вычислять квадратный корень.)

Эти требования применимы к работам, связанным с модификацией вцелом. Если некоторые части работы не основаны на Библиотеке, и могут по разумным соображениям считаться отдельными и независимыми программами сами по себе, то данная Лицензия и ее требования не применимы к этим частям, если они будут распространяться, как отдельные работы. Но при распространении тех же самых частей как частей работы, основанной на Библиотеке, распространение должно быть согласовано с положениями данной Лицензии, чьи ограничения относятся ко всем другим лицам, имеющим лицензию, и распространяется на всю работу целиком, и таким образом на любую ее часть, независимо от того, кем она была написана.

Этот раздел лицензии введен не для того, чтобы претендовать или оспаривать права на работу, написанную целиком вами, а для того, чтобы следить за соблюдением прав при распространении производных или коллективных программ, основанных на Библиотеке.

В дополнение следует сказать, что объединение другой работы, не основанной на Библиотеке с Библиотекой (или с работой, основанной на Библиотеке) на одном носителе или в одном томе памяти не переносит действие лицензии на эту работу.

3. Вы можете по своему выбору применять соглашения обычной General Public License вместо данной Лицензии относительно данной копии Библиотеки. Для того, чтобы сделать это вам нужно все пометки, ссылающиеся на данную Лицензию изменить так, чтобы они ссылались на General Public License, версия 2, а не на эту Лицензию. (Если появилась версия более новая чем General Public License, версия 2, вы

можете, если захотите, использовать именно эту версию.) Но не делайте каких-либо других изменений в этих пометках.

После того, как в предоставленную копию были внесены изменения, копия не может быть возвращена в исходное состояние, поэтому General Public License будет применяться ко всем последующим копиям и всем производным работам, основанным на этой копии.

Такая политика имеет смысл, когда вы хотите скопировать часть кода Библиотеки в программу, не являющейся библиотекой.

4. Вы можете копировать или распространять Библиотеку (или ее часть, или работу, основанную на Библиотеке, согласно положениям раздела 2) в объектном коде или исполнимом виде при выполнении условий разделов 1 и 2, сопровождая его полными соответствующими исходными текстами, который должен распространяться согласно требованиям разделов 1 и 2 на носителе, обычно используемом для предоставлении программного обеспечивания;

Если распространение исполняемых или объектных кодов реализуется предоставлением возможности копирования из указанного места, то

предоставления возможности копирования исходных текстов из того же места считается распространением исходных текстов, даже если третью сторону не заставляют их копировать.

5. Программа, не содержащая производных какой-либо части Библиотеки, но спроектированная так, чтобы использовать Библиотеку при компиляции и линковке называется "работой, использующей Библиотеку". Такая работа, сама по себе не является производной работой от Библиотеки, и поэтому она не попадает в область действия данной Лицензии.

Однако, линковка "работы, использующей Библиотеку" с Библиотекой создает исполняемый код, который является производной от Библиотеки (так как содержит часть Библиотеки), в отличии от "работы, которая использует Библиотеку". Поэтому исполняемый код находится в сфере действия данной Лицензии. В части 6 сформулированы положения относительно распространения такого исполняемого кода.

Когда "работа, использующая Библиотеку" использует данные из заголовочного файла, который является частью библиотеки, объектный код для такой работы может являться производной работой от Библиотеки, даже если исходный текст таковым не является. Истинно ли это или нет, особенно важно когда работа может быть собрана без Библиотеки или сама является библиотекой. Точных определений в данном случае не существует.

Если такой объектный файл использует только числовые параметры, описание структур данных и способов доступа к ним, небольших макросов и inline-функций (длиной до десяти строк), то использование такого объектного модуля неограниченно, вне зависимости от того, является ли данная работа законно основанной на Библиотеке работой. (Исполняемые модули, содержащие этот объектный код и часть Библиотеки, все еще остаются под действием положений раздела 6).

В противном случае, если работа является производной работой от Библиотеки, вы можете распространять объектный код данной программы согласно положениям раздела 6. Любой исполнимый код, содержащий эту программу, также попадает под действие положений раздела 6, независимо от того, был ли он собран непосредственно с Библиотекой.

6. Как исключение из предыдущих частей вы можете компилировать и линковать "работу, использующую Библиотеку" с Библиотекой для получения работы, содержащей части Библиотеки и распространять эту работу согласно требованиям по вашему выбору, при условии что эти требования позволяют изменять программу для собственного использования заказчиком и позволяют применять дизассемблирование в отладочных целях.

Вы должны поместить сообщение в каждой копии работы о том, что в ней используется Библиотека и что Библиотека и ее использование находятся под действием данной Лицензии. Вы должны также предоставить копию данной Лицензии. Если программа во время исполнения выдает информацию об авторских правах, вы должны туда включить сообщение об авторских правах на Библиотеку, также как и ссылку на копию данной Лицензии. Кроме того, Вы должны выполнить одно из следующих условий:

а. Сопровождайте вашу работу полным соотвествующим исходным текстом Библиотеки на машинном носителе, включая все изменения, которые были использованы в работе (которая должна распространяться

согласно положениям части 1 и 2); а, если работа является исполняемым кодом с прилинкованной Библиотекой, то сопровождайте ее полной "работой, использующей Библиотеку", то есть ее объектным кодом и/или ее исходным текстом с тем, чтобы пользователь мог изменить Библиотеку и затем перелинковать, чтобы создать исправленные исполняемые модули, содержащие изменения Библиотеки. (Ясно, что пользователю изменившему содержимое файлов определений Библиотеки не обязательно нужно иметь возможность перекомпилировать приложение чтобы пользоваться изменениями в определениях.)

b. Сопровождайте программу письменным предложением, имеющим силу по крайней мере три года, о предоставлении пользователю материалов, указанных в части 6а, за плату не превышающую стоимость поставки этих материалов.

c. Если распространение данной программы осуществлено посредством предоставления доступа для копирования из определенного источника, предложите точно такой же доступ для копирования вышеуказанных материалов оттуда же.

d. Убедитесь, что пользователь получил копии этих материалов, или, что вы уже отослали ему их.

