

Map Linux

(Linux Sea)

«Beach» by Sean MacEntee
is licensed under CC BY 2.0

Мир Linux

Свен Вермален

Copyright © 2009-2013 Sven Vermeulen

Перевод с английского: Галым Керимбеков*

Пост-верстка: Stas Bookman**

Книга "Мир Linux" предлагает мягкое, но не лишённое технических аспектов (с точки зрения конечного пользователя) знакомство с операционной системой на примере дистрибутива Gentoo Linux. Книга не станет описывать историю ядра Linux или дистрибутивов, либо погружать в детали, менее интересные для пользователей.

Онлайн-руководство Gentoo предлагает весьма подробный подход по ряду разделов и поэтому обязательно к чтению любым пользователем, желающим знать обо всех возможностях этой операционной системы. Несмотря на то, что издание "Мир Linux" и онлайн-руководство Gentoo определённо пересекаются между собой, книга ни в коем случае не призвана заменить последнее.

"Мир Linux" попытается сосредоточиться на темах, о которых повседневные пользователи, вероятно, должны знать, чтобы продолжать работать с Gentoo Linux.

Версия, которую вы читаете в настоящее время, является v1.17 и была сгенерирована 18.02.2016 г. Доступны также версии ODT и ePUB.

^{0*}kerimbekov.galym@yandex.ru

^{0**}nodor4mint@yandex.ru

Оглавление

1	Анатомия операционной системы	7
1.1	Ядро	8
1.2	Системные библиотеки	9
1.3	Системные инструменты	10
1.4	Средства разработки	11
1.5	Инструменты, предоставляемые конечному пользователю	12
1.6	Хорошо, что такое этот GNU?	12
1.7	Linux в качестве ядра операционной системы	13
1.8	Операционные системы Linux: дистрибутивы	14
1.9	Что такое дистрибутив?	17
1.10	Что обеспечивает дистрибутив?	18
1.11	Что такое архитектура?	19
1.12	Мифы, окружающие Linux	20
1.12.1	Linux трудно установить	21
1.12.2	Для Linux нет поддержки	21
1.12.3	Linux - свободное программное обеспечение, таким образом, дыры в системе безопасности находятся легко	22
1.12.4	Linux не имеет графику	22
1.12.5	Я не могу запустить свою программу в Linux	22
1.12.6	Linux безопасен	23
1.12.7	Linux слишком фрагментирован, чтобы когда-либо стать более крупным игроком	23
1.12.8	Linux - альтернатива Microsoft Windows	24
1.12.9	Linux — это противовес Microsoft	24
1.12.10	Слабые места	25
1.13	Упражнения	26
2	Как Свободное программное обеспечение влияет на Linux?	27
2.1	Свободное программное обеспечение	27
2.2	Что такое лицензии на программное обеспечение?	28
2.3	Какие существуют лицензии?	29
2.3.1	Public Domain (Общественное достояние)	29
2.3.2	Лицензия MIT и некоторые подобные лицензии BSD	29
2.3.3	GPL	30
2.4	Лицензии, утверждённые OSI	30
2.5	Лицензии, утверждённые FSF	30
2.6	Свободное программное обеспечение не является некоммерческим	31

2.7	Таким образом, Linux свободен?	32
2.8	Модель разработки	34
2.8.1	Многопроектная разработка	34
2.8.2	Прозрачная разработка	34
2.8.3	Циклы быстрых выпусков	35
2.9	Большая документационная база	37
2.10	Жизненный цикл ПО	38
2.11	Открытые стандарты	38
2.12	Стандарт иерархии файловой системы (FHS)	39
2.13	Linux Standard Base	39
2.14	Спецификации Free Desktop	40
2.15	Упражнения	41
3	Роль сообщества	43
3.1	Сообщества	43
3.2	Местные сообщества	44
3.3	Сообщества в сети	45
3.4	Поддержка	47
3.5	Справочники по документации	47
3.6	Интернет и форумы Usenet	48
3.7	Списки рассылок	48
3.8	Чат	49
3.9	Встречи в реальной жизни	49
3.10	Конференции	50
3.10.1	FOSDEM	50
3.10.2	FOSS.IN	50
3.10.3	LinuxTag	51
3.11	Упражнения	52
4	Запуск Linux	53
4.1	Архитектура системы	53
4.2	Учётные записи пользователей	55
4.2.1	Важное примечание	56
4.3	Процессы	56
4.3.1	Примечание	57
4.4	Файлы и файловая структура	57
4.5	Разрешения	58
4.6	Использование командной строки	59
4.7	Ориентирование в системе	60
4.8	Просмотр данных	62
4.9	Анализ вывода команды ls -l	63
4.10	Базовая манипуляция с файлами	64
4.10.1	Копирование файлов и каталогов	64
4.10.2	Перемещение и переименовывание файлов и документов	65
4.10.3	Удаление файлов и каталогов	65
4.10.4	Редактирование текстовых файлов	66
4.11	Просмотр текстовых файлов	67

4.12	Создание ссылок на файлы и каталоги	68
4.13	Автодополнение для файлов и команд	69
4.14	Переключение терминалов	69
4.15	Выход из сессии	70
4.16	Получение помощи	70
4.16.1	Man-страницы	71
4.17	Выключение	71
4.17.1	Информационные страницы	72
4.18	Краткая справка по синтаксису	74
4.19	Документация, предоставляемая пакетом	75
4.20	Онлайн-документация	76
4.21	Упражнения	77
5	Файловая система Linux	79
5.1	Структура	79
5.2	Монтирование файловых систем	81
5.3	Файловые системы	83
5.4	Разделы и диски	88
5.5	Команда 'mount' и файл fstab	90
5.6	Размещение раздела подкачки	92
5.7	Размещение файловых систем Linux	93
5.7.1	Системные размещения	93
5.7.2	Пользовательские размещения	94
5.7.3	Общие размещения	95
5.8	Специальные файловые системы, предоставляемые ядром	95
5.9	Корневая файловая система /	96
5.10	Область произвольных данных /var	97
5.11	Пользовательская область /usr	97
5.12	Домашний каталог /home	97
5.13	Разрешения и атрибуты	98
5.14	Чтение, запись и исполнение	98
5.15	Особые привилегии	102
5.15.1	Изменение привилегий	103
5.15.2	Изменение (смена) владельца	104
5.16	Атрибуты	104
5.16.1	Листинг или изменение атрибутов	104
5.16.2	Элементы POSIX ACL	105
5.16.3	Стандартные расширенные атрибуты	105
5.17	Обнаружение файлов	106
5.17.1	find	107
5.17.2	Регулярные паттерны find	107
5.17.3	Комбинирование паттернов find	108
5.17.4	Работа с результатом	108
5.18	Упражнения	110

Введение

В области настольных графических сред, Linux, предположительно, небольшой игрок (рыночные исследования оценивают долю на рынке приблизительно в 3%). Однако наиболее вероятно, что вы знаете двух или более человек, использующих Linux, некоторые из них даже исключительно. Если принять это во внимание, то либо вы знакомы с предпочтениями ОС у более, чем ста человек, либо данную статистику стоит рассматривать с некоторым сомнением.

Тем не менее, 3% - это всё ещё много (Вы когда-нибудь думали о том, насколько много настольных систем? Я никогда не находил ответа на этот вопрос). И если мы примем во внимание другие рынки (встраиваемые системы, серверы, сетевые устройства и прочее), доля Linux увеличится.

Но тем не менее, множество людей понятия не имеют, что такое Linux или как с ним работать. В этой книге я предлагаю техническое, краткое введение в операционную систему с пользовательской точки зрения. Я не собираюсь погружаться в понятия, преимущества или недостатки Свободного Программного обеспечения (тем не менее, несколько абзацев не станут болезненными) и рассказывать об истории и развитии операционных систем Linux. Для ознакомления с большим числом ресурсов по этим темам, обратитесь к разделу «Дальнейшие ресурсы» в конце этой главы.

Чтобы стать полноценной книгой о Linux как об операционной системе, важно сообщить пользователю об операционных системах в целом. Linux очень модульный и открытый, и это подразумевает, что пользователю виден каждый компонент в системе. Без понимания структуры ОС пользователю было бы трудно осмыслить предназначение каждого модуля. Поэтому, я посвящаю весь раздел операционным системам.

Как только мы дойдём до задач операционной системы, я продолжу рассказ о реальных системах Linux: дистрибутивах.

В завершение, каждая глава в этой книге предложит ряд упражнений, которые можно попытаться решить. Вы не сможете найти в этой книге ответы по каждому вопросу. Предпочтительнее было бы взглянуть на упражнения, как на средство для дальнейшего продвижения и помощь в поиске (и нахождении) больше тем, связанных с Linux. В конце книги приведён список подсказок и/или ответов по вопросам.

Глава 1

Анатомия операционной системы

Операционная система — фактически стек программного обеспечения, каждый элемент, разработанный для определённой цели.

- **Ядро системы:** управляет обменом данными между устройствами и программным обеспечением, системными ресурсами (такие как процессорное время, память, сеть...) и экранирует разработчика от сложности программирования устройства, предоставляя программисту интерфейс для управления аппаратными средствами.
- **Системные библиотеки** содержат программные методы для разработчиков, необходимые для написания программного обеспечения для операционной системы. Они включают в себя методы создания и манипулирования, обработки файлов, сетевого программирования, и т.д. Это - жизненно важная часть операционной системы, так как вы не можете (или не должны) связываться с ядром напрямую: библиотека экранирует системного программиста от сложности программирования ядра.
- **Системные инструменты** собраны с использованием системных библиотек и позволяют администраторам следить за системой: управлять процессами, перемещаться по файловой системе, запускать другие приложения, настраивать сеть...
- **Инструменты разработки** предоставляют средства для сборки нового программного обеспечения в (или для) системе. Несмотря на то, что это необязательная часть операционной системы, мне очень нравится упоминать их, так как с Gentoo их наличие является требованием (почему дело обстоит так, мы узнаем позже). Эти инструменты включают в себя компиляторы (переводят код в машинный), компоновщики (собирают машинный код и объеди-

няют его в рабочий двоичный файл) и средства, значительно упрощающие процесс сборки.

Другие находящиеся в системе библиотеки улучшают опыт написания кода разработчиками, обеспечивая доступ к методам, которые уже написаны другими. Примеры таких библиотек включают в себя графические (для управления окнами) или научные библиотеки. Их наличие не обязательно в каждой системе, но если вы захотите запустить определённый инструмент, это потребует установку соответствующих библиотек. Поверх этих дополнительных библиотек вы обнаружите готовые к установке инструменты конечного пользователя (пакеты офисных программ, мультимедийные утилиты, графические среды. . .).

1.1 Ядро

Как правило, ядро имеет четыре основные обязанности:

- управление устройствами
- управление памятью
- управление процессами
- обработка системных вызовов

Первая из них называется управление устройствами. Компьютерная система имеет несколько соединённых с ней устройств: доступны не только ЦП и память, но также и диски (и дисковые контроллеры), сетевые платы, графические карты, звуковые карты... Так как каждое устройство работает по-разному, ядру необходимо знать, что может делать устройство, как адресовать и управлять каждым из устройств, обеспечивая корректное функционирование в системе. Эта информация хранится в драйвере устройства: без такого драйвера ядро не знает об устройстве и поэтому не сможет управлять им.

Наряду с драйверами устройств, ядро также управляет обменом данными между ними: оно управляет доступом к совместно используемым компонентам так, чтобы все драйверы могли работать как единое целое. Весь обмен должен придерживаться строгих правил, а ядро подтверждать их соблюдение.

Компонент управления памятью управляет использованием памяти системой: оно отслеживает используемую и неиспользованную память, присваивает процессам, требующим её, и гарантирует то, что они не смогут управлять данными

друг друга. Чтобы осуществить это, ядро использует понятие адресов виртуальной памяти: адреса для одного процесса не являются действительными, и ядро отслеживает корректную картографию этих адресов. Также возможно, что в действительности данные отсутствуют в памяти, несмотря на то, что они присутствуют для процесса: такие данные хранятся в области подкачки. Поскольку область подкачки намного медленнее реальной памяти, использование этого пространства должно быть ограничено данными, которые считаются нерегулярно.

Чтобы гарантировать, что каждый процесс получает достаточно процессорного времени, ядро отдаёт приоритеты процессам и даёт каждому из них определённое количество процессорного времени, прежде чем остановит процесс и передаст ЦП следующему. Управление процессами имеет дело не только с делегацией процессорного времени (называется планированием), но также и с полномочиями безопасности, информацией о владельце процесса, обменом данными между процессами и т.д.

Наконец, чтобы фактически работать над системой, ядро должно обеспечиваться средствами, необходимыми для системы и программиста, чтобы контролировать само себя, предоставлять или получать информацию, исходя из которой могут быть приняты новые решения. Используя системные вызовы программист может написать приложения, которые запрашивают ядро для получения информации или просят, чтобы оно выполнило определённую задачу (например, управлять некоторыми аппаратными средствами и возвращать результат). Конечно, такие вызовы должны быть безопасны в использовании, чтобы вредоносные программисты не могли привести систему в нерабочее состояние хорошо обработанным вызовом.

Операционная система, такая как Gentoo Linux, использует Linux в качестве ядра.

1.2 Системные библиотеки

Поскольку ядро само по себе мало на что способно, для выполнения задач оно должно инициироваться триггерами. Такие триггеры создаются приложениями, однако эти приложения конечно должны знать, как устанавливать системные вызовы ядра. Поскольку каждое ядро имеет различный набор системных вызовов (очень специфичных по отношению к системе), программисты создали стандарты с которыми они могут работать. Каждая операционная система поддерживает эти стандарты, которые в дальнейшем переводятся в соответствующие системные вызовы для этой системы.

Стандарт в качестве примера - библиотека Си, вероятно наиболее важная доступная системная библиотека. Эта библиотека делает доступными довольно первостепенные операции для программиста, такие как основная поддержка ввода/вывода, строковые подпрограммы обработки, математические методы, управление памятью и операции с файлами. С помощью этих функций программист может создать программное обеспечение, которое основывается на каждой операционной системе, поддерживающей библиотеку Си. Эти методы далее переводятся библиотекой Си в определённые системные вызовы ядра (если они необходимы). Таким образом, у программиста нет необходимости знать о внутренностях ядра, он даже может написать программу (однократно), которая может быть собрана для множества платформ.

Не существует какой-либо единственной спецификации по тому, какова системная библиотека. Автор этой книги полагает, что системные библиотеки — это любые библиотеки, являющиеся частью стандартной, минимальной инсталляции операционной системы. Также системные библиотеки для одной операционной системы (и даже дистрибутив) могут отличаться от аналогичных библиотек для другой. У большинства дистрибутивов Linux имеются те же системные библиотеки, что неудивительно, так как все они могут выполнять одно то же программное обеспечение, если конечно оно собрано для этих библиотек. Некоторые дистрибутивы просто не отмечают одну библиотеку как часть стандартной, минимальной инсталляции, в отличие от других.

Самая распространённая системная библиотека для систем Linux - GNU Си, также известная как `glibc`.

1.3 Системные инструменты

Так же как и с системными библиотеками, не существует единой спецификации для системных инструментов. Тем не менее, в отличие от системных библиотек они достаточно прозрачны для конечного пользователя. Поэтому почти все дистрибутивы Linux используют те же системные инструменты или подобные инструменты с аналогичными функциями, но реализованные иначе.

Но что такое системные инструменты? Ну, вы все ещё не можете управлять своей системой с помощью ядра и некоторых библиотек программирования. Для этого вам необходим доступ к командам для ввода в интерпретированную и запущенную систему. Эти команды выполняют простейшие задачи, такие как навигация по файлам (переход в каталог, создание/удаление файлов, получение файла листингов...), управление информацией (текстовый поиск, сжатие, перечисление

различий между файлами...), управление процессом (запуск новых процессов, получение листинга процессов, завершение рабочих процессов...), задачи, связанные с полномочиями (изменение владельца файлов, изменение идентификаторов пользователей, обновление разрешений файлов...) и др.

Если вы не знаете, как иметь дело со всем этим материалом, то не имеете навыков работы с системой. Некоторые операционные системы скрывают эти задачи за усложнёнными средствами, другие имеют простые, но отдельные для каждой задачи и объединяют мощь всех этих инструментов. Unix (и Linux) является одним из последних. Системы Linux для большинства этих задач обычно имеют набор системных утилит GNU (GNU Core Utilities).