Что касается исполняемого кода, согласно правилам для "работы, использующей Библиотеку", передаваемый продукт должен содержать все данные и вспомогательные программы, необходимые для воспроизведения с их помощью исполняемого кода. Однако, в виде специального исключения, распространяемый исходный текст не должен содержать ничего из того, что обычно распространяется (в исходных текстах, или в двоичном виде) вместе с основными компонентами (компилятор, ядро, и т.д.) операционной системы для которой предназначен исполняемый код, только если эта компонента сама не сопровождает исполняемый код.

Может случиться, что эти требования противоречат лицензионным ограничениям других коммерческих библиотек, которые обычно не сопровождают операционную систему. Такое противоречие означает, что вы не можете совместно использовать их и Библиотеку в том исполняемом модуле, который вы распространяете.

7. Вы можете поместить библиотечные возможности, на которых основана ваша работа, в отдельную библиотеку, вместе с возможностями другой библиотеки, не охваченной положениями данной Лицензии, и распространять такую комбинированную библиотеку, обеспечив раздельное распространение работы, основанной на Библиотеке, и работы, основанной на возможностях другой библиотеки, и выполнив одно из двух условий:

a. Сопровождайте комбинированную библиотеку копией той же самой работы, основанной на Библиотеке, но некомбинированной, то есть без использования возможностей какой-либо другой библиотеки. Это распространение должно подчиняться условиям предыдущих разделов.

b. В комбинированной библиотеке выдавайте сообщения о том, что частью ее является работа, основанная на Библиотеке, в котором говорится, где можно найти некомбинированный вариант той же самой работы.

8. Вы не имеете права копировать, изменять, линковать, распространять Библиотеку, способом, отличным от тех, что были специально оговорены в этой Лицензии. Любая попытка копировать,

изменять, линковать, распространять Библиотеку, не оговоренная в лицензии, неправомерна, и автоматически будет аннулировать ваши права, утвержденные в этой Лицензии. Однако, лица, получившие копии или права от вас, согласно этой Лицензии не будут лишены прав, утвержденных в этой Лицензии, если это стороны в свою очередь не сделают каких-либо нарушений.

9. Вам не требуется выполнять соглашения данной Лицензии до тех пор, пока вы не подписали ее. Однако, ничто другое не дарует вам прав на изменение или распространение Библиотеки или производных работ, основанных на Библиотеке. Эти действия запрещены законом, если вы не

приняли данную Лицензию. Поэтому при изменении или распространении Библиотеки (или любой другой работы, на ней основанной) вы должны указывать то, что вы принимаете все условия и соглашения данной лицензии относительно копирования, распространения или изменения Библиотеки (или любой другой работы, на ней основанной).

10. Каждый раз, когда вы перераспространяете Библиотеку (или любую другую работу, на ней основанную), получателю автоматически должна быть передана Лицензия от лицензора, чтобы копирование, распространение, линковку и модификацию Библиотеки, в соответствии с условиями и соглашениями Лицензии. Вы не можете налагать каких-либо ограничений на права, дарованные получателю. Вы не ответственны за выполнение третьими сторонами условий данной Лицензии.

11. Если вследствие судебного решения, заявления о нарушении патента или по каким-то другим причинам (не ограниченным соглашениям о патенте) условия, наложенные на вас (в судебном порядке, по соглашению или каким-то другим причинам) противоречат условиям данной лицензии, они не освобождают вас от выполнения соглашений этой лицензии. Если у вас не получается распространять Библиотеку так, чтобы выполнять одновременно обязательства по отношению к данной Лицензии и какие-то другие обязательства, тогда, как следствие, вы не можете распространять Библиотеку совсем. Например, если патентная лицензия не позволяет бесплатное распространение Библиотеки всеми получившими Библиотеку непосредственно от вас или через вас, единственный способ удовлетворить обоим: и этому условию, и данной Лицензии - это полностью отказаться от распространения Библиотеки.

Если какая-нибудь часть этого раздела становиться недействительной при каких-то конкретных обстоятельствах, то считается, что оставшиеся части, равно как и весь раздел, применимы при других обстоятельствах.

Цель этого раздела не в том, чтобы подтолкнуть вас к нарушению каких-либо патентных прав или прав собственности или оспаривать справедливость подобных требований; единственная его цель - защита целостности системы свободного распространения программного обеспечения, которая реализуется практикой общедоступных лицензий.

Многие люди внесли большой вклад в широкий спектр программного обеспечения, распространяемого посредством этой системы, полагаясь на ее последовательное использование; от автора/дарителя решает, будет ли он или она распространять программное обеспечение посредством какой-то другой системы, и мы не можем навязывать этот выбор.

Целью данного раздела является уточнение того, что является следствием последующего содержания лицензии.

12. Если распространение и/или использование Библиотеки

ограничивается в определенных странах патентами, владелец исходных авторских прав, поместивший Библиотеку в сферу действия данной Лицензии, может добавить ограничение на географическое распространение, исключив соответствующие страны, так что распространение будет разрешено только среди стран, которые не были исключены. В таком случае, данная Лицензия включает это ограничение в число своих соглашений, как если бы оно было одним из основных.

13. Фонд свободно распространяемого программного обеспечения (Free Software Foundation) может время от времени публиковать пересмотренную и/или новую версию GNU Library General Public License. Такие новые версии будут похожи на данную, но могут отличаться в деталях, связанных с вновь возникшими проблемами.

Каждая версия имеет уникальный номер. Если в Библиотеке определяется версии лицензии, который применим к ней и ко всем более поздним версиям, вы можете следовать соглашениям и условиям этой версии или какой-нибудь более поздней версии, выпущенной Free Software Foundation. Если в Библиотеке не определяется версии данной Лицензии, вы можете выбирать любую версию, когда либо выпущенную Free Software Foundation.

14. Если вы хотите включить части Библиотеки в другие свободно распространяемые программы, чьи условия распространения отличаются от наших, напишите автору, чтобы спросить разрешение. Относительно программного обеспечения, защищенного copyright Free Software Foundation, авторское право принадлежит Free Software Foundation, напишите в Free Software Foundation; мы иногда делаем подобные

исключения. Наше решение будет руководствоваться двумя целями: сохранения статуса свободно распространяемого программного обеспечения для всех производных программ нашего свободного программного обеспечения и содействием в распределении и повторном использовании программного обеспечения в целом.

БЕЗ ГАРАНТИИ.

15. ПОСКОЛЬКУ БИБЛИОТЕКА ЛИЦЕНЗИРУЕТСЯ КАК СВОБОДНО РАСПРОСТРАНЯЕМАЯ, ДЛЯ НЕЕ НЕ ПРЕДОСТАВЛЯЕТСЯ НИКАКОЙ ГАРАНТИИ, ЗА ИСКЛЮЧЕНИЕМ ПРЕДУСМАТРИВАЕМОЙ СООТВЕТСТВУЮЩИМ ЗАКОНОМ. ЗА ИСКЛЮЧЕНИЕМ СЛУЧАЕВ, КОГДА ПИСЬМЕННО УКАЗАНО ОБРАТНОЕ, ВЛАДЕЛЬЦЫ АВТОРСКИХ ПРАВ И/ИЛИ ДРУГИЕ СТОРОНЫ, ПОСТАВЛЯЮТ БИБЛИОТЕКУ "КАК ЕСТЬ", БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, ЯВНЫХ ИЛИ НЕЯВНЫХ ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОДРАЗУМЕВАЕМЫМИ ГАРАНТИЯМИ ВОЗМОЖНОСТИ ПРОДАЖИ И ПРИГОДНОСТИ ДЛЯ КАКИХ-ТО КОНКРЕТНЫХ НУЖД. ВЕСЬ РИСК, СВЯЗАННЫЙ С КАЧЕСТВОМ И ПРОИЗВОДИТЕЛЬНОСТЬЮ БИБЛИОТЕКИ ЛОЖИТСЯ НА ВАС. В СЛУЧАЕ, ЕСЛИ ДОКАЗЫВАЕТСЯ НАЛИЧИЕ В БИБЛИОТЕКЕ ОШИБКА, ВСЕ РАСХОДЫ ПО НЕОБХОДИМОМУ ОБСЛУЖИВАНИЮ, ИСПРАВЛЕНИЯМ ИЛИ ВОССТАНОВЛЕНИЮ ВЫ ПРИНИМАЕТЕ НА СЕБЯ.

16. НИ В КАКИХ СЛУЧАЯХ, ЗА ИСКЛЮЧЕНИЕМ СЛУЧАЕВ ИСПОЛНЕНИЯ ЗАКОНА ИЛИ СОГЛАСИЯ В ПИСЬМЕННОЙ ФОРМЕ, ЛЮБОЙ ВЛАДЕЛЕЦ АВТОРСКОГО ПРАВА ИЛИ ДРУГОЕ ЛИЦО, ИМЕЮЩЕЕ ПРАВО ИЗМЕНЯТЬ И/ИЛИ РАСПРОСТРАНЯТЬ БИБЛИОТЕКУ, КАК ИЗЛОЖЕНО ВЫШЕ, НЕ НЕСЕТ ОТВЕТСТВЕННОСТЬ ЗА УЩЕРБ, ПРИНЕСЕННЫЙ ЛЮБЫМИ ОШИБКАМИ, ВКЛЮЧАЯ ОБЩИЕ, ЧАСТНЫЕ, СЛУЧАЙНЫЕ ИЛИ ПЕРМАНЕНТНЫЕ ПОВРЕЖДЕНИЯ, ПРОИЗОШЕДШИМИ ПРИ ИСПОЛЬЗОВАНИИ БИБЛИОТЕКИ ИЛИ ВЫТЕКАЮЩИЕ ИЗ НЕВОЗМОЖНОСТИ ЕЕ ИСПОЛЬЗОВАНИЯ (ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОТЕРЕЙ ИЛИ НЕВЕРНОМУ ПРЕДСТАВЛЕНИЮ ДАННЫХ, ИЛИ УБЫТКАМИ, ПОНЕСЕННЫМИ ВАМИ ИЛИ ТРЕТЬЕЙ СТОРОНОЙ ИЛИ НЕВОЗМОЖНОСТИ СОВМЕСТНОГО ИСПОЛЬЗОВАНИЯ ПРОДУКТА СОВМЕСТНО С ДРУГИМ ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ), ДАЖЕ В СЛУЧАЕ, ЕСЛИ ВЛАДЕЛЕЦ БЫЛ ПРЕДУПРЕЖДЕН О ВОЗМОЖНОСТИ ПОДОБНЫХ ПРОБЛЕМ.

Как применять данные соглашения к вашим новым библиотекам

Если вы разрабатываете новую библиотеку, и хотите, чтобы она нашла наиболее широкое применение, то самый лучший способ достичь этого - сделать ее свободно распространяемой, чтобы каждый мог распространять и изменять ее, подчиняясь определенным соглашениям. Вы

можете сделать это, разрешив перераспространение согласно вышеперечисленным положениям (или, согласно положениям GNU General Public License).

Чтобы ратифицировать эти условия, добавьте в библиотеку несколько замечаний. Лучше написать их в начале каждого исходного файла, добавив, что библиотека предоставляется без гарантии; и каждый файл должен, как минимум, содержать строку "copyright" и ссылку на то, где находится полное сообщение.