1.4 Средства разработки

С упомянутыми тремя компонентами у вас есть запущенная, рабочая система. Возможно вы не сможете сделать всё, что захотите, однако можно обновить систему до того состояния, прежде чем она станет выполнять желаемое. Каким образом? Устанавливая дополнительные инструменты и библиотеки до тех пор, пока у вас не будет своей функционирующей системы.

Эти дополнительные инструменты и библиотеки, несомненно, написаны программистами, а это значит, что они должны быть способны собрать свой код так, чтобы он работал в системе. Некоторые системы, такие как Gentoo Linux, даже собирают их вместо того, чтобы полагаться на исходный код, предварительно собранный другими людьми. Чтобы собрать эти инструменты, необходим исходный код каждого из них, а также инструментов, необходимых для преобразования исходного кода в исполняемые файлы.

Называются они «Инструментарий разработки» (tool chain): набор связанных между собой инструментов, необходимых для разработки рабочего приложения. Общий набор инструментов разработки состоит из текстового редактора (для написания кода), компилятора (для преобразования кода в специфичный для машины язык), компоновщика (для объединения машинного кода нескольких исходных текстов - включая предварительно собранные "совместно используемые" (shared) библиотеки в единый исполняемый файл) и библиотек (их я просто упоминал как "совместно используемые").

Этот инструментарий имеет наибольшее значение для разработчика; это — жизненно важное средство разработки, но не единственное. К примеру, разработчики графических приложений обычно нуждаются в инструментах для создания графики, или даже инструментах, связанных с мультимедией, для добавле-

ния звуковых эффектов в их программу. Средство разработки - общее существование для инструмента, необходимого разработчику для создания чего-либо, но несущественного для операционной системы пользователя среднего уровня, который не является разработчиком.

Наиболее известные средства разработки также предоставляются Фондом GNU, а именно, набор компиляторов GNU (GNU Compiler Collection), также известный как gcc.

1.5 Инструменты, предоставляемые конечному пользователю

Как только разработчик закончит создание своего продукта, у вас появится инструмент конечного пользователя с сопроводительными библиотеками (которые могли бы потребоваться другим инструментам, который представляют собой модифицированные сборки на основе этого продукта). Инструменты, делающие систему уникальной для пользователя, представляют собой то, что он хочет сделать с ней. Несмотря на невостребованность самой операционной системой, они необходимы конечному пользователю и поэтому очень важны для его системы.

Большинство операционных систем не устанавливает все инструменты или большинство из них по причине того, что их слишком много чтобы выбрать. Некоторые операционные системы даже не обеспечивают пользователя средствами их установки, но полагаются на изобретательность программиста, на создание им установщика, который интегрирует инструмент в систему. Иные системы поставляют с собой небольшое, но значительное подмножество инструментов конечного пользователя, чтобы их пользователи могли быстро обновить свои системы в любом виде, каком они пожелают, не требуя долгого и изнурительного поиска через Интернет (или ещё хуже, магазин аппаратного или программного обеспечения) программ, в которых они нуждаются.

Примеры инструментов конечного пользователя известны как пакеты офисных программ, инструменты графического дизайна, мультимедийные проигрыватели, коммуникационное программное обеспечение, интернет-браузеры.

1.6 Хорошо, что такое этот GNU?

Проект **GNU** - усилие нескольких программистов и разработчиков по созданию свободной, Unix-подобной операционной системы. GNU является рекурсив-

ным акронимом, означающим, что GNU не Unix, поскольку она подобна ему, но не содержит его кода и (и остаётся) является свободной. Фонд GNU, юридическое лицо, стоящее за проектом GNU, видит свободу как нечто большее, чем просто свободу в финансовом значении: программное обеспечение должно быть свободным для использования в любых целях вообще, для изучения, изменения исходного кода ПО и его поведения, копирования и свободного распространения изменений, внесённых вами.

Данная идея свободного программного обеспечения - благородная мысль, активная в умах многих программистов: следовательно, множество тайтлов программного обеспечения находится в свободном доступе. Программа обычно сопровождается лицензией, которая объясняет, что вы можете и не можете делать с ней (также известное как "Лицензионное соглашение для конечного пользователя EULA). Свободное программное обеспечение также имеет подобную лицензию - в отличие от EULA, вместо того, чтобы запрещать она фактически многое позволяет. Пример такой лицензии - GPL - Универсальная общедоступная лицензия GNU.

1.7 **Linux** в качестве ядра операционной системы

Когда мы смотрим на операционную систему Linux, её основным компонентом является ядро. То самое, которое используют все подобные системы - ядро Linux или просто Linux. Да, правильно, систему Linux называют по имени ядра.

Теперь несмотря на то, что все операционные системы Linux используют одноимённое ядро, многие из них используют различные адаптированные версии. Это вызвано тем, что его разработка имеет несколько ответвлений. Самым важным я назову ванильное ядро. Основное ядро разработки, где продолжает работать большинство его разработчиков; любое другое основывается на нём. Другие ядра предоставляют функции, которые ванильное ядро ещё не имеет (или отказалось в пользу других); тем не менее, они полностью с ним совместимы.

Ядро Linux впервые увидело свет в 1991 году, разрабатывается (и все ещё актуально) Линусом Торвальдсом. Развивалось оно стремительно (версия 1.0.0 уже в 1994 г.) как в размере (версия 1.0.0 имела более чем 175000 строк кода), так и в популярности. За эти годы его модель разработки осталась прежней: существуют немного крупных игроков в разработке, которые решают, что входит и что остаётся вне кода ядра, но большинство вкладов происходит от десятки сотен волонтеров (усвой вклад в ядро версии 2.6.21 несло более чем 800 человек).

Последняя версия ядра на момент написания этой книги - 3.10.7. Первые два

числа играют роль основной версии, третье число - вспомогательная версия (главным образом, обозначающее выпуски с исправлениями ошибок). Иногда добавляется четвёртое число, если необходимо одноразовое исправление ошибки. При разработке ядра Linux обычно постепенно увеличиваются основные числа (большую часть времени второе число), обозначающие выпуски с функциональными улучшениями: с каждым инкрементом пользователи (и разработчики) узнают, что таким образом, ядро получает новые функции.

Как только выпускается новая версия ядра Linux, оно не распространяется всем его пользователям. Нет, это уже та область, где играют роль дистрибутивы...

1.8 Операционные системы Linux: дистрибутивы

Если бы пользователь захотел установить систему Linux без дополнительной справки, ему пришлось бы самостоятельно собрать ядро, компоненты операционной системы (библиотеки, конечные инструменты...) и отслеживать изменения в мире свободного программного обеспечения (такие как выход новых версий или исправлений безопасности). И несмотря на то, что всё это совершенно возможно (обратитесь к проекту Linux From Scratch), большинство пользователей пожелало бы что-нибудь более... удобное для них.

Откройте для себя дистрибутивы. Проект дистрибуции (такой как Проект Gentoo) ответственен за расширение системы Linux (дистрибутива) для конечного пользователя, по сути точка контакта для её установки.

Проекты дистрибуции делают выбор относительно программного обеспечения:

Каким образом пользователи должны установить операционную систему?

Возможно, пользователи поощряются к выполнению как можно больше шагов в процессе установки ("дистрибутив" Linux From Scratch, вероятно имеет самый интенсивный процесс установки). Наиболее противоположным этому является загрузочный CD или USB образ системы, которая даже не требует какой-либо настройки или установки: она просто загружает среду и вы можете начать использовать такой дистрибутив.

Какие имеются варианты установки (CD, DVD, сеть, Интернет...?)

Большинство дистрибутивов Linux предлагает вариант установки с CD/DVD, поскольку это самый популярный способ для получения программного обеспечения. Однако существуют множество других вариантов. Можно установить дистрибутив по сети, используя начальную сетевую загрузку (популярный подход в

корпоративной среде, так как он делает возможной установку без сопровождения), или из другой операционной системы.

Какое программное обеспечение должно быть доступно пользователю)

Популярные настольные дистрибутивы предлагают конечным пользователям широкий диапазон программного обеспечения. Это позволяет дистрибутиву стать общепризнанным, поскольку он соответствует потребностям многих пользователей. Однако существуют более усовершенствованные дистрибутивы, которые сконцентрированы на определённом рынке (абонентские установки для мультимедийных представлений, брандмауэры и управление сетью домашние устройства автоматизации...) и конечно эти дистрибутивы предлагают пользователям различные тайтлы программ.

Каким образом собрано доступное программное обеспечение (определённая система, функции...)?

Если дистрибутиву требуется, чтобы программное обеспечение работало на наибольшем количестве типов процессоров (Pentium, i7, Athlon, Xeon, Itanium...), необходимо собрать программное обеспечение для универсальной платформы (скажем i686), а не для определённой (Itanium). Конечно, это означает, что программа не станет использовать все функции, которые обеспечивают новые процессоры, но в действительности оно все ещё будет работает на множестве систем.

То же самое справедливо и для функций, поддерживаемых определёнными тайтлами программ. Некоторые из них предлагают дополнительную поддержку ipv6, ssl, truetype fonts... Однако, при необходимости, вы должны скомпилировать её в программе. Дистрибутивы, предлагающие программное обеспечение в двоичном формате (этим и занимается большинство из них) должны сделать этот выбор за своих пользователей. Чаще всего, они пытаются предложить поддержку как можно большего количества функций, однако далеко не все конечные пользователи нуждались бы этим или даже желали бы.

Так ли важна интернационализация программного обеспечения?

Некоторые дистрибутивы предназначены для определённых групп пользователей, в зависимости от языка и географии. Существуют полностью локализованные для определённой группы (скажем для "пользователей, говорящих на бельгийско-нидерландском языке"или "канадских франкоговорящих пользователей"), но также и пытающиеся предложить локализацию для наибольшего количества групп.

Как пользователи должны обновлять и обслуживать свою систему?

Множество дистрибутивов предлагают автоматизированный, но не у всех в наличии живой процесс обновления программного обеспечения (например, когда установленная вами инсталляция постепенно растёт и становится последней версией дистрибутива без каких-либо определённых действий). Некоторые из них даже требуют загрузку с последнего установочного CD и выполнение шагов по обновлению.

Каким образом пользователь настраивал бы свою систему?

Если вы — пользователь графического варианта Linux, то определённо не хотите слышать о редактировании конфигурационных файлов или действиях, связанных с командной строки. Таким образом наиболее вероятно, вы станете искать дистрибутивы, предлагающие полный графический интерфейс для настройки системы. Тем не менее некоторым пользователям в самом деле нравится идея редактирования таких файлов напрямую, так как она предлагает наибольшую гибкость (но также и самую высокую кривую обучения) и дистрибутивы чаще всего предлагают эти предпочтения. Некоторые дистрибутивы даже не позволят обновить их напрямую, так как они (пере)создаются без вмешательства, так или иначе перезаписывается всё, что вы изменяли.

Какова целевая группа пользователей дистрибутива?

Большинство настольных дистрибутивов предназначены для домашних/офисных пользователей, но существуют дистрибутивы для детей или учёных. Некоторые созданы для разработчиков, а другие для людей старшего возраста, слабовидящих и не имеющих доступ к Интернету.

Какую политику дистрибутив использует в своём программном обеспечении?

Организации, такие как FSF, имеют собственную философию касательно того, на что мир (программного обеспечения) должен быть похожим. Множество дистрибутивов предлагают способ реализовать это. Например, некоторые из них допускают лишь программное обеспечение, лицензируемое в соответствии с утверждённой FSF лицензией. Другие дистрибутивы позволяют пользователям использовать несвободное. Есть и реализующие более высокую философию безопасности, предлагая наиболее защищённый подход к операционным системам.

Дистрибутив должен быть в свободном доступе?

Конечно, деньги — нередко основная причина принятия решений. Не все дистрибутивы свободно загружаемы/доступны в Интернете, несмотря на то, что большинство из них таково. Но даже когда дистрибутив находится в свободном доступе, всё ещё есть необходимость получить коммерческую поддержку, даже только для обновлений безопасности дистрибутива.

Вы обнаружите несколько дистрибутивов; каждый из этих проектов отвечает на вопросы, которые немного отличаются от других. Следовательно, выбор правильного дистрибутива часто является задачей, в которой вы должны ответить на многие вопросы, прежде чем найти подходящий.

Конечно, когда вы только начинаете работать с Linux, у вас скорее всего ещё нет твёрдого мнения касательно этих вопросов. Это хорошо, так как если вы хотите использовать Linux, то должны начать с дистрибутива, который обеспечивает наилучшую поддержку. Расспросите кого-нибудь рядом, возможно у вас есть друзья, которые могли бы помочь. И будем честны, что может быть лучше персональной поддержки?

1.9 Что такое дистрибутив?

Дистрибутив — это набор программного обеспечения (названного пакетами), объединённого вместе в единый набор, который создаёт полностью функциональную среду. Пакеты содержат тайтлы программного обеспечения (сборка других проектов) и возможно содержат исправления (обновления), специфичные для дистрибутива для лучшей интеграции пакетов или гармоничности со всей средой. Эти пакеты обычно содержат не только копию выпусков, созданных другими проектами программного обеспечения, но и большую логику с целью приспособить программное обеспечение к философии дистрибутива.

Возьмите к примеру KDE. KDE - (графическая) настольная среда, связывающая вместе несколько десятков инструментов меньшего размера. Некоторые дистрибутивы предлагают своим пользователям оригинальную установку KDE, другие немного изменяют её для достижения индивидуального оформления по умолчанию и т.п.

Другим примером был бы MPlayer, мультимедийный проигрыватель, наиболее известный широкой поддержкой различных видеоформатов. Однако, если вы хотите просмотреть видеофайлы Windows Media (WMV), в этом случае потребуется встроенная поддержка (несвободных) win32 кодеков. Некоторые дистрибутивы предоставляют её для MPlayer, другие нет. Gentoo Linux позволяет выбирать, хотите вы её или нет.

1.10 Что обеспечивает дистрибутив?

Если вы хотите использовать дистрибутив, то можете (но не обязаны) использовать инструменты, созданные проектом дистрибутива для упрощения нескольких задач:

- для установки дистрибутива, можно использовать одну или более программ установки, обеспечиваемых проектом
- чтобы установить дополнительные пакеты в системе, можно использовать один или несколько инструментов управления программным обеспечением, обеспечиваемых проектом
- для настройки системы можно использовать один или несколько инструментов настройки, обеспечиваемых проектом.

Я не могу не подчеркнуть достаточную важность термина. Вы не обязаны использовать программу установки дистрибутива (всегда можно установить дистрибутив по-другому), и не обязаны устанавливать программы, используя инструменты управления программным обеспечением (можно собирать и устанавливать вручную), и не обязаны настраивать систему инструментами для настройки (вы можете вручную редактировать конфигурационные файлы различных приложений).

Почему в таком случае дистрибутив перекладывает все эти усилия на инструменты? Причина заключается в том, что они намного упрощают использование системы для пользователя. Возьмите в качестве примера установку программ. Если вы не используете инструмент управления программным обеспечением, то должны собрать программу самостоятельно (что может отличаться в зависимости от программы, которую вы хотите собрать), отслеживать обновления (исправления ошибок и безопасности), удостовериться в том, что установили все зависимое программное обеспечение (программное обеспечение, которое зависит от другого, что, в свою очередь, зависит от библиотеки *a*, *b* и *c*...), и отслеживать установленные файлы, чтобы система не переполнилась хламом.

Другое основное дополнение, которое обеспечивают дистрибутивы — это пакеты программного обеспечения. Пакет содержит программный тайтл (например браузер Mozilla Firefox) с дополнительной информацией (такой как описание программного тайтла, информация о категории, зависимостях и библиотеках...) и логику (как установить программное обеспечение, как активировать определённые модули, которые оно обеспечивает, как создать запись меню в графических

средах, как его собрать, если он ещё не собран...). Это может отразиться на сложности пакета, что является одной из причин того, почему дистрибутивы обычно не могут выпустить новый пакет в день выпуска версии программы.

Однако, большая часть информации и логики для исправлений безопасности остаётся такой же, выпуск исправлений безопасности для программного обеспечения обычно приводит к быстрому выпуску проектом дистрибутива исправлений безопасности к пакету с этим ПО.

Проект дистрибутива обеспечивает следующие элементы поддержки наряду с программным обеспечением из которого состоит дистрибутив:

- документация по дистрибутиву
- инфраструктура, откуда Вы можете загрузить дистрибутив и его документацию
- ежедневные обновления пакетов для нового программного обеспечения
- ежедневные обновления безопасности поддержка по дистрибутиву (которая может быть в виде форумов, почтовой, телефонной или даже коммерческой договорной),
- ...

Отныне проект дистрибутива — нечто большее, чем всё это. Разработчики могут сотрудничать, объединяя все пакеты в единый проект, чтобы создать систему, которая расширяет ряд операционных систем "коммерческого сорта". Для достижения этой цели большинство проектов дистрибутива имеет подразделения для связей с общественностью, пользовательских отношений, отношений с разработчиками, управления версиями, документации и переводов и т.д.

1.11 Что такое архитектура?

Я ещё не упоминал об архитектурах, тем не менее они важны. Позвольте мне для начала определить понятие набора инструкций.

Набор инструкций ЦП - это набор команд, который понимает определённый процессор. Эти команды выполняют множество функций, таких как арифметические функции, операции памяти и управление потоком. Программы могут быть написаны с их использованием, но обычно программисты применяют высокоуровневый язык программирования, поскольку программа, написанная на этом языке (названном ассемблером упомянутого ЦП), может запускаться только на

этом ЦП. Таким образом, ассемблер настолько низкоуровневый, что написать инструмент с помощью него не совсем просто. Инструменты, использующие ассемблер — это компиляторы (которые переводят высокоуровневый язык программирования в ассемблер), загрузчики (которые загружают операционную систему в память), и некоторые базовые компоненты операционных систем (ядро Linux имеет некоторый ассемблерный код).

Теперь у каждого типа ЦП имеется различный набор инструкций. Intel IV Pentium имеет набор инструкций, отличающийся от набора Intel Pentium III; Sun UltraSPARC III имеет набор инструкций, отличающийся от набора Sun UltraSPARC III. Однако они очень схожи. Это вызвано тем, что они находятся в том же семействе. Центральные процессоры того же семейства понимают определённый набор инструкций. Программные инструменты, созданные для одной системы команд, работают на всех центральных процессорах того же семейства, но не могут использовать в своих интересах весь набор инструкций ЦП, на котором они работают.

Семейства центральных процессоров сгруппированы по архитектурам. Архитектура - глобальная переменная и представляет понятие всей системы; она описывает, как получает доступ к дискам, как обрабатывается память, как определяется процесс начальной загрузки. Она определяет большие, концептуальные различия между системой. Например, диапазон систем, совместимый с Intel, сгруппирован в архитектуре x86; если вы загружаете такую систему, её процесс начальной загрузки запускается с BIOS (Базовая система ввода-вывода). Системы, совместимые со Sparc Sun сгруппированы в архитектуре sparc; если Вы загружаете такую систему, процесс начальной загрузки запускается с PROM.

Архитектура важна, так как дистрибутивы Linux зачастую поддерживают несколько архитектур, и вы определённо должны знать, какую архитектуру ваша система использует. Это по всей вероятности x86 или amd64 (оба довольно эквивалентны), однако необходимо понять, что существует также и другая архитектура. Вы найдёте инструменты, которые не поддерживаются для вашей архитектуры даже при том, что они доступны для вашего дистрибутива, или некоторые пакеты могут иметь последнюю версию в наличии на одной архитектуре, но не на других.

1.12 Мифы, окружающие Linux

Linux часто расхваливается в СМИ - иногда этому есть объяснение, однако большую часть времени его нет. И хотя я ранее обсуждал, чем является Linux, напомню вкратце:

Linux - общее обозначение, относящееся к операционной системе Linux, набору инструментов, работающих под управлением ядра Linux и большую часть времени предлагаемых через проект дистрибутива.

Конечно, зачастую это не вносит ясность для пользователей, незнакомых с миром вне Microsoft Windows. Несмотря на то, что лучший способ узнать, что такое Linux, это использовать его, мне кажется важным разоблачить некоторые мифы, прежде чем продолжить остальную часть книги.

Мифы

Миф — история, которая популярна, но не верна. Мифы, окружающие Linux будут существовать всегда. Следующие несколько разделов попытаются предложить мои идеи касательно развенчанию большинства из них...

1.12.1 Linux трудно установить

Кто-то всегда может указать на дистрибутив, который трудно установить. "Дистрибутив"Linux From Scratch - фактически документ, объясняющий весь процесс установки дистрибутива Linux путём сборки компиляторов, программного обеспечения, размещением файлов, и т.д. Да, это тяжело и даже могло быть сложным, если бы документация не была актуальной.

Тем не менее, множество дистрибутивов (даже большинство из них) просты в установке. Они предлагают тот же подход к установке, как и другие операционные системы (включая Microsoft Windows), вместе с онлайн-справкой (экранная справка) и оффлайн-справкой (руководство по установке). Некоторые дистрибутивы могут даже быть установлены всего двумя или тремя вопросами, вы можете даже использовать Linux без необходимости устанавливать его вообще.

1.12.2 Для Linux нет поддержки

Были дни, когда для Linux не существовало коммерческой поддержки, но это было в прошлом веке. Теперь Вы можете получить операционную систему Linux от крупных поставщиков программного обеспечения, таких как Novell или RedHat (с поддержкой), или использовать свободно загружаемый дистрибутив Linux и получить контракт с компанией, которая предлагает поддержку этого дистрибутива.

Все дистрибутивы также предлагают превосходную бесплатную поддержку (это то, о чем я расскажу в следующих нескольких главах) и у многих из них в наличии активная последующая обработка и анализ безопасности, приводящие к быстрым исправлениям, как только уязвимость находят или сообщают о ней. Часто нет никакой потребности в получении коммерческой поддержки для пользователя настольных систем, поскольку каналы поддержки в свободном доступе предлагают главное преимущество по сравнению с некоторыми другими собственническими операционными системами.

1.12.3 Linux - свободное программное обеспечение, таким образом, дыры в системе безопасности находятся легко

Так как это свободное программное обеспечение, «дырам» в системе безопасности намного труднее остаться в исходном коде. Существует слишком много глаз, наблюдающих за исходным кодом, а у многих проектов свободного программного обеспечения имеется очень активное сообщество разработчиков, которое проверяет и перепроверяет изменения исходного кода множество раз, прежде чем они будут предложены конечному пользователю.

1.12.4 Linux не имеет графику

Ядро Linux не является графическим ядром, однако инструменты, функционирующие ниже ядра, могут быть графическими. Даже больше, большинство дистрибутивов предлагает полный графический интерфейс для каждого возможного аспекта операционной системы: она загружается с графикой, вы работаете графическим методом, устанавливаете программное обеспечение графическим образом и даже диагностируете проблемы, используя графический подход. Несмотря на то, что вы можете работать исключительно с командной строкой, большинство дистрибутивов сконцентрировано на графической среде.

Эта книга не является хорошим примером относительно этого мифа, поскольку она фокусируется на командной строке. Однако это лишь из-за личных предпочтений автора.

1.12.5 Я не могу запустить свою программу в Linux

Для многих тайтлов Microsoft Windows это действительно так. Но почти наверняка существует программное обеспечение, доступное в Linux и предлагающее те же функции, что и то, к которому вы обращаетесь. Некоторые программы даже

доступны для Linux: популярные браузеры Firefox и Chrome - два примера, пакет офисных программ в свободном доступе, OpenOffice.org — ещё один пример.

Существуют также эмуляторы и библиотеки, которые предлагают интерфейс, позволяющий приложениям Microsoft Windows работать в Linux. Тем не менее, я не рекомендую использовать данное программное обеспечение для каждого возможного тайтла. Это самое последнее средство в том случае, когда вам определённо требуется конкретный тайтл, но при этом вы уже выполняете большинство работ в Linux.

1.12.6 Linux безопасен

Это также миф. Linux не более безопасен, чем Microsoft Windows или Apple Mac OS X. Безопасность — это больше, чем сумма всех уязвимостей в программном обеспечении. Она основана на компетентности пользователя, администратора, конфигурации системы и др.

Linux может быть очень безопасным: существуют дистрибутивы, фокусирующиеся на интенсивной безопасности посредством дополнительных параметров настройки, конфигураций ядра, выбора программного обеспечения и прочего. Но вы не нуждаетесь в таком дистрибутиве, если хотите иметь безопасную систему Linux. Предпочтительнее всего прочесть документацию по безопасности дистрибутива, и удостовериться в том, что вы регулярно обновляете свою систему, не запускаете программное обеспечение, в котором не нуждаетесь или не посещаете сайты, законность которых вам неизвестна.

1.12.7 Linux слишком фрагментирован, чтобы когда-либо стать более крупным игроком

Множество групп именуют Linux как фрагментируемый по причине множества дистрибутивов. Однако пользователь одного дистрибутива может легко работать с пользователями других дистрибутивов (здесь нет никакой проблемы). Пользователь одного дистрибутива может также помочь пользователям других дистрибутивов, так как их программное обеспечение - все ещё то же самое (здесь также нет никакой проблемы). Даже больше, программное обеспечение, создаваемое на одном дистрибутиве, прекрасно работает на другом дистрибутиве (здесь тоже нет никакой проблемы). Широко распространённая доступность дистрибутивов - это сила, а не слабость, поскольку она предлагает больше выбора (а также больше знаний и опыта) конечному пользователю.

Возможно, люди ссылаются на различные существующие деревья ядра Linux. Тем не менее, все эти деревья основаны на том же ведущем ядре (часто называемом как "ванильное") и каждый раз, когда ведущее ядро производит новую версию, эти деревья обновляют свой собственный код, поэтому ответвления никогда не отстают. Дополнительные деревья, которые существуют из-за целей разработки (дополнительные патчи для неподдерживаемых аппаратных средств, прежде чем они будут объединены с ведущим ядром, дополнительными патчами для определённых решений для виртуализации, которые в противном случае станут несовместимыми или не могут быть объединены из-за проблем лицензии, дополнительные патчи, которые слишком навязчивы и требуют времени, прежде чем они будут стабилизированы, и т.д.).

Возможно, люди ссылаются на различные графические среды (такие как KDE и GNOME). Тем не менее, они умалчивают о функциональной совместимости между этими графическими средами (вы можете запустить приложения KDE в GNOME и наоборот), о стандартах, которые это разнообразие создают (стандарты работающие с форматами файлов, записями меню, связывании объектов и прочее), и прочее.

Управляемая фрагментация - это то, что предлагает Linux (и свободное программное обеспечение в целом). Управляемая, так как соответствует открытым стандартам и свободным спецификациям, которые хорошо документированы и которых придерживается все программы. Фрагментированная, так как сообщество хочет предложить конечным пользователям больше выбора.

1.12.8 Linux - альтернатива Microsoft Windows

Linux - не альтернатива, это другая операционная система. Существует различие между значениями. Альтернативы пытаются предложить ту же функциональность и интерфейс, используя различные средства. Linux - иная операционная система, так как не стремится предложить ту же функциональность или интерфейс Microsoft Windows.

1.12.9 Linux — это противовес Microsoft

Это не так, поскольку люди, у которых есть определённые чувства к Microsoft, часто используют и Linux. Linux пытается быть, ни чем иным, как системой, полностью взаимодействующей с любой другой. Проекты программного обеспечения совершенно определённо желают, чтобы их программное обеспечение работало на любой операционной системе, не только на Microsoft Windows или Linux.

1.12.10 Слабые места

И все же не вся информация распространяется вокруг мифов. Некоторые из них - реальные слабые места, над которыми в Linux все ещё нужно продолжать работать.

Поддержка игр прогрессирует медленно

Это правда. Несмотря на то, что вокруг существует множество игр среди свободного программного обеспечения, большинство из них разработано исключительно для Microsoft Windows, и не все игры могут быть запущены в Linux с использованием эмуляторов или библиотек, таких как WINE (но к счастью, множество). Не так уж просто попросить разработчиков игр разработать их для Linux, поскольку большинство из них концентрирует свои усилия на библиотеках (такие как DirectX), доступных только для Microsoft Windows.

Тем не менее, в этой области недавно произошли улучшения. Valve выпустила Steam для Linux, обеспечивая игровой процесс в настольном Linux. Это дало большой сдвиг "играм на Linux".

Однако также появляется и другая тенденция: все больше игр выпускается только для консолей, отбрасывая среду ПК. Я лично не знаю, как игры разовьются в будущем, но думаю, что реальные экшн-игры будут фокусироваться больше на игровые приставки.

Новейшие аппаратные средства принимаются в Linux не сразу

Если поставщик аппаратных средств не предлагает драйверы для Linux, то действительно потребуется некоторое время, прежде чем поддержка аппаратных средств будет обеспечена в ядре Linux. Однако это процесс, охватывающий не многие годы, а скорее месяцы. Возможность состоит в том, что совершенно новая видео- или звуковая карта будет поддерживаться уже в течение 3 - 6 месяцев после выпуска.

То же является истиной и для карт беспроводной сети. Принимая во внимание то, что ранее это было слабостью, поддержка карт беспроводной сети отныне хорошо скоординирована в сообществе. Основная причина этого состоит в том, что большинство поставщиков теперь официально поддерживает свой беспроводной чипсет для Linux, предлагая драйверы и документацию.

Упражнения

1. Создайте список дистрибутивов Linux, о которых Вы слышали, и проверьте каждый из них, как они выполняют свои задачи в сферах, которые Вы находите важными (например, доступность документации, переводов, поддержки определённых аппаратных средств, мультимедиа...).
2. Перечислите 7 архитектур ЦП.
3. Почему новые выпуски ядра не распространяются конечному пользователю незамедлительно? Какую роль дистрибутивы играют в этом процессе?

Дальнейшие ресурсы

- + Почему Открытый исходный код / Свободное программное обеспечение, Дэвид А. Уилер - статья об использовании Open Source Software / Free Software (OSS/FS).
- + Distrowatch, популярный сайт, который пытается отследить все доступные дистрибутивы Linux и имеет еженедельное освещение в новостях.

Глава 2

Как Свободное программное обеспечение влияет на Linux?

Введение

ОС Linux стала все более и более популярной главным образом из-за свободы, которую она позволяет (и конечно же низкая цена или стоимость нулевого сбора всей операционной системы). В этой главе мы увидим, как эти свободы претворяются в жизнь, как они защищены и поддерживаются.

Мы также взглянем на модель разработки, используемую проектами свободного программного обеспечения, так как это - главный результат упомянутых свобод, того, что часто делает такие проекты более интересными, чем коммерческие проекты программного обеспечения с закрытым исходным кодом. Модель разработки - также одно из главных сильных мест свободного программного обеспечения.

2.1 Свободное программное обеспечение

Если мы отделимся от всех технических аспектов, то заметим, что Linux отличается от коммерческого программного обеспечения с закрытым исходным кодом в одном важном аспекте: лицензирование. Лицензирование — это то, что управляет свободным программным обеспечением...

2.2 Что такое лицензии на программное обеспечение?

Программное обеспечение — чья-то интеллектуальная собственность. Интеллектуальная собственность — тяжёлое слово, которое не должно быть интерпретировано ни к чему иному, кроме как к результату некоторого усилия по созданию того, что не является простой копией. Если вы пишете что-то, получившийся текст — это ваша интеллектуальная собственность (если вы не скопировали его где-нибудь).

Интеллектуальная собственность защищена законом. Авторское право защищает интеллектуальную собственность, запрещая другим копировать, адаптировать, воспроизводить и/или перераспространять вашу "вещь" без вашего согласия. Тем не менее, имейте в виду, что не каждая интеллектуальная собственность защищена авторским правом, а оно отличается от страны к стране. Примером интеллектуальной собственности, не защищённой авторским правом, является математический метод: даже при том, что изобретатель метода должен был потратить многие годы на его обдумывание, его метод не защищён (но если он написал текст об этом методе, то сам текст). Авторское право присваивается автоматически: это ничего не стоит Вам и принято в широком аспекте.

Другая защита — патент. Патенты предоставляются (или должны) новым изобретениям, не известным общественности во время запроса патента. Они часто используются для защиты интеллектуальной собственности, которая не защищена авторским правом: методов создания материала (в том числе медицинского содержания). К сожалению, отрасль часто злоупотребляет патентами намного больше, когда у них есть патент с широким спектром действия: он покрывает слишком многое, позволяя компании вынудить других не использовать метод, который они фактически имеют право использовать. Кроме того, запрос и получение патента очень дорогостоящие, и только более крупные компании имеют возможность получить (и защитить) несколько патентов. У меньших компаний или людей нет средств получить патент, не говоря уже о собственной защите в суде лишь потому, что они, возможно, использовали метод, который описан в одном или более патентах.