> Одна строка сообщает имя библиотеки и краткое изложение того, что она делает.
>
> Copyright (C) год имя автора
>
> Эта библиотека является свободно распространяемой; вы можете перераспространять и/или модифицировать ее, в соответствии с условиями GNU Library General Public License, опубликованной Free Software Foundation; версии 2, или (по вашему выбору) более поздней версии.
>
> Эта библиотека свободно распространяется в надежде, что она будет полезной, но БЕЗ КАКОЙ-ЛИБО ГАРАНТИИ, даже без гарантии того, что библиотека применима для ваших целей. Для получения дополнительной информации смотрите общие положения лицензии GNU.
>
> Вы должны были получить копию GNU Library General Public License вместе с библиотекой; если этого не произошло, напишите в Free Software Foundation, Inc.,685 Mass Ave, Cambridge, MA 02139, USA

Напишите также, как связаться с вами по электронной и по обычной почте.

Кроме того, в любом случае, вам нужно указать вашего работодателя (если вы работаете как программист), и "отказ от авторских прав" на программу, если это необходимо. Ниже приведен пример (измените имена):

> Voyodyne, Inc., настоящим отказывается от всех своих авторских прав на библиотеку 'Frob' (которая бьет баклуши), созданной James Random Hacker.
>
> подпись Ty Coon, 1 апреля 1989
> Ty Coon, президент.

Далее приводится оригинальный англоязычный текст этого документа.

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License,

applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a ``work based on the library'' and a ``work that uses the library''. The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

## Terms and Conditions for Copying, Distribution and Modification

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called ``this License''). Each licensee is addressed as ``you''.

A ``library'' means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The ``Library'', below, refers to any such software library or work which has been distributed under these terms. A ``work based on the Library'' means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term ``modification''.)

``Source code'' for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that

you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. The modified work must itself be a software library.

b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you

distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a ``work that uses the Library''. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a ``work that uses the Library'' with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a ``work that uses the library''. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a ``work that uses the Library'' uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not.

Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a ``work that uses the Library'' with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally

distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free

redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY
~~ ~~~~~~~~

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the ``copyright'' line and a pointer to where the full notice is found.

one line to give the library's name and a brief idea of what it does.
Copyright (C) year name of author

> This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
>
> You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a ``copyright disclaimer'' for the

library, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.
>
> signature of Ty Coon, 1 April 1990
> Ty Coon, President of Vice

## Приложение C. Введение в Vi

vi (произносится "vee eye") является единственным редактором, который вы можете найти почти в каждом Unix'е. Первоначально он был написан в Калифорнийском Университете, и его версии можно найти почти в каждом издании Unix, включая Linux. Поначалу, к нему трудно привыкнуть, но у него есть много мощных возможностей. Вообще говоря, мы предполагаем, что новички будут изучать Emacs, с которым работать легче, однако, те, кто собирается использовать большее одной платформы, или те, кому Emacs не нравится, могут попытаться освоить vi.

Для того, чтобы понять, как клавиша k может передвигать курсор на одну строку вверх, и почему существует три различных режима использования, необходим краткий исторический обзор vi. Если вам не терпиться поскорее изучить редактор, две обучающие части помогут вам из новичка стать профессионалом, который знает все команды, которые когда-либо могут потребоваться. Этот раздел также включает в себя руководство для "продвинутых" пользователей.

Даже если vi не станет вашим любимым редактором, приобретенные знания не пропадут даром. Почти наверняка в Unix'е, который вы используете, есть какой-то вариант редактора vi. Вам может

понадобиться vi для инсталляции другого редактора, например, Emacs. Многие утилиты, приложения и игры используют некотоые из команд vi.

### C.1 Краткая историческая справка о Vi

Ранние текстовые редакторы были построчные и обычно использовались на допотопных примитивных терминалах. Типичным редактором, работающим в этом режиме был Ed. Это очень мощный и эффективный редактор, использующий очень мало компьютерных ресурсов, и хорошо работавший с терминалами того времени. vi предлагает пользователю альтернативу, со значительно расширенным множеством команд по сравнению с ed и с визуальным интерфейсом.

vi, как мы теперь знаем, происходит от строчного редактора ex. На самом деле ex выглядит как один из режимов vi. Визуальная компонента ex может быть инициализирована из командной строки с помощью команды vi, или внутри самого ex.

Редактор ex/vi был разработан в Калифорнийском Университете Уильямом Джоем. Исходно он поставлялся как неподдерживаемая утилита, до его официального включения в релиз AT&T System 5 Unix. Он стал даже более популярным, чем большинство современных полноэкранных редакторов.

Благодаря популярности vi, возникло множество его версии и модификаций, которые можно найти в большинстве операционных систем. Цель этой главы не в том, чтобы описать все команды vi или его версии. Во многих версиях поведение редактора изменилось и расширилось, большинство не поддерживают первоначальный набор команд vi.

Если вы хорошо, на практике, знакомы ed, вам не придется долго изучать vi. Даже если у вас нет намерения использовать vi в качестве ежедневного редактора, его vi может пригодиться.

### C.2 Краткий курс работы с Ed

Цель этого курса - помочь вам начать работать с ed. С ed легко работать, и чтобы начать, не требуется большого опыта. Лучший способ изучить что-либо - это попрактиковаться, поэтому следуйте инструкциям и поработайте в редакторе, прежде чем обсуждать о его практические преимущества.

#### C.2.1 Создание файла

ed может редактировать только один файл в данный момент времени. Ниже приведен пример создания текстового файла с помощью ed.

```
/home/larry# edaThis is my first text file using Ed.This is really fun..w
firstone.txt/home/larry# q
```

Вы можете проверить содержимое файла, используя команду cat

```
Unix'a.
/home/larry# cat firstone.txt
```

Вышеприведенный пример иллюстрирует несколько важных моментов.

Когда вы вызывали ed, ваш файл был пуст. Клавиша "a" используется, чтобы добавить в файл текст. Чтобы закончить введение текста, используется точка ".", которая ставиться в первой колонке текста. Чтобы сохранить файл, используется "q" в сочетании с именем файла, а клавиша "q" используется для выхода из редактора.