Я использую слово «злоупотребление», так как компании часто получают патенты для методов, которые широко используются или настолько несерьёзны, что вы задались бы вопросом, что за патентное бюро предоставило те патенты (запросы патентов проверяются или должны быть проверены на свою законность, прежде чем их предоставят).

Я воздержусь от развития этой (политически неоднозначной) темы в дальнейшем и перейду к лицензиям на программное обеспечение. Лицензия - это кон-

тракт между Вами - пользователем и владельцем авторского права на программное обеспечение. Она указывает, что вы можете и не можете делать. Любое нелицензируемое программное обеспечение, полностью защищено авторским правом, что означает, что вы не можете иметь его, не говоря уже о запуске.

Большинство лицензий товарного типа часто называются EULA или лицензионными соглашениями конечного пользователя. Они обычно указывают, что вам разрешается делать с программным обеспечением (часто включая то, для чего вам разрешают его использовать). EULA чаще подчёркивают то, что запрещено, чем то, что позволено. Одна из многих тем - перераспространение программного обеспечения. Большинство EULA явно отвергает перераспространение.

Linux (и свободное программное обеспечение в целом) отличается. Сопроводительная лицензия предоставляет право копировать программное обеспечение, получать исходный код, изменять и перераспространять (с или без модификации) и даже продавать его. Поскольку существует множество вариантов, то возможно имеется и множество популярных лицензий.

2.3 Какие существуют лицензии?

Я перечислю несколько наиболее популярных лицензий, но уточню, что вокруг существует более 800 лицензий. Множество из тех лицензий довольно схожи (или точно такие же), и сообщество свободного программного обеспечения должно начать консолидировать все эти лицензии в намного меньший набор. К сожалению, они этого пока так и не сделали. К счастью, здесь применяются правила 90-10: 90% всего свободного программного обеспечения используют 10% свободного программного обеспечения (или другой) лицензии. Другие лицензии используются лишь незначительно, иногда только для единственного приложения.

2.3.1 Public Domain (Общественное достояние)

Когда программное обеспечение размещено под общественным достоянием, Вы свободны распоряжаться им вне зависимости от предпочтений: автор размещает любым правом, которым он может для обеспечения полной свободы его программы.

2.3.2 Лицензия MIT и некоторые подобные лицензии BSD

Лицензия MIT и некоторые подобные лицензии BSD почти похожи на Public Domain, однако просят сохранить уведомление об авторском праве без изменений. Это очень популярная лицензия, поскольку автор позволяет распоряжаться

им вне зависимости от предпочтений, пока его имя сохраняется в примечаниях продукта об авторском праве.

2.3.3 GPL

Общественная Лицензия GNU - наиболее широко используемая лицензия свободного программного обеспечения, но для некоторых людей также самая строгая лицензия свободного программного обеспечения. GPL указывает, что можно делать с программным обеспечением, пока обеспечивается исходный код модификаций каждому, кому распространена изменённая версия и также пока эта модификация находится под GPL.

Ядро Linux — лицензировано GPL.

2.4 Лицензии, утверждённые OSI

Лицензия, утверждённая OSI - это лицензия, придерживающаяся определения Открытый исходный код, написанного по инициативе открытого исходного кода, следующие моменты которой являются свободной интерпретацией:

- свободное перераспространение
- доступный исходный код
- модификации позволяет (включая перераспространение)
- никакой дискриминации (люди, сферы...)

2.5 Лицензии, утверждённые FSF

Лицензия, утверждённая FSF придерживается определения Свободное программное обеспечение, написанного Фондом свободного программного обеспечения, следующие моменты которого являются ядром определения:

Вы должны быть свободны...:

- выполнять программу в любых целях
- изучать, как программа работает и адаптировать её по своим потребностям
- перераспространять копии
- улучшать программу и выпускать свои изменения в общественность

2.6 Свободное программное обеспечение не является некоммерческим

Свободное программное обеспечение часто воспринимается как проект человека, увлечённого своим хобби в чистом виде: оно не было бы коммерчески жизнеспособным, чтобы привнести свободное программное обеспечение в мир предприятий (enterprise). В конце концов, если оно находится в свободном доступе, какую прибыль компания могла бы получать. Ничто не могло быть таким далёким от истины. . .

Это правда, что свободное программное обеспечение требует иного взгляда на программное обеспечение в коммерческой среде (включая компании). Компании, которые используют программное обеспечение, хотят быть уверенными в том, что у них имеется поддержка, когда что-либо идёт не так, как должно. Они часто закрывают (дорогостоящие) контракты на поддержку с компанией-разработчиком, где определены соглашения об уровне обслуживания (сокращённо SLA). На основе этих контрактов у компании имеется страхование, таким образом, если определённые услуги станут недоступными, поддерживающая компания предпримет всё что может, чтобы вернуть услугу, или в некоторых случаях, компенсировать финансовый ущерб, который нанесло банкротство.

Большую часть времени эти контракты на поддержку закрыты самой компанией-разработчиком, поскольку у неё имеется большая часть знаний касающихся программного обеспечения (поскольку, это вероятно, единственная компания с доступом к программному коду). К сожалению, по столь хорошей причине, как эта, компании не смотрят на свободное программное обеспечение, "поскольку здесь нет никакой поддержки". Что не является истиной; поддержка свободного программного обеспечения все ещё (коммерчески) доступна, но большую часть времени не от самих создателей. И несмотря на то, что это пугает компании, причина того, почему эта поддержка все ещё так же хороша как с массовым ПО, остаётся той же: поддерживающая компания имеет доступ к исходному коду инструмента и профессиональные знания об инструменте. У неё, вероятно, имеются разработчики в самом проекте программного обеспечения.

Компании, занимающиеся его продажей, конечно, часто против свободного программного обеспечения. Когда крупный доход этих компаний зависит от продаж их программного обеспечения, оно не было бы жизнеспособным, чтобы стать свободным. И если бы оно было, конкурирующие компании имели бы полный доступ к исходному коду и улучшали бы их собственный продукт с помощью него.

Тем не менее я не думаю, что это недостаток. Компании-разработчики про-

граммного обеспечения должны использовать свою основную силу: знания об инструменте. Как упомянуто прежде, другие компании часто хотят закрыть контракты на поддержку, чтобы гарантировать услугу, которую предоставляет программное обеспечение; если бы компания-разработчик создала свободное программное обеспечение, это не изменилось бы. Для многих компаний-разработчиков контракты на поддержку — основной источник дохода.

Все ещё возможно продать свободное программное обеспечение; некоторые новаторские компании оплачивают модификации, так как не имеют ресурсов, чтобы делать это самим. Эти компании могут сохранять модификации частными, если лицензия свободного программного обеспечения позволяет это), но могут также представить эти модификации общественности, внося их в сам проект программного обеспечения.

Главное доказательство этого - принятие свободного программного обеспечения основными игроками, такими как Sun Microsystems и IBM, появление новых игроков, которые создают свой бизнес на нем, такие как RedHat или MySQL (недавно приобретённый Sun Microsystems). Последняя компания использует подход двойного лицензирования: исходный код MySQL доступен в двух лицензиях, одно в качестве свободного программного обеспечения для общественности и другое более закрытое, для компаний, которым необходима поддержка со стороны самого MySQL. Использование подхода двойного лицензирования позволяет компании поддерживать фиксированное состояние их продукта при сохранении программного обеспечения в свободном виде. Поддержка фиксированного состояния продукта, конечно, намного проще, чем поддерживать программное обеспечение в целом.

Однако, не стоит думать, что каждый проект свободного программного обеспечения готов к предпрятию, или, что Вы сможете найти (оплачиваемую) поддержку каждого такого проекта. Вы должны тщательно проверить каждый тайтл, если хотите использовать программное обеспечение, свободное или нет. Дистрибутивы для конечных пользователей помогают выбирать. Если дистрибутив упаковывает определённый тайтл, это означает, что он стабилен и хорошо поддерживается.

2.7 Таким образом, Linux свободен?

Да, Linux свободен. Он, конечно, свободен в смысле "свободы слова" и несмотря на то, что большинство тайтлов также бесплатно в смысле "бесплатного пива", Вы не должны удивляться, если увидите дистрибутивы, за которые можете

или должны заплатить. В этом случае, вы можете платить за носитель программного обеспечения (записанный DVD), сопровождающую распечатанную документацию, 30-дневную установку и поддержку по использованию или для ресурсов, которые дистрибутив должен получить сам (такие как инфраструктура).

Большинство дистрибутивов имеет возможность бесплатных загрузок с онлайн-документацией и замечательной общественной поддержкой (активные списки рассылки или интернет-форумы), что и является причиной такой популярности Linux: Вы можете загрузить, установить и использовать несколько дистрибутивов, чтобы решить, какой является лучшим. Вы можете попробовать программное обеспечение (без потери функциональности), и не обязаны даже платить за него, чтобы продолжать использовать (как это имеет место с условно-бесплатным программным обеспечением). Gentoo - один из таких проектов дистрибутива. Такие дистрибутивы получают свою финансовую поддержку (для инфраструктуры и организационных потребностей, включая юридическую поддержку и бюрократические документы) от пользовательских пожертвований или продаж отпечатанных DVD. Компании также склонны поддерживать дистрибутивы в финансовом отношении или аппаратными средствами/пропускной способностью.

Некоторые дистрибутивы становятся доступны, только когда вы платите за них. В этом случае, вы чаще всего платите за поддержку или дополнительное программное обеспечение в дистрибутиве, которое не находится в свободном доступе. Популярный дистрибутив - RedHat Enterprise Linux, дистрибутив Linux, в частности, предназначенный для компаний, которые необходимо установить серверы Linux. Вы платите не только за поддержку, но также и за ресурсы, которые RedHat вложил в дистрибутив, чтобы сертифицировать его для другого программного обеспечения (такого как Oracle и SAP) таким образом, чтобы вы могли выполнять (с поддержкой со стороны компании-разработчика) данное программное обеспечение на своих инсталляциях RHEL.

Однако, важно понять, что проекты дистрибутива разрабатывают только очень небольшую часть программного обеспечения, которое вы устанавливаете в своей системе. Оно в большинстве прибывает из других проектов свободного программного обеспечения, и эти проекты часто не получают вознаграждений от проектов дистрибутива. Тем не менее они в действительности сталкиваются с теми же проблемами, как любой другой (большой) проект: бюрократические документы, юридическая поддержка, потребности инфраструктуры... Таким образом, нет ничего удивительного в том, что у этих проектов имеются те же потоки дохода, как и у проектов дистрибутива: пользовательские вознаграждения, коммерческое спонсорство и продажи программного обеспечения/поддержки.

2.8 Модель разработки

Ввиду особенности проектов свободного программного обеспечения, вы обнаружите, что у неё имеются некоторые различия с коммерческой рекламой закрытого исходного кода от готового программного обеспечения. . .

2.8.1 Многопроектная разработка

Один дистрибутив обеспечивает агрегацию программного обеспечения. Каждый из этих тайтлов создан проектом программного обеспечения, который обычно отличается от проекта дистрибутива. Следовательно, когда вы устанавливаете дистрибутив на свою систему, он содержит программное обеспечение от сотен проектов во всем мире.

Таким образом, чтобы получить поддержку по дефекту, который нашли, или проблеме, с которой вы сталкиваетесь, первым местом для поиска поддержки стал бы дистрибутив, однако, вероятность состоит в том, что дистрибутив поместит вопрос о поддержке в апстрим. Это означает, что он передаст запрос в проект, разрабатывающий программное обеспечение, с которым у Вас имеется проблема.

2.8.2 Прозрачная разработка

Свободное программное обеспечение как правило разрабатывается прозрачно: если вы заинтересованы в разработке вашего любимого программного тайтла, то можете быстро выяснить, что она себя представляет и как принять в ней участие.

Традиционно, для хранения исходного кода программные проекты используют систему одновременных версий, такую как CVS или SVN. Подобные системы позволяют десяткам (или даже тысячам) разработчиков одновременно работать над одним и тем же исходным кодом и отслеживать все произведённые изменения (таким образом, они могут быть обращены). Это касается не только проектов свободного программного обеспечения — большинство из них использует такую систему. Тем не менее, проекты свободного программного обеспечения как правило позволяют людям, не являющимся разработчиками следить за прогрессом в разработке, предоставляя им доступ к системе в режиме «только для чтения». Это позволяет персонально отслеживать каждое изменение в программе.

Большинство проектов программного обеспечения не могут использовать встречи в реальной жизни для обсуждения будущего программы или принятия решений по ее дизайну: их разработчики разбросаны по всему миру. Решением этой

проблемы являются коммуникационные системы, такие как списки рассылок, IRC (чат) или форумы (Интернет или UseNet). Большинство этих коммуникационных систем также открыты для обычных людей для участия в обсуждениях, что в свою очередь означает то, что конечные пользователи могут напрямую общаться с разработчиками.

Последнее имеет важное преимущество: изменения, запрошенные пользователями, напрямую направляются разработчикам, что снижает частоту неверных толкований, позволяя проектам обновлять их программу наиболее аккуратно и чаще.

2.8.3 Циклы быстрых выпусков

Большие проекты программного обеспечения имеют тысячи контрибьюторов и несколько десятков разработчиков. Разработчики значительно мотивируются энтузиазмом на разработку программы. В противном случае они не станут работать над программой по причине отсутствия прочих мотиваций (например зарплаты, выдаваемой чеком), тем не менее, это должно быть сказано также про проекты (их не так уж и мало), которые платят разработчикам. В результате, программа быстро прогрессирует и обзаводится новыми возможностями (некоторые проекты даже имеют ежедневные выпуски, содержащие новые возможности).

Чтобы удостовериться в корректности тестирования новых возможностей и исправлений, моментальные снимки разрабатываемых программ передаются сообществу тестеров, а стабильные выпуски выпускаются для общественности чаще в качестве нового выпуска программы. Различные виды выпусков обширно используются в среде проектов свободного программного обеспечения:

- ночные снимки являются выдержками из исходного кода за определённый период времени, которые собираются и выкладываются в онлайн для использования каждому желающему. Эти выпуски создаются автоматически и являются новейшими так как отображают состояние программного тайтла за определённый отрезок времени. Они очень экспериментальные и предназначены лишь для разработчиков или опытных контрибьюторов
- разрабатываемые выпуски - это промежуточные выпуски, схожие с ночными, но в некоторой степени координируемые разработчиками. Как правило, они имеют журнал изменений, в котором приводятся изменения, внесённые с момента предыдущего выпуска. Такие выпуски предназначены для опытных тестеров, не возражающих против того, что время от времени программа может работать некорректно (иметь дефект).

- Бета-выпуски содержат предварительное состояние того, как должен выглядеть финальный выпуск. Он может быть неполностью стабильным или завершённым, однако отдельные лица, не участвующие в регулярных тестах могут опробовать и увидеть, как выглядел бы новый выпуск и что он содержит запрошенные ими исправления. Бета-выпуски также важны для дистрибутивов, так как они могут начать разработку пакетов для программы и подготовиться к финальному выпуску
- Релиз кандидаты — это кандидаты на финальный выпуск. Содержат программу, которую разработчики желают выпустить. Они ожидают некоторый период времени, так как тестеры и широкая общественность могут провести свои тесты, чтобы убедиться в отсутствии в них ошибок и прочего. Теперь в программу добавляются только исправления, но не возможности. Если новых (или важных) ошибок не обнаружено, кандидат на выпуск становится новым выпуском
- Стабильные выпуски — это финальные выпуски всего процесса разработки. Эти выпуски теперь используются пользователями и дистрибутивами, а весь процесс разработки начинается заново.

Стабильные выпуски также имеют обыкновение выпускаться в определённых градациях, что отражается их нумерацией версии. Популярной схемой нумерацией является $x.y.z$, где:

- x — основная версия; данная цифра версии обновляется в случае кардинальных изменений в программе. Зачастую такие выпуски также требуют все зависимые от них пакеты для своего обновления, а также потому, что они могут использовать возможности или библиотеки, которые были изменены.
- y — второстепенная версия; данная цифра версии обновляется всякий раз, когда программа обновляется и получает множество новых особенностей
- z — версия исправления; данная цифра версии обновляется всякий раз, когда основные исправления добавляются в программу

В качестве примера, я перечислю дату выпусков KDE 4.9. Так как KDE является полноценной графической средой, её цикл выпусков «медленнее» других. Если вы сравните цикл выпусков, к примеру, с аналогичным циклом Microsoft Windows, то обнаружите, что он всё ещё молниеносно быстрый. Конечно, это может быть похоже на сравнение яблок и апельсинов... но смотрите сами:

- 2012-05-30: выпущен KDE 4.9 beta1
- 2012-06-14: выпущен KDE 4.9 beta2
- 2012-06-27: выпущен 1-й релиз кандидат KDE 4.9
- 2012-07-11: выпущен 2-й релиз кандидат KDE 4.9
- 2012-08-01: выпущен KDE 4.9
- 2012-10-02: выпущен KDE 4.9.2
- 2012-11-06: выпущен KDE 4.9.3
- 2012-12-04: выпущен KDE 4.9.4
- 2013-01-02: выпущен KDE 4.9.5

Просто, для справки, KDE 4.10 beta 1 выпущен 21 ноября 2012 года, всего через 6 месяцев после выпуска бета-версии KDE 4.9, а KDE 4.10 (теперь именуется «KDE Software Compilation — Набор приложений KDE», так как приложения для KDE самостоятельны и следуют своему циклу релизов) выпущен 6 февраля 2013 года, - через месяцев после KDE 4.9.

2.9 Большая документационная база

Поскольку проект чаще всего не может предоставить людей и платную поддержку для ПО, его успех в большинстве своём основан на документации. Если сопроводительная документация содержит сведения о нём, опытные или независимые пользователи могут найти в ней ответы на все интересующие их вопросы.

Проекты свободного программного обеспечения обычно имеют документацию высокого профиля, которая чаще всего оказывается лучше чем документация закрытого ПО, доступная в сети. Множество больших проектов имеют в наличии всю документацию, доступную даже на нескольких языках. И если вы не нашли ответа в документации проекта, есть шансы, что где-либо один или несколько пользователей написали независимое руководство по этому ПО.

В Интернете существует множество сайтов, ссылающихся на различные ресурсы с документацией и возникает такая же проблема как и с самим СПО: зачастую у вас имеется слишком большое количество ресурсов, что затрудняет в поиске правильного документа, который мог бы проинструктировать вас по программе

через получение пользовательского опыта. Однако, в отличие от изобилия тайтлов ПО (усложняющего поиск правильного ПО, соответствующего задаче) пользователю легче узнать хороша документация или нет, таким образом нет необходимости в 'её распространении'.

2.10 Жизненный цикл ПО

Если вы приобрели ПО у маленькой, неизвестной компании, может получиться так, что через много лет этой компании больше не станет, как и поддержки самого продукта с этого момента. Что-то похожее является истиной и в отношении свободного ПО: если проект решает, что ему не хватает ресурсов для продолжения разработки (обычно по причине нехватки разработчиков), он прекращает разработку, что обычно отражается и на прекращении поддержки от пользователей.

Тем не менее, в отличие от случая с продуктом компании, исходный код свободного ПО остаётся доступным для общественности. Если вам крайне необходимо заставить ПО работать на вас, вы можете получить исходный код и продолжить его разработку самостоятельно (или заплатить другим, чтобы они выполнили это для вас). Вы также можете быть уверены, что ПО останется свободным.

Если владельцы авторского права вдруг решат, что ПО находится под другой лицензией, с которой вы в дальнейшем не согласны, вы можете получить исходный код прежде чем владельцы авторского права решат сменить лицензию и продолжить разработку в рамках этой лицензии (когда ПО все еще находится под оригинальной лицензией, но не новой) Этот процесс (когда группа разработчиков несогласна с планом разработки ПО и начинает новый проект, основанный на том же исходном коде) называется форк проекта.

Наиболее известный пример форка — создание проекта X.org, форка другого проекта под названием XFree86, который в определённый момент решил сменить лицензию. Не только это было основанием для форка: некоторые разработчики были недовольны политикой разработки касательно новых возможностей ПО и её темпом. Оба проекта всё ещё существуют, однако X.org отныне наиболее популярный из них.

2.11 Открытые стандарты

Поскольку в разработку вовлечено множество проектов, важно, чтобы они как можно больше следовали стандартам. Только соответствие открытым стандартам позволяет проектам легко и эффективно работать вместе. Далее приводятся

несколько важных стандартов или наиболее заметных спецификаций в мире свободного ПО.

2.12 Стандарт иерархии файловой системы (FHS)

Первым стандартом с которого я начну обсуждение является Стандарт иерархии файловой системы, сокращённо FHS. Этот стандарт используется почти всеми дистрибутивами и описывает расположение файлов в файловой системе Linux. Об этом можно прочитать по адресу <http://www.pathname.com/fhs>, однако многие другие ресурсы также описывают структуру FHS.

Структура файловой системы для систем Unix/Linux достаточно отличается от структуры файловой системы, замеченной в Microsoft Windows. Вместо маркировки разделов по букве, системы Unix/Linux отображают файловую систему в виде древовидной структуры, начиная от корневого каталога и далее построением каталогов и файлов. Вы можете назвать ветки в этой структуре каталогами, которые оканчиваются файлами. Если вы думаете, что ранее не встречались с файловой системой Unix/Linux, то подумайте ещё раз: URL-адреса, которые вы используете в Интернете основаны на её структуре. К примеру: URL-адрес <http://www.gentoo.org/doc/en/faq.xml> указывает на файл `faq.xml`, который расположен на сервере <http://www.gentoo.org>, в каталоге `/doc/en`. Таким образом, `/` — это корневой каталог, «`doc`» — ветка этого каталога, а «`en`» — ветка, принадлежащая ветке «`doc`».

Дистрибутивы, соответствующие FHS, позволяют пользователям легко переключаться между ними: структура файловой системы остается прежней, таким образом навигация между папками, файлами устройств ... не меняется. Это дает возможность упаковщикам создавать пакеты для нескольких дистрибутивов (до тех пор пока они используют тот же формат пакета). Но прежде всего, это позволяет пользователям одного дистрибутива Linux помочь пользователям других дистрибутивов, так как между структурой их файловых систем нет различий.

Текущая версия стандарта — 2.3, выпущена 29 января 2004 года.

2.13 Linux Standard Base

Linux Standard Base или LSB состоит из структуры, двоичной совместимости, необходимых библиотек и команд и др. для операционной системы Linux. Если дистрибутив соответствует стандарту LSB, он способен устанавливать, запускать и сопровождать LSB-совместимые (программные) пакеты.

Дистрибутивы должны соответствовать LSB если для них необходимо удостовериться в том, что они не отклоняются от этого стандарта. В последующем, LSB является попыткой удостовериться, что дистрибутивы остаются схожими с учётом библиотек, команд, или пользовательского опыта в общем. Это хорошее стремление с целью обеспечить отсутствие фрагментации в мире Linux.

Поскольку LSB является мостом, объединяющим другие стандарты, включая вышеупомянутый FHS, а также Единую спецификацию Unix (SUS), которая определяет, какой должна быть система Unix. Однако, нельзя сказать, что ОС Linux является Unix, так как нам необходимо сертифицировать её (что требует серьёзную финансовую поддержку) и эта сертификация просуществует недолго вследствие частых изменений в системе Linux.

2.14 Спецификации Free Desktop

По адресу <http://www.freedesktop.org> вы найдёте набор спецификаций для рабочего стола, которые также широко известны в сообществе свободного программного обеспечения. Несмотря на то, что они не являются стандартами (так как freedesktop не является их представителем и эти спецификации не конвертированы стандарты OASIS или ISO) многие дистрибутивы соблюдают их.

Эти спецификации определяют, как создаются и сопровождаются записи в меню, где должны размещаться значки, как должно происходить перемещение между различными библиотеками (больше относится к Qt и GTK+, графическим библиотекам KDE и GNOME).

Упражнения

1. В чём отличие между GPLv2 и GPLv3?
2. Часть стандарта LSB - ELF или формат исполняемых и компокуемых файлов, скомпилированный код которого используется различными дистрибутивами Linux/Unix. Вы можете назвать операционную систему, поддерживающую формат ELF кроме Linux/Unix?
3. Некоторые пользователи считают быстрые выпуски слабостью сообщества свободного программного обеспечения: они «вынуждены» обновлять своё ПО очень часто, даже если оно свободно это всё равно занимает некоторое время (а иногда является головной болью). Некоторые дистрибутивы пытаются помочь пользователям, предлагая только стабильное ПО (стабильное как само по себе так и в выпусках версий). Как это возможно?
4. Как стало возможным то, что многие дистрибутивы позволяют обновлять ПО до последних версий без необходимости установочного CD или переустановки с нуля?

Дальнейшие ресурсы

- + Собор и Базар [<http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/>], автор Эрик Стивен Рэймонд — эссе на тему о двух различных моделях разработки, используемых в сообществе Свободного Программного обеспечения.
- + Фонд Свободной информационной инфраструктуры [<http://www.ffii.org>] — некоммерческая организация, нацеленная на установление свободного рынка в информационных технологиях.
- Битва за программные патенты [<http://www.gnu.org/philosophy/fighting-software-patents.html>], автор Ричард Столлман — взгляд фонда GNU на программные патенты.

Глава 3

Роль сообщества

Сообщества

Самое важное достоинство СПО заключается в его сообществе. Как и любая технология или концепт, СПО имеет своих приверженцев, защищающих и возвеличивающих его. Сообщество Свободного Программного Обеспечения (далее «сообщество СПО», прим. переводчика) само по себе очень яркое, живое и жаждет помочь другим открыть удивительный мир СПО...

3.1 Сообщества

Сообщества СПО схожи с реальными сообществами, но используют Интернет как главный коммуникационный узел. Следовательно, они не группированы в сети подобно реальным, однако разбросаны по миру. Несмотря на это, Интернет гарантирует, что участники сообщества общаются с другими также как и с соседями, даже если находятся на расстоянии световых лет (фигура речи).

Интернет представляет значительную ценность для этих сообществ: вас оценивают не по цвету кожи, возрасту или по тому как вы выглядите. Важно то, как вы взаимодействуете с другими, как вы представляете себя и откликаетесь в обсуждениях. Споры в сообществе часто могут быть довольно жаркими, особенно когда в факты в теме беседы недостаточны для получения хорошего ответа. И когда споры сменяются почти обменом оскорблений, рождается флейм.

В этих случаях факты и доводы обычно далеки. Вам однозначно стоит избегать флейма в беседах, где могут приниматься решения, но его действительно невозможно предотвратить, так как это результат деятельности людей, наиболее заинтересованных в теме, которую они защищают, особенно когда нет точного ответа на флеймообразующий вопрос.

Примерами таких флеймов являются вопросы «Какой самый лучший дистрибутив Linux» или «Какой текстовый редактор мне стоит выбрать», поскольку на эти вопросы нет точного ответа: дистрибутив, наиболее хорошо подходящий одному, может быть худшим для другого, то же самое можно сказать и о текстовых редакторах. На латиноамериканском языке можно сказать *”de gustibus et coloribus non est disputandum”* (что на русский язык можно перевести как «на вкус и цвет мастера нет» прим. переводчика) и это очень верно описывает характер этих вопросов.

Когда у вас нет выбора, нет и флейма: нельзя сравнить один продукт сам с собой. Однако в мире СПО, выбор — важное понятие. У вас есть выбор среди множества свободных операционных систем (кроме Linux есть множество разновидностей BSD, Sun Solaris 10 и даже менее популярных, но обещающих систем, таких как GNU Hurd), дистрибутивов (доступно свыше ста дистрибутивов), графических сред (ни дня не проходит без священных войн а-ля GNOME против KDE), офисных пакетов и т.д.

«Лучший дистрибутив» - это неоднократно оспариваемая тема и тем не менее эта книга оказывает влияние на нее. Поэтому лучшим моим ответом для вас будет то, что лучшего дистрибутива не существует, по крайней мере не в общих чертах. Понятие «лучший» рассматривается теми людьми, у которых есть свои предпочтения на счёт операционной системы. И множество этих людей очень яростно защищает свой дистрибутив.

Сообщества дистрибутивов очень активны, так как они достаточно большие. Сообщество Gentoo известно своей отзывчивостью: канал чата Gentoo не умолкает ни минуты (более 800 участников в любое время) как и форум (свыше тысячи комментариев в день) и списки рассылок. Конечно, общий флейм обычно происходит на нейтральной стороне, но ежедневны и горячие споры на другие темы.

По этой причине большинство сообществ имеют людей, ответственных за здоровую атмосферу в беседах, и предотвращающих рост флейма. Те, кто провоцируют флейм на канале связи (называются троллями) отслеживаются операторами: они могут выгнать или даже «забанить» нарушителя правил с канала чата, операторы списка рассылок удаляют их из списка, а модераторы форумов удаляют их профили. Можно сказать, что эти люди играют роль полицейских в сообществе.

3.2 Местные сообщества

Особый вид сообщества в каком-либо регионе. Такие сообщества нередко организуют встречи (конференции, беседы, барбекю и т.п.) и предлагают помощь

населению по месту своего нахождения.

ГПЛ (Группы пользователей Linux) являются успешными примерами таких сообществ: эти группы объединяются вместе, обсуждают развитие в мире Linux и помогают другим участникам в установке Linux-систем (Инсталфесты Linux — это местные встречи, участники которых оказывают помощь в установке вашего любимого дистрибутива на компьютер). Вы можете обнаружить ГПЛ в своей местности.

Множество ГПЛ предлагает различные услуги для своих пользователей, которых обычно не встречается в сообществах коммерческого ПО. Более того, они предоставляют эти услуги безвозмездно.

- Индивидуальная, локальная помощь в установке, настройке и обслуживании дистрибутива Linux или другого СПО
- Курсы, беседы и презентации, предлагающие более глубокое погружение в мир доступного СПО
- Специфичная документация, предназначенная для нужд её пользователей

Если у вас есть время, которое вы можете уделить этому, я настоятельно советую присоединиться к местной ГПЛ — даже если вы не ищете помощи, вы всё ещё можете предложить свои опыт и знания другим обрести связи (да, социальные сети важны).

3.3 Сообщества в сети

Когда пользователи хотят обсудить на тему конкретного ПО или дистрибутива, то это нередко приводит к формированию онлайн-сообществ. Эти сообщества не организуют (или лишь немного расширяют) встречи в определённом регионе (часто «в реальной жизни») но по большей части используют Интернет в качестве площадки для встреч («онлайн»-встречи).

Онлайн-сообщества располагают преимуществом, заключающимся в том, что их члены могут быть в любой точке мира, и они, также как и ГПЛ, всё ещё предоставляют помощь своим пользователям, в большинстве случаев на безвозмездной основе:

- онлайн-помощь с установкой, настройкой и обслуживанием СПО

- В конкретных случаях, сообщества даже могут предоставить интерактивную помощь с использованием технологий, таких как SSH (Secure SHell - безопасный командный процессор, позволяющий пользователям получать доступ и работать в другой системе) и VNC (Virtual Network Computing — система удалённого администрирования, позволяющая пользователям получать доступ и работать в другой системе, используя графику, или просматривать сеансы в режиме «только для чтения»).
- курсы и онлайн-презентации
- документация, более специализированная для программных тайтлов, однако также нередко и локализованная (переведённая)

Это стало возможным благодаря различным технологиям, доступным в Интернете, включая:

- Вики (программное обеспечение с возможностью совместной разработки документации в сети) - это ПО стало достаточно популярным для разработки и выпуска документации. Его использование позволяет пользователям редактировать существующую документацию или стать авторами новой (с простым обзором), а результаты их деятельности немедленно станут доступны для других.
- (Веб)форумы, где люди могут принять участие в обсуждениях, размещая свои сообщения и отвечая на другие. Преимущество таких форумов заключается в том, что они доступны через ваш веб-браузер (которые все еще допускаются большинством фаерволов), информацию можно получить спустя продолжительное время после закрытия обсуждения, а сообщения могут быть дополнены изображениями, вложениями и отформатированным текстом.
- Списки рассылок, схожие (по функциональности) с веб-форумами, но организованные через электронную почту. Люди подписываются на список рассылки, принимают все письма, отправленные в список рассылки, на свой личный почтовый ящик. Ответы на эти письма отсылаются обратно в списки рассылок, где они снова распространяются на всех участников списков рассылок. Списки рассылок достаточно популярны в сообществе СПО, так как легко модерируются и могут быть отфильтрованы. Также, такие письма достигают адресатов быстрее, чем сообщения на веб-форумах, таким образом, вы можете рассматривать список рассылки как платформу мгновенных обсуждений.