Самое главное наблюдение - ed работает в двух режимах. В начале редактор находится в командном режиме. Команда определяется символами, поэтому чтобы удостовериться, что именно пользователь имеет ввиду, ed использует текстовый и командный режимы.

C.2.2 Редактирование существующего файла

В следующем примере показано, как добавить строку текста в существующий файл.

/home/larry# ed firstone.txtaThis is a new line of text..wq Если вы посмотрите содержимое файла с помощью команды cat, вы увидите что новая строка была вставлена между старыми первой и второй строками. Как ed узнал, где именно размещать новую строку текста?

При чтении файла ed поддерживает указатель на текущую строку. Команда "a" вставляет текст после текущей строки. ed может поместить текст и перед текущей строкой - с помощью команды i.

Теперь становится очевидным, что ed работает с текстом построчно. Любые команды ed применяются только к текущей строке.

Добавим строку в конец файла.

/home/larry# ed firstone.txt $a The last line of text. . w q Модификатор команды "$" сообщает ed'у о том, что надо добавлять строчку после последней строки. Чтобы добавить строчку после первой строки текста, надо использовать модификатор "1". Таким образом, мы можем добавлять строчки перед номером строки или после нее.

Как узнать, какой текст находится в текущей строке? Команда "p" отобразит содержимое текущей строки. Если вы хотите сделать текущей строкой строку с номером 2 и посмотреть содержимое этой строки, напечатайте следующее:

/home/larry# ed firstone.txt 2p q C.2.3 Подробнее о номерах строк

Вы уже знаете, как выводить содержимое текущей строки с помощью команды "p". Мы также знаем, что существуют модификаторы номеров строк для команд. Напечатаем содержимое второй строки:

2p Существуют несколько специальных модификаторов, которые указывают на позицию, которая может изменяться во время сеанса редактирования. "$" означает последнюю строку текста. Напечатаем последнюю строку.

$p Текущий номер строки использует использует специальный модификатор ".". Выведем текущую строку при помощи модификатора.

.p Это может показаться ненужным, хотя это очень полезно в контексте интервалов номеров строк.

Чтобы напечатать содержимое текста от первой строки до второй, нужно передать ed этот диапазон.

1,2p Первое из чисел указывает на начальную строку, второе указывает на конечную строку. Затем вторым числом в диапазоне команды будет текущая строка.

Выведем содержимое файла от начала файла до текущей строки.

1,.p Выведем содержимое файла от текущей строки до конца файла.

.,$p Осталось только вывести содержимое всего файла, это предоставляется сделать вам.

А вот так можно удалить первые две строки файла.

1,2d Команда "d" удаляет текст построчно. Удалим весь созданный нами
файл.

1,$d Если вы сделали много изменений, но не хотите сохранять ваш файл,
лучше всего выйти из редактора без предварительного сохранения файла.

Большинство пользователей не используют ed как постоянный редактор. В более современных редакторах полноэкранный режим редактирования и более гибкий набор команд. Ed - хорошее введение в vi и он помогает понять происхождение его команд.

C.2 Краткое курс работы с Vi

Цель этого курса - помочь вам начать работать с vi. Мы не предполагаем, что вы обладаете опытом работы с vi. В этом разделе будут описаны десять основных команд vi. Знания этих основных команд достаточно, чтобы спокойно редактировать. Но вы можете расширить ваш "словарь" команд vi, когда это потребуется. Рекомендуем иметь под

рукой компьютер, чтобы выполнять новые команды по ходу знакомства с ними.

C.3.1 Запуск vi

Чтобы запустить vi, просто напишите vi и имя файла, который вы хотите создать. Вы увидите экран с колонкой тильд ("~") слева. Сейчас vi находится в командном режиме. Все, что вы напишите, будет восприниматься, как команда, а не как вводимый текст. Для того, чтобы ввести текст, вам нужно написать команду. Вот две основные команды ввода :

i - вставить текст слева от курсора

a - вставить текст справа от курсора

Так как вы находитесь в начале пустого файла, не имеет значение, какую из двух команд выбрать. Напишите одну из них, а затем напишите следующий текст (стихотворение Августа Де Моргана, вы можете найти его в книге Кернигана и Пайка "Unix - универсальная среда программирования"):

Great fleas have little fleas<Enter> upon their backs to bite 'em,<Enter> And little fleas have lesser fleas<Enter> and so ad infinitum.<Enter> And the great fleas themselves, in turn,<Enter> have greater fleas to go on;<Enter> While these again have greater still,<Enter> and greater still, and so on.<Enter> <Esc>

Обратите внимание, что вы нажимаете клавишу Esc чтобы закончить режим вставки и вернуться в командный режим.

C.3.2 Команды управления курсором

h - передвинуть курсор на одну позицию влево

j - передвинуть курсор на одну позицию вниз

k - передвинуть курсор на одну позицию вверх

l - передвинуть курсор на одну позицию вправо

Эти комады можно повторить, если вы будете держать нажатой клавишу. Попробуйте подвигать курсор по вашему тексту. Если вы сделайте недозволенное действие, например, нажмете "k", когда курсор будет находится в верхней строке, экран мигнет или терминал издаст звуковой сигнал. Не беспокойтесь, это не повредит вашему файлу.

C.3.3 Удаление текста

x - удаляет символ в месте, где находится курсор

dd - удаляет строку

Передвиньте курсор на вторую строчку и позиционируйте его так, чтобы он находился под апострофом 'em. Нажмите клавишу "x", и ' исчезнет. Теперь нажмите "i", чтобы прейти в режим вставки, и напишите

th. Нажмите клавишу Esc, после того, как закончите.

C.3.4 Сохранение файла

:w - сохранить (записать на диск)

:q - выйти из редактора

Убедитесь, что вы находитесь в командном режиме, нажав клавишу Esc. Теперь напишите ":w". При этом все, что вы написали, сохраниться в файле.