- IRC (Ретранслируемый интернет-чат) интерактивный способ связи с множеством людей. ПО для мгновенных сообщений многим известно как MSN или Google Talk. IRC — это нечто старое, но всё ещё активно используемое, поскольку оно поддерживает чат-комнаты, где могут участвовать несколько сотен человек. IRC — это наиболее быстрая платформа для участия в беседах и может быть рассмотрена в качестве способа создания «онлайн» встреч.
- Социальные платформы, ориентирующиеся на людей, такие как Google Plus или Facebook, люди, имеющие общие интересы сотрудничают и ведут беседу на любимые темы. Их преимущество в том, что они лучше интегрированы с последними технологическими достижениями; люди с современными мобильными телефонами (смартфонами) доступны в сети продолжительное время и быстро взаимодействуют со всем что случается в сообществах.

3.4 Поддержка

Сообщества обычно выполняют роль поддержки пользователей: если у вас есть вопросы об их проекте ПО, они захотят ответить и оказать помощь. Если вы считаете, что ПО имеет недостаточную функциональность, они помогут вам дополнить её или заставить работать вместе с другими инструментами (или даже направить вас на другие проекты ПО, если они чувствуют, что вам нужно что-то, для чего их любимый инструмент не был создан).

Поддержка может быть оказана на многих этапах...

3.5 Справочники по документации

Справочник по документации обычно создаётся с одной целью: описать как сделать что-то с инструментом. Такие справочники тем не менее чаще называются HOWTO (пошаговые инструкции). В них вкладывается много усилий, так как они должны быть корректными, хорошо оформленными и завершёнными. Хорошее HOWTO — это то, после прочтения которого у вас останется меньше вопросов. Если вы спросите сообщество как сделать что-либо нужное вам и это описано в HOWTO, то вас направят к нему (иногда с более грубой отсылкой на термин RTFM или «Read The Fucking Manual» (что означает «Читайте чёртову инструкцию» - однако третье слово после артикля может читаться как «Fine» - «Хороший»).

Прочие виды документации — ЧАВО (Часто Задаваемые Вопросы), которые в общем виде являются весьма небольшими HOWTO или ответами скорее на кон-

цептуальные вопросы, чем технические. Когда вы впервые столкнулись с определенным инструментом, вам стоит прочитать ЧАВО прежде чем задать вопрос. Высокая вероятность заключается не только в том, что вы найдёте ответ на свой вопрос, вы можете узнать больше об интересующем вас инструменте.

Некоторые сообщества предлагают базу знаний. Такие системы могут рассматриваться как сборник вопросов и ответов, но в отличие от ЧАВО они не дают ответы на частые вопросы. Базы знаний чаще всего предлагают решения для специфичных настроек.

3.6 Интернет и форумы Usenet

Интернет-форумы (веб) или форумы Usenet (пользовательская сеть) являются наиболее интерактивным подходом к получению поддержки. Интернет-форумы имеют дополнительное преимущество в том, что вы можете придать тексту своих вопросы определённое форматирование: вы можете показать код команды, исключения или ошибки лучше, чем простым текстом. Вы даже можете добавить скриншоты. Форумы позволяют любому пользователю достаточно быстро получить помощь: форумы читаются многими людьми и их внешний вид достаточно прост для просмотра новых тем.

Ещё одно дополнительное преимущество интернет-форумов заключается в том, что как только ответ на вопрос получен, он сохраняется в базе данных форума.

Следовательно, весь форум может рассматриваться как база знаний с множеством ответов. Наиболее популярные темы обычно прикрепляются, что означает, что тема останется в первом ряду даже без возможности обсуждения, увеличивая вероятность её прочтения новыми пользователями.

Форумы Usenet (или новостных групп) ещё один способ поддержки, однако, нужно отметить, что новостные группы редко используются для средств СПО. Обычно вы найдёте новостную группу в случае если проект не предоставляет свой форум. (любой может запустить новую новостную группу) однако это приводит к тому, что интернет-форумы и форумы Usenet оказываются связаны: комментарии с одного форума сливаются с другим.

3.7 Списки рассылок

Наиболее прямой способ — это списки рассылок, где индивидуально просматриваются несколько десятков (или даже сотни) e-mail адресов. Список рассылки нередко оказывается быстрее форумов, поскольку множество разработчиков их

завсегда и из-за их простоты: списки рассылок представляют собой обычный текст, который можно легко фильтровать.

Большинство списков рассылок также архивируется, позволяя вам бегло пробежать по старым темам в списке. Они используются как первичный канал связи для нужд разработки, тогда как форумы подходят для пользователей и их опыта. Некоторые проекты имеют внутренние списки рассылок для разработчиков и они недоступны для общественности. Не потому, что они хотят скрыть от пользователей результаты разработки: такие списки рассылок используются для связи по проблемам безопасности, персональной информации и общения на темы, которые являются юридически сложными для защиты, в случае если их обнаружат.

3.8 Чат

Чат-беседа — в большинстве случаев почти прямая форма связи с другими участниками. Множество проектов СПО использует IRC (ретранслируемый интернет-чат) в качестве центрального канала связи. Пользователи могут легко получать помощь через IRC, где разработчики общаются и обсуждают изменения.

Чат-каналы могут быть очень популярны. Главный чат-канал Gentoo (`#gentoo` в сети `freenode`) может содержать от 800 до 1000 участников в любое время.

3.9 Встречи в реальной жизни

Периодически группы разработчиков собираются вместе для поддержки «в реальной жизни» или обсуждения развития их ПО. Во многих случаях, такие встречи предоставляют людям возможность получить интерактивную помощь. Мы обсуждали встречи GPL (где встречи в реальной жизни — частое явление), однако сообщества ПО также организовывают встречи в реальной жизни. Большинство из подобных встреч предлагает способ при котором разработчики встречаются с друг-другом (впервые), обсуждают темы и учатся друг у друга.

В некоторых случаях организуются хакфесты. Во время этих встреч, разработчики объединяются вместе с одной целью: разработать новые возможности или устранить ошибки в их ПО. Несмотря на это, это возможно и вне сети, хакфесты позволяют разработчикам свободно связываться с друг-другом и помочь другим разработчикам в их проблемах. Встречи в реальной жизни позволяют разработчикам легко обнаружить имеющуюся проблему (некоторые проблемы могут

быть слишком сложными или отнимать много времени для их документирования).

3.10 Конференции

В мире СПО конференции организуются часто. Во время этих конференций

- ведутся беседы об определённых тайтлах ПО (на тему дизайна, возможностей, развития и т.д.) или о проекте (инфраструктура, предоставляемые услуги, используемые технологии и т.д.)
- организуются выставочные стенды, где возможности проектов будут представлены широкой публике. Для дистрибутивов часто используются такие стенды для демонстрации процесса установки с CD/DVD носителей и показать системы с запущенными дистрибутивами.
- Компании предоставляют сведения о том, как они используют (или разрабатывают) СПО (и иногда привлекают разработчиков)

3.10.1 FOSDEM

FOSDEM или Европейская конференция разработчиков свободного и открытого ПО проходит в Брюсселе, Бельгия, в начале каждого года (примерно в середине февраля). Во время этой конференции обсуждается написание кода и разработка ПО, однако вы найдёте выставки, посвящённые различным проектам ПО/дистрибутивов или комнаты разработчиков (где каждый отдельный проект может предложить беседу на темы, специфичные для проекта).

FOSDEM проходит всего два дня и стал главной конференцией в сообществе Свободного Программного Обеспечения особенно в Европе, так как множество других конференций располагаются в США.

3.10.2 FOSS.IN

FOSS.IN или Конференция, посвящённая свободному и открытому ПО в Индии одна самых больших конференций FOSS Азии. Она проводится в конце каждого года в Бангалоре, Индия, в рамках которой ведутся беседы, обсуждения, симпозиумы, встречи и т.д. от международных докладчиков и разработчиков.

3.10.3 LinuxTag

LinuxTag — это выставка свободного ПО, главным образом фокусирующаяся на операционных системах и решениях, основанных на Linux. В отличие от FOSDEM, LinuxTag фокусируется больше на интеграции Linux (и свободного ПО) в больших средах, предлагая выставки для коммерческих компаний и некоммерческих организаций.

Её слоган «Where “.COM” meets “.ORG”» («Где компании встречаются с организациями»). Вы можете посещать LinuxTag ежегодно весной.

Упражнения

1. Попробуйте найти способы обсуждений в сети (веб-форумы, списки рассылок, IRC)

Дальнейшие ресурсы

- + The Ottawa Linux Symposium [<http://www.linuxsymposium.org>] проводится ежегодно в Оттаве, Канада, во время летних каникул.
- + Linux Kongress [<http://www.linux-kongress.org>] почти всегда проводится в Германии, но несмотря на это, один раз мероприятие проводилось в Англии.
- + Linux.conf.au [<http://linux.conf.au/>] проходит в Австралии в начале каждого года.
- + Ohio Linux Fest [<http://www.ohiolinux.org/>] проводится в Огайо каждой осенью.
- + Linux Fest Northwest [<http://www.linuxfestnorthwest.org/>] проводится в Вашингтоне каждой весной.
- + SCaLE (Southern California Linux Expo) [<http://scale7x.socallinuxexpo.org/>] выставка проводится поздней зимой в Южной Калифорнии.
- + Ontario Linux Fest [<http://onlinux.ca/>] ЛинуксФест в Онтарио
- + Конференция LinuxWorld и Expo [<http://www.linuxworldexpo.com/>]
- + Freed.IN [<http://freed.in/>]

Глава 4

Запуск Linux

Знакомство

Чтобы работать с Linux, вам необходим доступ к его среде. Лучшей средой является установленный на вашем компьютере дистрибутив Linux, однако вы можете использовать варианты такие как Linux Live CD (загрузочный компакт-диск, содержащий готовую к запуску систему). Установка Linux весьма дистрибьютоспецифична, а процедура установки Gentoo хорошо документирована на официальном сайте [<http://www.gentoo.org/doc/en/handbook>]. Пока что я не буду касаться процедуры установки. Позднее, в этой книге, я сфокусируюсь на установке Gentoo с помощью командной строки, поскольку так вы сможете лучше понять её различные этапы. Далее, я расскажу о некоторых темах, специфичных для Linux, которые очень важны для понимания, но не буду перегружать излишними деталями: позднее почти каждая тема будет широко рассмотрена.

Пользователи, не использовавшие в своей работе Linux или Unix, обнаружат значительное различие между ними (в сравнении с Microsoft Windows). В разделе «Архитектура системы» я проведу краткий обзор основных различий между этими ОС. В дальнейшем, я подробно объясню, как использовать интерфейс командной строки Linux (который вам однозначно стоит освоить, если вы собираетесь работать с Gentoo Linux). И наконец, я покажу, как искать помощь по командам.

4.1 Архитектура системы

Linux — многопользовательская система. Администратор — это пользователь под названием root, который может делать с системой все что угодно. Несмотря на то, что можно прекрасно работать в системе в качестве администратора, настоятельно рекомендуется создать индивидуальные учётные записи пользователей

для любого желающего, кому необходим доступ к системе и дать этим пользователям доступ только к важным ресурсам или административным командам (в наличии имеются средства, позволяющие пользователю переключаться с одной учётной записи на другую, для запуска определённых команд от другого пользователя).

Для вашего персонального компьютера это означает, что вам придётся создать одну дополнительную учётную запись для себя. Если ПК используется несколькими людьми (скажем, домашний ПК или рабочая станция в офисе), вы, вполне вероятно, захотите создать учётные записи для каждого возможного пользователя. Использование одной записи как общей — плохая привычка: если вам нужно поделиться файлами и каталогами с другими, есть несколько способов лучше.

Используемая вами учётная запись ограничена, но однозначно не является непригодной к использованию. Приведу несколько отличий между непривилегированным пользователем и пользователем `root`:

- в возможности персонального аккаунта входит изменение файлов, созданных в нём, таким образом, нет какой-либо возможности повредить систему (так как файлы операционной системы неизменяемы для персональных аккаунтов). Это также означает, что пользователь такого аккаунта не сможет устанавливать какое-либо ПО в систему (это нормально — я расскажу вам, как повысить свои привилегии, чтобы использовать их только при необходимости).
- Пользователь персонального аккаунта не сможет заполнить раздел жёсткого диска, что могло бы подвергнуть систему опасности: по умолчанию 5% свободного пространства каждого раздела резервируется только для нужд привилегированных пользователей.
- Если подозрительное событие использует ваш аккаунт, журнальные файлы могут помочь в отслеживании случившегося, однако пользователь персонального аккаунта не способен изменить их.
- ...

Любая задача, запущенная пользователем, происходит в виде процесса в системе. К примеру, пользователь, запустивший браузер приводит как минимум к появлению одного процесса, указывающего на программу-браузер. По умолчанию этот процесс запускается с привилегиями пользователя, запустившего его, и тем не менее, он имеет доступ к его файлам и ресурсам. Ещё одна причина,

по которой не следует использовать аккаунт root: вам бы не хотелось, чтобы подозрительный файл сценария (скрипт) выполнялся в вашем браузере и вызвал тем самым ошибку, пригодную к использованию злоумышленниками (нет, это не фантазия, это вполне может произойти), которые могут заставить браузер стирать важные системные файлы. . .

Файлы и каталоги сами по себе не скрывают свои разрешения, представляют три набора разрешений: одно для владельца, второе для владеющей группы и третье — для всех. Более подробно разрешения файлов рассмотрим позднее.

4.2 Учётные записи пользователей

Поскольку пользователь root является администратором системы без ограничений в привилегиях, то любые его действия потенциально могут нанести вред системе:

- удалить не те файлы или каталоги (при этом их нельзя восстановить)
- удаление или манипуляции с системными файлами (возможно приведет к неработоспособности системы, да, перезагрузка это не исправит)
- отключить сетевой доступ или внести изменения в поведение сети
- размещать файлы в некорректных местах (возможна потеря файлов и конфликт с системными операциями)
- совершить «убийство» процесса или напротив манипуляции с некорректными процессами (что возможно приведёт к неработоспособности сервисного сценария, его отключению или неверному функционированию)

По этой причине, должны быть созданы пользовательские учётные записи. Обычная учётная запись:

- может редактировать или удалять файлы, принадлежащие только этому пользователю
- не может вмешиваться в общесистемные настройки (включая сетевой доступ, доступ к устройству и т.д.)
- размещать файлы только в своём (предназначенном) домашнем каталоге
- может только убивать / управлять процессами, запущенными им самим

4.2.1 Важное примечание

Пользовательские аккаунты, не имеющие административных привилегий, всё ещё способны нанести вред. К примеру, запущенная пользователем неработоспособная программа может удалить все файлы пользователя в системе. Тем не менее, это основная трагедия для конечных пользователей (пожалуйста, не забывайте создавать регулярные резервные копии важных данных), тогда как система всё ещё может загружаться, так как её файлы нетронуты.

Вслед за пользовательскими аккаунтами любой установленный Linux также имеет системные аккаунты. Это обычные учётные записи, схожие с пользовательскими (с такими же ограничениями), однако они не используются для входа в систему (у них нет действующего пароля), обычно не имеют отдельного домашнего каталога и применяются операционной системой для запуска определённых сервисов с ограниченными правами (повышая безопасность) вместо административных.

4.3 Процессы

Процесс — это задача, запущенная в системе. Каждый процесс запускается от имени конкретного пользователя: это не только означает, что процесс может получить доступ к ресурсам, доступным для этого пользователя, но и то, что только этот пользователь (и пользователь root) могут управлять им (например «убить» его).

Каждый процесс имеет одного родителя (за исключением высшего процесса, коим всегда является `init`) и может иметь несколько потомков. Если процесс погибает, то все его потомки также погибнут (поэтому никогда не «убивайте» процесс `init`, так как это приведёт к уничтожению всех процессов в системе).

Запущенные процессы не обязательно потребляют такты ЦП. На простаивающей системе может быть запущено несколько десятков (или даже сотни) процессов. Так происходит потому, что они имеют три различных состояния: активное, спящее, ожидающее, остановленное и т.д. К счастью, операционная система Linux очень эффективно балансирует требования приложений с имеющимися ресурсами, поэтому вы, как конечный пользователь, не заметите большого количества запущенных процессов.

4.3.1 Примечание

Один процесс также может иметь множество нитей. Нити — часть приложения (но управляются операционной системой) и позволяют параллельно запускать код внутри приложения. Программа с несколькими нитями всё ещё является одиночным процессом.

4.4 Файлы и файловая структура

В Linux (и во всех Unix и Unix-подобных ОС) файловая система имеет иерархическую структуру. Сверху, в самом начале, расположена корневая файловая система, которая именуется «/». Ниже следуют файлы и каталоги, соответствующие стандарту, описанному в предыдущем разделе, он назывался «Стандарт иерархии файловой системы». Каждая позиция в структуре диаметрально противоположна. Возьмём к примеру путь `/home/swift`, который является домашним каталогом учётной записи «swift». Другой пример — это каталог `/usr/share/doc/googleearth-4.2.198.2451/`, где размещается документация определённой версии программы Google Earth.

Вслед за обычными файлами (такие как текстовые файлы или файлы, имеющие отношение к программе) и каталогами в Linux также имеется понятие:

- *ссылок*, которые названы по-другому, ссылаются на те же файлы, но под другими именами (что позволяет одному файлу иметь несколько ссылок),
- *символических ссылок*, которые указывают на другие файлы (отличающиеся от обычных ссылок, так как они всего лишь указатели и могут указывать на что-то, чего не существует) и в некоторой степени сравнимы (но не совсем) с ярлыками в Microsoft Windows.
- *именованных каналов* или *FIFO*, являющиеся временными буферами, где один процесс может писать, а другой считывать
- *сокетов доменов unix*, которые создаются одним процессом, а остальные несколько процессов могут записывать в них информацию или считывать её из них
- *файлы символьного и блочного устройств*, которые представляют оборудование в системе, позволяя обмениваться данными с устройством посредством обычных операций с файлами

Все эти специальные файлы представлены как обычные файлы, расположенные в файловой системе Linux. Возьмём к примеру устройство `/dev/sda1`. Оно представляет собой известный системе первый раздел (1) на первом (a) устройстве SCSI (sd). `/dev/sda1` — это файл блочного устройства, так как операции чтения и записи происходят в блоках (определённый объем данных), в то время как операции символьного устройства основаны на посимвольном методе.

В следующей главе я расскажу о файлах устройств и о том, как Linux управляет ими.

4.5 Разрешения

Файлы (и каталоги, а также любые специальные файлы, упомянутые ранее — в этой книге я предпочту ссылаться на них также как на «файлы») связаны с тремя типами разрешений: один из которых принадлежит владельцу, другой - группе и третий всем остальным. Каждый тип содержит несколько флагов:

- доступ к файлу или каталогу в режиме чтения
- доступ к файлу или каталогу в режиме записи
- разрешение на исполнение файла (возможно, по причине того, что он является двоичным файлом приложения или сценарием) или разрешение на вход в каталог (это означает, что вы можете переместиться внутрь каталога)

Помимо них, для исполнения файлов и каталогов могут быть заданы дополнительные разрешения, но на данный момент они менее важны для понимания:

задание файлу или каталогу идентификатора пользователя/группы (когда файл исполняем, процесс запускается от имени владельца, но не от имени пользователя/группы, запустившей его) (применяется только к группе; в этом случае, файлам, созданным в каталоге автоматически присваивается владелец группы каталога, что позволяет наиболее легко обмениваться файлами между аккаунтами (учётными записями))

флаг, запрещающий удаление (запрещает удалять файлы в задействованном каталоге, за исключением тех случаев, когда пользователь является владельцем файла и

Этот флаг нуждается в некотором объяснении. Он используется при большом количестве каталогов, доступных для записи (известные как каталог `/tmp`, где хранятся временные файлы) и используется для предотвращения удаления пользователями файлов, которых они не создавали. В остальных случаях каталог, доступный всем для записи, позволяет другим пользователям удалять файлы из него (поскольку они тоже имеют доступ к каталогу в режиме записи). Флаг, запрещающий удаление также известен как флаг *sticky*.

4.6 Использование командной строки

Большинство дистрибутивов Linux после загрузки предоставляют вам графический дисплей авторизации или даже выполняет вход в систему автоматически. Но не случае с Gentoo Linux. Более того, вас встретит приглашение командной строки, которое попросит, чтобы вы идентифицировали себя:

```
This is seaheaven.homeworld (Linux x86_64 3.8.5) 22:30:00  
seaheaven login :
```

В этом приглашении вас просят ввести имя пользователя для учетной записи, которое, вы, несомненно, создали в ходе установки. Далее вам попросят ввести пароль. Если имя пользователя существует и пароль совпадает, вас поприветствует приглашение командной строки. Следующий листинг отображает обычное приглашение для учётной записи пользователя «captain» в системе с именем «seaheaven»:

```
captain@seaheaven ~ $
```

Оно структурировано следующим образом:

- имя пользователя, зарегистрированного в системе
- имя системы, где зарегистрирован пользователь
- текущее расположение пользователя на файловой системе (означает его домашний каталог)имя системы, где зарегистрирован пользователь
- символ приглашения, указывающий на тип пользователя, обычный (\$) или root (#).

В следующих разделах я вкратце познакомлю вас с миром командной строки Linux.

4.7 Ориентирование в системе

Приглашение — это обычно что-то отображаемое командным процессором (shell). Командный процессор, shell — интерактивная программа, которая принимает вводимые пользователем команды и выполняет их в системе. Приведу наиболее удачный пример команд, используемых для ориентирования по файловой системе:

pwd показывает текущий рабочий каталог (где в данный момент находится пользователь)

cd позволяет пользователю сменить каталог, в котором он находится (переместиться на другой)

ls отображает содержимое текущего каталога (листинга) или любого другого по запросу

Пример пользовательского сеанса показан ниже:

Для перемещения с одного каталога на другой имеется несколько способов использования команды **cd**. Заметьте, что команда **pwd** используется, чтобы показать текущее местонахождение. Она не является частью процесса перемещения между каталогами!

Вы можете использовать относительный путь дерева каталогов. В этом примере, команда «**cd Documents**» переходит от каталога `/home/captain` к каталогу `/home/captain/Documents` через относительный путь «Documents».

```
$ pwd
/home/captain
$ cd Documents
$ pwd
/home/captain/Documents
```

Вы можете вернуться назад используя специальный символ `'..'` (две точки), таким образом, команда «**cd ..**» перейдет с каталога `/home/captain/Documents` на каталог `/home/captain`.

```
$ pwd
/home/captain/Documents
$ cd ..
$ pwd
/home/captain
```

Вы также можете использовать абсолютный путь. К примеру, чтобы немедленно переместиться в `/etc/init.d/` (специальный каталог, о котором будет рассказано позже), вы можете набрать команду «**cd /etc/init.d**».

```
$ pwd
/home/captain/Documents
$ cd /etc/init.d
$ pwd
/etc/init.d
```

Специальный символ `~` (тильда) означает собой домашний каталог, поэтому команда «**cd ~**» перенесет вас в этот каталог. Более того, даже простой ввод команды «**cd**» выполнит то же самое. Символ `~` может быть использован в двух способах, для указания на ваш домашний каталог (команда `cd ~/Documents` переместит в каталог `/home/captain/Documents`) или на чей-то другой. В дальнейшем, если вы добавите имя пользователя к символу «`~`», то команда `"~raghat/public_html/"` преобразуется в `/home/raghat/public_html`.

```
$ pwd
/etc/init.d
$ cd ~/Documents
$ pwd
/home/captain/Documents
```

Если по каким-то причинам выполнение этой команды заканчивается неуспешно (отсутствует разрешение на вход в каталог или его не существует), вас уведомят о неудаче.

```
$ pwd
/home/captain
$ cd /etc/cron.daily
bash: cd: /etc/cron.daily: Permission denied
$ pwd
/home/captain
```

4.8 Просмотр данных

Чтобы просмотреть содержимое каталога, вы используете команду **ls**. По умолчанию она отображает файлы и каталоги, однако имеет множество опций. Я не пытаюсь объяснить предназначение каждой из них по отдельности, однако некоторые достаточно важны для усвоения.

Прежде всего, это опция «-l», которая отображает не только файлы и каталоги, но и информацию о каждом из них:

```
captain@seaheaven ~ $ ls -l
drwxr-xr-x 2 captain users 4096 2013-10-31 22:24 Documents
drwxr-xr-x 2 captain users 4096 2013-10-31 22:24 Movies
drwxr-xr-x 2 captain users 4096 2013-10-31 22:25 Music
drwxr-xr-x 2 captain users 4096 2013-10-31 22:24 Pictures
-rw-r--r-- 1 captain users 66 2013-10-31 22:30 opentasks.txt
```

Кроме всего прочего в этой информации содержатся три типа разрешений (три раза по три символа: Read (Чтение), Write (Запись) и eXecute (Исполнение), пользователь-владелец и группа, размер (в этом примере - множество каталогов по 4 килобайта) и дата их создания.

Ещё одна интересная опция - «-a», с помощью которой команда **ls** отображает все файлы (включая скрытые). *Скрытые файлы* в системе Unix или Linux начинаются с точки. В своем домашнем каталоге вы обнаружите множество скрытых файлов, которые по умолчанию скрыты. Приведенный ниже пример иллюстрирует набор таких файлов.

```
captain@seaheaven ~ $ ls -a
Documents          Movies      Music      Pictures
.history          .maildir   .profile   .vimrc
```

Это не означает, что скрытые файлы действительно скрыты, они не отображаются в обычном режиме отображения и не захламляют вывод. Большинство этих файлов конфигурационные (например `.vimrc` является файлом настроек для редактора Vim в каталоге текущего пользователя).

Как и в случае с большинством утилит командной строки, вы можете комбинировать аргументы; даже указать их вместе, таким образом **`ls -l -a`** является тем же, что и **`ls -la`**. Их порядок не имеет значения, они одинаковы.

4.9 Анализ вывода команды **`ls -l`**

Вы обнаружите, что регулярно пользуетесь командой «`ls -l`» для получения нужной информации о файле или каталоге. Представляю вам краткое объяснение различных полей из `ls`:

```
$ ls -l 12-servicemanagement.xml
-rw-r--r-- 1 swift users 26976 Apr 22 21:21 12-servicemanagement.xml
```

- **`-rw`** — тип и привилегии файла (в данном случае: чтение и запись) для пользователя («swift»)
- **`-r--r--`** — привилегии файла (в данном случае: только чтение) для группы с доступом («users»)
- **`1`** — количество жестких ссылок на файл или каталог
- **`swift`** — пользователь, владеющий файлом или каталогом
- **`users`** — группа, владеющая файлом или каталогом
- **`26976`** — размер в байтах
- **`Apr 22 21:21`** — дата последнего изменения
- **`12-servicemanagement.xml`** — название файла или каталога

4.10 Базовая манипуляция с файлами

Основными операциями при работе с файлами является их копирование, перемещение, переименование и удаление, что и мы и объясним далее...

4.10.1 Копирование файлов и каталогов

Для копирования вы можете использовать команду **ср**. Её базовый синтаксис прост: *ср источник назначение*. Назначением может быть любой каталог в который будет скопирован файл. Если это будет файл, то он будет перезаписан исходным (под таким же именем).

```
captain@seaheaven ~ $ ls Documents
captain@seaheaven ~ $ cp opentasks.txt Documents/tasks.txt
captain@seaheaven ~ $ ls Documents      tasks.txt
captain@seaheaven ~ $ cp opentasks.txt Documents
captain@seaheaven ~ $ ls Documents
opentasks.txt    tasks.txt
```

По умолчанию, команда **ср** создает файл назначения с привилегиями и правами доступа пользователя, запустившего её. В следующем примере я скопирую файл `/etc/inittab` в текущий каталог, который обозначен точкой (.):

```
$ ls -l /etc/inittab
-rw-r--r-- 1 root root 1890 Feb 15 20:39 /etc/inittab
$ cp /etc/inittab .
$ ls -l inittab
-rw-r--r-- 1 swift users 1890 Apr 22 22:49 inittab
```

Вы можете «попросить» **ср** сохранить всю информацию файла (привилегии, владельца, временную метку), однако для этого вам нужны привилегии пользователя `root`.

Чтобы рекурсивно скопировать каталог и всё его содержимое, примените аргумент **-r**. В следующем примере, скопируем каталог под названием `workdocuments`, а также все файлы и подкаталоги в нём, в текущий рабочий каталог.


```
$ cp -r /media/usb/workdocuments .
```

4.10.2 Перемещение и переименовывание файлов и документов

Чтобы переместить файл, следует использовать команду `mv`. Её синтаксис `mv источник назначение`. Если назначением является не каталог, а файл, то данная команда также выполняет переименовывание файлов. В следующем примере файл `task.txt` будет перемещён из каталога `Documents` в каталог `/media/usb/workdocuments`:

```
captain@seaheaven ~ $ ls Documents/  
opentasks.txt      tasks.txt  
  
captain@seaheaven $ mv Documents/tasks.txt /media/usb/workdocuments  
captain@seaheaven $ ls Documents/ opentasks.txt  
captain@seaheaven $ ls /media/usb/workdocuments tasks.txt
```

4.10.3 Удаление файлов и каталогов

Последняя задача — удаление файлов. Команда `rm` имеет следующий синтаксис: `rm имя_файла`. Чтобы удалить весь каталог, вам необходимо указать его имя, однако это сработает, только если он пустой. Иначе следует указать параметр `-r` (необходим для рекурсивного удаления каталога и всех файлов внутри).

```
captain@seaheaven ~ $ rm Documents/opentasks.txt  
captain@seaheaven ~ $ rm tasks.txt
```

Популярный ряд параметров `rm -rf`: это действие рекурсивно удаляет файлы и каталоги, а также заставляет команду не запрашивать подтверждение от пользователя (`force` — принудительно). Удостоверьтесь в том, что вы знаете, что делаете, так как Linux не имеет возможности обратить такие изменения!

```
$ rm -rf Documents
```

частью, команда `rm` работает с привилегиями пользователя, запустившего её, что не позволит удалить системные файлы без привилегий пользователя `root`:

```
$ rm -f /etc/passwd
```

```
rm: невозможно удалить «/etc/passwd»: Отказано в доступе
```

4.10.4 Редактирование текстовых файлов

Знание о том, как редактировать текстовые файлы является важной задачей любого пользователя Linux. Существует множество текстовых редакторов, таких как популярные Vim и Emacs. В Gentoo рекомендуется использовать Nano в качестве редактора начального уровня, так как он понятен и легок в использовании. Я же предпочитаю Vim.

```
captain@seaheaven ~ $ nano opentasks.txt
```



Рис. 4.1: Файл `opentasks.txt`, открытый в редакторе Nano в качестве рабочего документа

Первая строка отображает имя открытого файла, тогда как последние две строки набор команд, позволяющих работать с файлом. Символ `^` означает, что вам необходимо использовать клавишу Control (Ctrl) при использовании комбинации Ctrl+O, чтобы сохранить файл. Для выхода из редактора, нажмите Ctrl+X.

Некоторые текстовые файлы весьма чувствительны к новым строкам — это конфигурационные файлы — одни из наиболее важных. По умолчанию, при достижении боковой колонки nano преобразовывает строки автоматически. Чтобы

«сказать» nano, что вам нужно сохранить их в исходном виде, добавьте аргумент **-w**: **«nano -w opentasks.txt»**. Рекомендуется при редактировании конфигурационных файлов.

4.11 Просмотр текстовых файлов

Если вы хотите просмотреть текстовый файл, но не желаете запускать текстовый редактор, можно использовать утилиты **cat** или **less**. Это также позволит вам просматривать содержимое файлов, не допуская случайных изменений.

С помощью **cat** вы «вываливаете» содержимое файла в терминал (или консоль). Если файл большой, текст пробежит очень быстро. Можно попытаться прокручивать его, используя комбинацию клавиш **Shift+PgUp**, одна длина этого буфера ограничена.

С помощью **less**, можно просматривать содержимое файла постранично. Команда **less** называется пейджером (pager), поддерживает прокрутку вверх / вниз (при помощи клавиш со стрелками или **PgUp/PgDown**), поиск определенного текста и т.д.

Использование утилиты **less** не ограничивается одним лишь просмотром содержимого текстовых файлов, она применяется и для man-страниц (о которых мы поговорим чуть позже в этой главе).