"q" - это команда выхода из редактора vi. Если вы хотите сохранить файл и выйти, напишите ":wq". Существует также удобное сокращение для команды ":wq" - "ZZ". Так как вам достаточно часто приходится запустив программу, столкнуться с какой-то проблемой, вызвать программу из редактора, сделать небольшое изменение, затем выйти из редактора и опять запустить программу, то команду "ZZ" вы

будете использовать часто. (На самом деле, "ZZ" не точный синоним для ":wq" - если вы не делали каких-либо изменений в редактируемом файле со времени последнего сохранения, "ZZ" просто выйдет из редактора, тогда как ":wq" (лишний раз) сохранит файл перед выходом.)

Если у вас все перепуталось и вы хотите начать все сначала, можно написать ":q!" (помните о том, что сначала надо нажать клавишу Esc). Если вы опустите "!", vi не позволит вам выйти без сохранения.

C.3.5. Что дальше

Десять команд, которые вы только что изучили, достаточны для работы. Однако вы узнали лишь малую часть команд редактора vi. Существуют команды копирования, перемещения текста из одного места файла в другое, настройки редактора по вашему вкусу, и так далее. В общем, получается около 150 команд.

C.4 Урок для "продвинутых"

Преимущество и сила vi в том, что его можно использовать, зная небольшое количество команд. Большинство пользователей vi чуствуют вначале небольшое неудобство, и, спустя короткий промежуток времени пытаются узнать какими еще командами можно пользоваться.

В этом разделе предполагается, что пользователь изучил краткое описание vi (в предыдущих разделах) и уверенно себя чуствует в vi. Здесь описаны некоторые из наиболее мощных средств ex/vi от копирования текста до макроопределений. Также описан ex и его настройки, которые помогут вам приспособить редактор под собственные нужды. Рекомендуется испытывать команды на примерах.

Эта глава описывает не все команды vi, но наиболее часто используемые и многие другие здесь описаны. Даже если будете использовать альтернативный текстовый редактор, надеюсь, вы оцените vi и его возможности.

C.4.1 Команды управления курсором

Самая важная возможность редактора - это возможность перемещение курсора по тексту. Здесь приведены команды перемещения курсора.

h - передвинуть курсор на один символ влево

j - передвинуть курсор на один символ вниз

k - передвинуть курсор на один символ вверх

l - передвинуть курсор на один символ вправо

Некоторые реализации позволяют передвигать курсор при помощи клавиш-стрелок.

w - передвинуть в начало следующего слова

e - передвинуть в конец следующего слова

E - передвинуть в конец следующего слова перед пробелом

b - передвинуть в начало предыдущего слова

0 - передвинуть в начало строки

^ - передвинуть на первое слово текущей строки

$ - передвинуть в конец строки
<CR> - передвинуть в начало следующей строки
"-" - передвинуть в начало предыдущей строки
G - передвинуть в конец файла
1G - передвинуть в начало файла
nG - передвинуть на строку с номером n
<Cntl> G - вывести номер текущей строки
% - передвинуть на соответствующую скобку
H - передвинуть на верхнюю строку экрана
M - передвинуть на среднюю строку экрана
L - передвинуть на нижнюю строку экрана
n| - передвинуть курсор в колонку n

Экран будет автоматически прокручиваться, когда курсор достигнет верха или низа экрана. Есть альтернативные команды, которые могут управлять прокруткой текста.

<Cntl> f - прокрутить на экран вперед
<Cntl> b - прокрутить на экран назад
<Cntl> d - прокрутить на пол-экрана вниз
<Cntl> u - прокрутить на пол-экрана вверх

Вышеперечисленные команды управляют перемещением курсора. Некоторые из команд используют модификатор, в виде числа перед командой. Обычно команда повторяется заданное число раз.

Передвинем курсор на заданное число позиций влево.

nl - передвигает курсор на n позиций влево.

Если вы хотите ввести несколько пробелов перед текстом, можно использовать модификатор команды для команды вставки. Введите число повторения, "i", пробел, а затем нажмите клавишу ESC.

ni - вставляет текст и повторяет его n раз.

Команды, работающие со строками, используют модификатор, чтобы ссылаться на номер строки. Хорошим примером является "G".

1G - Передвинуть курсор на первую строку.

В vi есть большое количество команд, которые могут быть использованы для перемещения курсора по файлу: от посимвольного передвижения до непосредственного помещения курсора на строку. В vi можно из командной строки помещать курсор в выбранную строку.

vi +10 myfile.tex

Эта команда открывает файл myfile.tex и помещает курсор на 10 строку от начала файла.

Поэкспериментируйте с командами этой главы. Очень немногие могут запомнить все эти команды за один сеанс работы. Большинство пользователей используют только часть этих команд.

Итак, вы можете перемещать курсор по экрану, а как же изменять текст?

### C.4.2 Изменение Текста

Цель этой главы - научиться изменять содержимое файла. В vi есть очень много команд, которые помогут вам в этом.

В этой главе мы изучим, как добавлять текст, изменять существующий текст и удалять текст. По окончанию изучения этой главы вы научитесь создавать произвольный текстовый файл. Оставшиеся части посвящены командам, помогающим чувствовать себя комфортно.

Повторяющиеся строки могут быть введены с помощью клавиши Enter. Если вы допустили опечатку в той же строке, в которой в данный момент набираете текст, то вы можете использовать клавишу backspace для передвижения по строке. В различных версиях vi это реализовано по-разному. В некоторых курсор перемещается назад и текст остается виден. Другие же удаляют текст за курсором. В некоторых версиях используются даже клавиши стрелок для перемещения курсора в режиме

ввода. Это нетипично для vi. Если после backspace текст остался видимым и вы нажимаете клавишу ESC на той строке, где вы использовали клавишу backspace, то текст за курсором будет стерт. Поработайте в редакторе, чтобы привыкнуть к его поведению.

a - Добавить текст, начиная с текущей позиции курсора.

A - Добавить текст к концу строки

i - Вставить текст слева от курсора

I - Вставить текст слева от первого символа текущей строки, не являющегося пробелом.

o - Создать новую строку и добавить текст ниже текущей строки

O - Создать новую строку и добавить текст выше текущей строки

В vi есть несколько команд удаления, которые могут быть расширены при помощи модификаторов команд.

x - Удалить один символ под курсором

dw - Удалить символы, начиная с текущей позиции до конца слова

dd - Удаляет текущую строку.

D - Удалить символы, начиная с текущей позиции до конца строки

Модификаторы могут быть использованы для расширения команд. Следующие примеры - только часть всех возможностей.

nx - Удалить n символов под курсором

ndd - Удалить n строк

dnw - Удалить n слов. (То же, что и ndw)

dG - Удалить с текущей позиции до конца файла

d1G - Удалить с текущей позиции до начала файла

d$ - Удалить с текущей позиции до конца строки (То же, что и D)

dn$ - Удалить от текущей строки до конца n-ой строки

Вышеприведенный список команд показывает, что и команды удаления могут делать очень многое. Это становиться очевидным, если команды удаления сочетать с командами перемещения курсора. Следует обратить внимание на команду "D", так как она игнорирует директивы модификатора.

Может оказаться, что вам нужно будет отменить последнее изменение. Следующие команды восстанавливают текст после изменения.

u - Отменить последнюю команду

U - Отменить все изменееения, сделанные в этой строке

:e! - Редактировать заново. Восстанавливает состояние, полученное после последней записи.

vi не только позволяет вам отменять изменение, он может отменять undo. При помощи команды 5dd удалите 5 строк, а затем восстановите их при помощи "u". Изменения в свою очередь можно восстановить при помощи "u".

В новой версии vi есть команды, которые позволяют изменять текст вместо того, чтобы удалить и перепечатать.

rc - Заменить символ под курсором на "c" (Передвигает курсор направо, если используется модификатор повторения, например, 2rc)

R - Заменить текст новым текстом

cw - Изменить текущее слово.

c$ - Изменить текст, начиная от текущей позиции до конца строки

cnw - Изменить n слов (как и ncw)

cn$ - Изменить до конца n-ой строки

C - Изменить до конца строки (как и c$)

cc - Изменить текущую строку

s - Подставить текст вместо текущего символа

ns - Подставить текст вместо n символов

Последовательность команд замены, позволяющих вводить строку символов символов, заканчивается нажатием клавиши ESC.

Команда "cw" начинает действует с текущей позиции в слове и до конца слова. Когда используется команда замены с указанием области

изменения, vi помещает "$" в позицию последнего символа области. Новый текст может превышать или быть меньше длины исходного текста.

### C.4.3 Копирование и перемещение частей текста

Чтобы переместить текст, необходимо использовать сразу несколько команд. В этой части описаны именованные и безымянные буфера и команды, которые вырезают кусок текста и вставляют текст из буфера.

Процесс копирования текста состоит из трех основных частей.

Копирование текста в буфер.

Перемещение курсора в нужную позицию.

Вставка текста в редактируемый текст.

Чтобы скопировать текст в безымянный буфер используйте команду "y".

yy - Помещает копию текущей строки в безымянный буфер.
Y - Помещает копию текущей строки в безымянный буфер.
nyy - Помещает следующие n строк в безымянный буфер
nY - Помещает следующие n строк в безымянный буфер

yw - Помещает слово в безымянный буфер
ynw - Помещает n слов в безымянный буфер
nyw - Помещает n слов в безымянный буфер
y$ - Помещает текст от текущей позиции до конца строки.

Безымянный буфер - это временный буфер, содержимое которого легко может быть испорчено другими командами. Может оказаться, что текст будет нужен в течение длительного времени. В этом случае используются именнованный буфер. В vi есть 26 именованных буферов. Идентификацинным именем буфера являются буквы алфавита. Чтобы отличить команду от именованного буфера, vi использует символ ". Если что-то записывается в именованный буфер, идентификационное имя которого написано строчными буквами, его старое содержимое затирается, а если заглавными, то новый текст дописывается к старому содержимому.

"ayy - Помещает текущую строку в именованный буфер a.
"aY - Помещает текущую строку в именованный буфер a.
"byw - Помещает текущее слово в именованный буфер b.
"Byw - Дописывает слово к содержимому именованного буфера b.
"by3w - Помещает следующие три слова в именованный буфер b.

Используйте команду "p", чтобы вставить содержимое буфера в редактируемый текст.

p - Вставить содержимое безымянного буфера справа от курсора
P - Вставить содержимое безымянного буфера слева от курсора
nP - Вставить n копий содержимого безымянного буфера слева от курсора
"ap - Вставить содержимое именованного буфера справа от курсора
"b3P - Вставить 3 копии содержимого именованного буфера b слева от курсора

При использовани vi в xterm есть еще одна опция для копирования текста. Выделите текст, который вы хотите скопировать, перемещая курсор мыши по тексту. Нажав левую кнопку мыши и передвигая мышь, вы проинвертируете текст. Текст автоматически будет помещен в служебный буфер X сервера. Чтобы вставить текст из буфера, нажмите среднюю кнопку. Не забудьте установить vi в режим вставки, так как вводимый

текст может быть интерпретирован как команды и результат будет непредсказуем. Используя точно такой же способ можно скопировать одно слово, два раза нажав на левую кнопку мыши, при этом курсор должен находиться на слове. Слово вставляется из буфера также, как описано выше. Содержимое буфера будет изменяться, только если создана новая выделенная область.

Процесс перемещения текста состоит из трех основных частей.

Удаление текста и помещение его в буфер.

Перемещение курсора в нужную позицию.

Вставка текста из именованного или безымянного буфера.

Процесс перемещения текста отличается от процесса копирования только тем, что на первом шаге текст не копируется, а перемещается в буфер. При выполнении команды dd строка удаляется и помещается в безымянный буфер. После этого вы можете вставить содержимое буфера, точно также, как вы это делали при копировании текста.

"add - Удалить строку и поместить в именованный буфер a.

"a4dd - Удалить 4 строки и поместить в именованный буфер a

dw - Удалить слово и поместить его в безымянный буфер

В разделе об изменении текста вы можете найти дополнительные примеры удалении текста.

Если в системе произошел фатальный сбой, содержимое именованных и безымянных буферов теряется, но содержимое редактируемого текста может быть восстановлено (см. Полезные команды).

C.4.4 Поиск и замена текста

В vi есть несколько команд поиска. Можно искать как отдельные символы, так и регулярные выражения.

Две основных команды поиска - это "f" и "t".

fc - Найти вхождение следующего символа c. Двигаться направо,

к следующему.

Fc - Найти вхождение следующего символа c. Двигаться налево, к предыдущему.

tc - Передвинуться направо, к символу перед следующим c.

Tc - Передвинуться налево, к символу за предыдущим c.
(В некоторый версиях это то же, что и Fc)

";" - Повторяет последние f,F,t,T команды

"," - То же, что и ";" но меняет направление исходной команды.

Если искомый символ не найден, vi издаст звуковой или какой-нибудь другой сигнал.

vi позволяет искать строку в редактируемом тексте.

/str - Искать правее и ниже следующее вхождение str.

?str - Искать левее и ниже следующее вхождение str.

n - Повторить последнюю команду / или ?

N - Повторить последнюю команду / или ? в обратном направлении

При использовании команд "/" или "?" очистится нижняя строка экрана. Введите в ней строку поиска, а за ней клавишу ввода.

Строка в команде "/" или "?" может быть регулярным выражением. Регулярное выражение - это описание множества символов. Это описание формируется при помощи текста, в который вставлены специальные символы. Специальными символами в регулярном выражении могут быть . * [] ^ $.

. Соответствует любому одиночному символу, кроме символа перевода строки.

\ Отменяет действие специального символа.

* Соответствует любому (включая 0) числу вхождений предыдущего символа.

[] Один из символов в скобках

^ Следующий символ должен стоять в началом строки.

$ Предыдущие символы должны быть концом строки.

[^] Любой символ, кроме стоящего в скобках за знаком ^

[-] Диапазону символов

Единственный способ привыкнуть к регулярным выражениям - это использовать их. Далее следует несколько примеров.

| | |
|---|---|
| c.pe | Соответствует cope, cape, caper и так далее |
| c\.pe | Соответствует c.pe, c.per и так далее |
| sto*p | Соответствует stp, stop, stoop и так далее |
| car.*n | Соответствует carton, cartoon, carmen и так далее |
| xyz.* | Соответсвует подстроке, начинающейся на xyz и заканчивающейся сиволом конца строки. |
| ^The | Соответствует любой строке, начинающейся на The. |

| | |
|---|---|
| atime$ | Соответствует любой строке, заканчивающейся на atime. |
| ^Only$ | Соответствует строке, в которой единственное слово Only |
| b[aou]rn | Соответствует barn, born, burn. |
| Ver[D-F] | Соответствует VerD, VerE, VerF. |
| Ver[^1-9] | Соответствует подстроке, начинающейся на Ver, за которой следует любой символ, не являющийся числом. |
| the[ir][re] | Соответствует their,therr, there, theie. |
| [A-Za-z][A-Za-z]* | Соответствует любому слову. |

vi использует командный режим ex, чтобы выполнять операции поиска и замены. Все команды, начинающиеся с двоеточия являются запросами режиме ex.

Команды поиска и замены позволяют осуществлять поиск и замену строк в заданном диапазоне с использованием регулярных выражений. У пользователя могут запросить подтверждение перед тем, как выполнить подстановку. А теперь будет полезно освежить в памяти раздел о представлении номеров строк в описании ed.

Общий вид команды:

:<start>,<finish>s/<find>/<replace>/g

| | |
|---|---|
| :1,$s/the/The/g | - Просматривает весь файл целиком и заменяет the на The . |
| :%s/the/The/g % | - означает весь файл целиком. (То же, что и выше). |
| :.,5s/^.*//g | - Удаляет содержимое, начиная с текущей позиции до 5 строки. |
| :%s/the/The/gc | - Заменяет the на The, но просит подтверждение |
| :%s/^..../g | - Удаляет первые четыре символа в каждой строке. |

Команда поиска является очень мощным средством, когда сочетается с использованием регулярных выражениями. Если не указана директива "g", замена выполняется только в первом подходящем месте каждой строки.

Иногда вы можете пожелать использовать строку поиска в строке замены. Вы можете заново написать строку замены, но vi позволяет использовать строку замены, содержащую специальные символы.

:1,5s/help/&ing/g - Заменяет help на helping в первых 5 строках.
:%s/ */&&/g - Увеличивает число пробелов между словами в два раза.

Использование полной строки поиска имеет свои пределы, поэтому vi позволяет использовать выделенные круглые скобки для задания диапазона подстановок. Используя выделенную цифру 1, которая идентифицирует диапазон в определении, можно построить строку замены.

| | |
|---|---|
| :s/^\(.*\):.*/\1/g | - Удалить все после двоеточия включая само двоеточие. |
| :s/\(.*\):\(.*\)/\2:\1/g | - Переставить слова по разные стороны двоеточия. |

Вам не помешает перечитать описание вышеперечисленных возможностей vi еще раз. В vi есть очень мощные команды, которых нет

современных редакторах. Но стоимость этих команд - главный аргумент против vi. Команды могут оказаться трудными для чтения и запоминания. Большинство полезных вещей могут быть немного неудобны в первое время.

Немного времени и практики - и команды vi станут для васестественными, как дыхание.