Таблица 4.1: Краткий список часто используемых комбинаций утилиты **less**

Прокрутить вниз на одну строку	Клавиши e, j <ввод> или со стрелкой вниз
Прокрутить выше на одну строку	Клавиши u, k <ввод> или со стрелкой вверх
Прокрутить вниз на одну строку	Клавиши f , <пробел> или PgDown
Прокрутить выше на одну строку	Клавиши b или PgUp
/ <искомый текст>	Запрос <искомого текста> в начале, используйте « n » для подтверждения.
h	Отобразить экранную справку
?<искомый текст>	Запрос <искомого текста> в конце, используйте « ? » для подтверждения.

Если вам интересно, имя **less** — это отсылка к старому, но всё ещё поддерживаемому пейджеру **more**. С помощью **more** вы также можете просматривать файлы постранично, однако он поддерживает только прокрутку вниз на одну строку (<ввод>) или на одну страницу (<пробел>). Таким образом, **less** больше... чем **more**.

4.12 Создание ссылок на файлы и каталоги

Linux поддерживает два вида ссылок: символические и жесткие. Вам может быть любопытно, почему я рассказываю об этом во вводной главе о запущенном Linux. Что ж, потому, что символические ссылки довольно часто используются в Linux, тогда как вы обнаружите, что жесткие ссылки используются намного реже. На самом деле, ранее мы уже видели вывод команды, имеющей отношение к жестким ссылкам: взгляните на вывод команды «**ls -l**».

Символические ссылки, также известные как симлинки, просты для понимания. Это не существующие файлы, но ссылаются на них где либо в файловой системе. Они ссылаются на строку размещения, которая позволяет ссылке существовать, даже если файлы, к которым она обращается, (больше) не существуют. Например, у вас может быть символическая ссылка, указывающая на `/home/swift/orpentasks.txt`, даже в том случае, если этот файл (больше) не существует.

Жесткие ссылки, напротив, прямые. Они ссылаются на то же расположение на жестком диске, используемое в качестве точки назначения (через иномы), а файловая система не освобождает пространство, занятое им до тех пор, пока ссылки на это расположение не перестанут существовать. На самом деле, каждый файл, который вы видите в файловой системе, это ссылка на его расположение. Используя жесткие ссылки, можно создавать дополнительные ссылки.

Из-за своей технологии, такие ссылки обращаются только к файлу, расположенному на том же накопителе и не воспринимают каталоги. Чтобы создавать ссылки, используйте `ln` (жесткие ссылки) или `ln -s` (симлинки):

```
$ ln имя_файла_назначения новое_имя_файла
$ ln -s имя_файла_назначения новое_имя_файла
```

Благодаря им, вы можете управлять файлами намного легче. Например, у вас может быть важный текстовый файл, расположенный по адресу `/Documents/Work/Thesis/myThesis.txt`, который ссылается на него из домашнего каталога пользователя (ссылка `/myThesis.txt` ведет на `/Documents/Work/Thesis/MyThesis.txt`) тем самым, не придется просматривать все каталоги каждый раз, когда нужно открыть или отредактировать файл.

4.13 Автодополнение для файлов и команд

Эффективная возможность большинства командных интерпретаторов Linux - автодополнение для файлов и команд.

К примеру, при редактировании файла под названием `opentasks.txt`, можно написать «`nano o`», а затем нажать клавишу `<tab>`. Если набранный текст «`opentasks.txt`» принадлежит файлу или каталогу, начинающемуся на букву «`o`», командный интерпретатор автоматически дополнит команду так, что она примет вид «`nano opentask.txt`». Если же имеется больше файлов или каталогов, начинающихся на «`o`», двойное нажатие клавиши `<tab>` позволит рассмотреть список претендентов на выбор.

То же верно и для команд. Если слово `nano` начинается на `na` (это не так, но предположим), то набрав его часть `na` и нажав клавишу `<tab>`, можно дополнить её остальной частью. Нажатие `<tab>` дважды выведет список всех совпадений не только для команд:

```
$ na<tab><tab>
namei          nautilus
nano           nasl
nasm           native2ascii
$ na
```

4.14 Переключение терминалов

Одно из преимуществ Unix/Linux заключается в поддержке нескольких терминалов. Когда вы входите в систему, то обычно видите один из (виртуальных) терминалов так, словно он единственный. Можно переключаться с одного терминала на другой используя комбинацию `Alt+F#`, где `F#` - это `F1`, `F2`, ... и другие клавиши. Если вы находитесь в графической сессии, следует использовать `Ctrl+Alt+F#`. Графический дисплей расположен на первом терминале, который не может быть использован для входа (в большинстве случаев `Alt+F7`).

Поддержка нескольких терминалов позволяет вам входить в систему несколько раз и выполнять необходимые задачи параллельно, через различные терминалы. К примеру, при конфигурировании программы, удобно иметь её конфигурационный файл, открытым в одном терминале и саму программу, запущенной в другом.

Подсказка: если вы работаете с командной строкой, то можете использовать команду **chvt** чтобы переключаться между терминалами: выполнение **chvt 2** переключает на второй терминал (имеет сходство с Alt+F2).

4.15 Выход из сессии

Чтобы выйти из существующей сессии, введите **enter** или нажмите Ctrl+d:

```
captain@seaheaven ~ $ exit
This is seaheaven.homeworld (Linux x86_64 3.8.5) 22:30:00
seaheaven login:
```

И наконец, если хотите выключить систему, сперва нужно стать суперпользователем. Вы можете сделать это, переключившись на другой терминал и войдя в систему как суперпользователь (root), но можете также использовать команду **su**. Эта команда попросит вас ввести пароль суперпользователя, после чего вы станете известным системе как пользователь с его привилегиями.

```
captain@seaheaven ~ $ su -
Password: (Введите пароль root)
root@seaheaven ~ #
```

Теперь, можно выполнить команду **shutdown** для немедленного выключения (-h) или перезагрузки (-r) системы:

```
root@seaheaven ~ # shutdown -h now
```

4.16 Получение помощи

В первое время Linux может казаться довольно сложным, поскольку утилиты, используемые новичками являются теми же, которые используются опытными пользователями. Новички не смогут понять предназначение расширенных параметров этих утилит. Лишь настоящие книги, такие как эта, могут помочь поль-

зователям найти свой путь, однако в то же время любой пользователь, будь то начинающий или опытный, нуждается в отправной точке информации. К счастью, существуют несколько ресурсов, к которым вы можете обратиться...

4.16.1 Man-страницы

Man страница, также называемая страницей с инструкцией доступна на системе Linux через команду `man`. Используя эту команду, вы можете получить информацию, объясняющую что она делает, какой синтаксис необходимо использовать, какие параметры она поддерживает и какие команды или файлы имеют к ней отношение.

4.17 Выключение

```
~$ man emerge
```

Страница с инструкцией обычно структурирована следующим образом:

- название команды с однострочным описанием
- синопсис команды, отображающий поддерживаемый командой синтаксис
- параметры команды с объяснением каждого из них
- дополнительная информация о команде, такая как, ограничения по использованию, поведение команды по умолчанию,
- сведения об авторских правах и владельце
- другие связанные с этой страницы

Страницы с инструкциями прилагаются не только к командам, но и к файлам (конфигурационные или файлы устройств), системным вызовам или функциям библиотек (сведения, касающиеся программирования) и даже к стандартам и соглашениям. Эти разделы распределены по соответствующим секциям и соответственно пронумерованы:

1. пользовательские команды
2. системные вызовы

3. функции библиотек
4. специальные файлы (файлы устройств)
5. форматы файлов и протоколы
6. игры и развлекательные программы
7. обзор, соглашения и различные разделы
8. административные и привилегированные команды

Для получения информации об этих разделах, откройте ознакомительную страницу к каждому из разделов. К примеру, ознакомительная страница к играм доступна по команде **man 6 intro**. Важно знать, какой раздел доступен, особенно если информация содержится в различных разделах под тем же названием. Например, passwd: имеется страница с инструкцией для команды **passwd (man 1 passwd)** и еще одна страница для файла passwd (**man 5 passwd**) Если ввести команду **man passwd**, то отобразится первый раздел или топик для этой страницы (в данном примере, первый раздел).

Попав на страницу с инструкцией, вы можете ориентироваться в ней, используя клавиши со стрелками вверх/вниз и PgUp//PgDown. Также можно выполнить поиск по конкретному запросу (нажмите / и введите искомое слово) и многое другое. Полное объяснение всех команд просмотрщика инструкций (который во многих системах является утилитой less) можно получить, нажав 'h'.

Чтобы завершить просмотр страницы, нажмите 'q'.

Если вы уверены, что страница с инструкцией имеется где-то в системе, но не знаете ее название, можно использовать команду **apropos** (которая эквивалентна команде **man -k**). Если передать ключевое слово команде, она отобразит список страниц, связанных с этим запросом. Чтобы создать базу данных, сперва необходимо запустить команду **makewhatis**. На большинстве система она выполняется в фоновом режиме ежедневно, согласно запланированной задаче.

```
~ apropos ebuild
```

4.17.1 Информационные страницы

Другая справочная утилита на вашей системе — это справочная система **info**. Инструмент **info** позволяет просматривать **info**-страницы в иерархическом пред-

ставлении (переход к предыдущей/следующей секции, выше, ...) а так же с использованием гиперссылок (выбор топики на странице для перехода к разделу соответствующего топики).

Давайте взглянем на info-страницу GCC:

```
~$ info gcc
```

```
...
```

```
File: gcc.info ,
```

```
Node: Top,
```

```
Next: G++ and GCC,
```

```
Up: (DIR)
```

```
Introduction
```

```
*****
```

This manual documents how to use the GNU compilers, as well as their features and incompatibilities, and how to report bugs. It corresponds to GCC version 4.1.2. The internals of the GNU compilers, including how to port them to new targets and some information about how to write ...

В начале страницы можно обнаружить последовательность документов (тут нет предыдущего или следующего раздела, так как это первая страница). Следующий топик (к которому можно перейти нажатием n) называется «G++ и GCC».

В самом тексте вы найдете куски текста, разделенного по двум колонкам:

- * G++ and GCC:: You can compile C or C++ programs.
- * Standards:: Language standards supported by GCC.
- * Invoking GCC:: Command options supported by 'gcc'.
- * C Implementation:: How GCC implements the ISO C specification.
- * C Extensions:: GNU extensions to the C language family.
- * C++ Extensions:: GNU extensions to the C++ language.
- * Objective-C:: GNU Objective-C runtime features.

Являясь ссылками, они позволяют вам перемещаться по топикам. Наведите курсор на надпись «Invoking GCC» и нажмите клавишу ввода:

File: gcc.info, Node: Invoking GCC, Next: C Implementation, Prev: Standards, 3 GCC Command Options ***** When you invoke GCC, it normally does preprocessing, compilation, assembly and linking. The "overall options" allow you to stop this process at an intermediate stage. For example, the '-c' option says not to run the linker. Then the output consists of object files output by the assembler.

Теперь вы находитесь в блоке под названием «Invoking GCC». Следующий раздел - «C Implementation», а предыдущим (p) был «Standards», и если вы пройдете дальше (**u**), то снова окажетесь в самом начале документа.

Информационные страницы часто используются различными проектами GNU (такими как GCC). Тем не менее, страницы с инструкциями более популярны, возможно, поскольку разработчики находят их простыми для написания.

4.18 Краткая справка по синтаксису

Если вы не ищете объяснения команды, но хотите знать синтаксис, большинство команд предоставляет краткую справку при помощи аргументов

-h или -help.

\$ man -h

man, version 1.6e

usage: man [-adfhtwW] [section] [-M path] [-P pager] [-S list]
[-m system] [-p string] name ...

a : find all matching entries

c : **do** not use **cat** file

d : print gobs of debugging information

D : as **for** -d, but also display the pages

f : same as whatis(1)

h : print this **help** message

k : same as apropos(1)

K : search **for** a string **in** all pages

t : use troff to format pages **for** printing

```

w : print location of man page(s) that would be displayed
    (if no name given: print directories that would be searched)
W : as for -w, but display filenames only
C file : use 'file' as configuration file
M path : set search path for manual pages to 'path'
P pager : use program 'pager' to display pages
S list : colon separated section list
m system : search for alternate system's man pages
p string : string tells which preprocessors to run
e - [n]eqn(1) p - pic(1)
t - tbl(1) g - grap(1)
r - refer(1) v - vgrind(1)

```

4.19 Документация, предоставляемая пакетом

Некоторые пакеты предоставляют больше документации в виде руководств (в форматах HTML, PDF и др.), файлов README и т.д. Вы можете найти ее по адресу /usr/share/doc. В Gentoo Linux документы упакованы с помощью bzip или gzip (две хорошо известные в мире Unix утилиты архивации). Чтобы просмотреть их, необходимо сперва распаковать в область на жестком диске, где имеются привилегии для записи.

Допустим, пакет zip содержит документ, объясняющий какой алгоритм использует утилита для (раз)архивирования данных:

```

$ cd /usr/share/doc/zip-*
$ ls
algorithm.txt.bz2 BUGS.bz2 CHANGES.bz2 MANUAL.bz2
README.bz2 timezone.txt.bz2 txtvsbin.txt.bz2 WHATSNEW.bz2
WHERE.bz2 ziplimit.txt.bz2
$ bunzip2 -c algorithm.txt.bz2 > ~/algorithm.txt

```

Полученный файл, algorithm.txt, можно найти в каталоге (- аббревиатура домашнего каталога текущего пользователя). В этом случае, файл является обычным текстовым файлом, который можно просмотреть с помощью команды less,

также используемой для man-страниц.

В вышеуказанном примере используется перенаправление: вывод команды (**bunzip2 -c algorithm.txt.bz2**) не показывается на экране, но перенаправляется в файл (**algorithm.txt**). Но об этом мы поговорим позже.

Также возможно чтение (сжатого) файла, используя команду **bzless**:

```
$ bzless algorithm.txt.bz2
```

4.20 Онлайн-документация

Многие проекты имеют богатую документацию. Советуем вам внимательно взглянуть на веб-сайты проектов или поискать в Интернете доступные руководства и книги. Иногда, лучшая документация может быть написана не самим проектом, но его пользователями.

Упражнения

1. Попробуйте организовать ваш домашний каталог по логическим разделам. К примеру, создайте каталоги, где вы будете хранить личные документы, изображения, документы по работе и просто временные документы.
2. В общем, системы Unix/Linux предлагают вам четыре метода архивации (сжатия). Системы Unix/Linux используют Gzip или BZip2 вместе с tar, тогда как лишь Zip — хорошо известный формат в операционных системах Microsoft Windows. Каким образом «tar» используется с gzip/bzip2? Что же такое четвертый метод архивации?
3. Как уже было упомянуто, системы Unix/Linux не имеют функциональности, позволяющей обратить изменения: когда вы удаляете файл, он удаляется. Однако, из-за особенности техники удаления файлов (фактически удаляется лишь ссылка к файлу, однако его содержимое остается на диске до перезаписи) существуют инструменты, которые могут помочь вам вернуть удаленные файлы. Попробуйте найти несколько примеров подобных утилит восстановления для Linux.

Дальнейшие ресурсы

- + Introduction to Linux, A Hands-On Guide [<http://www.tldp.org/LDP/intro-linux/html/index.html>]
- + by Machtelt Garrels (also available as PDF [<http://www.tldp.org/LDP/intro-linux/intro-linux.pdf>])

Глава 5

Файловая система Linux

Введение

Файловая система Linux имеет вид иерархически структурированного дерева, где каждая область имеет определённое значение. Структура файловой системы определена в стандарте иерархии файловой системы, описание которого вы можете найти в этой главе. Тем не менее, всё больше дистрибутивов вносит изменения в структуру своих файловых систем (но и в их содержимое), поэтому стандарту необходимо обновляться. В каком случае настройка отличается от текущего стандарта, я расскажу далее в этой главе.

Конечно, файловая система всегда хранится на носителе (будь то жесткий диск, компакт-диск, или фрагмент памяти); Каким образом эти носители связаны с ней и как Linux отслеживает их — это всё также будет разъяснено в этой главе.

5.1 Структура

Файловая система имеет древовидную структуру. Её корень, по случайному совпадению, называется корнем файловой системы, однако представляется выше всех остальных её ветвей и распознается по символу косой черты «/». Это наивысшая точка размещения, куда вы можете попасть. Тут почти всегда располагаются только каталоги:

```
~$ cd /
~$ ls -F
bin/    home/   opt/    srv/    var/
boot/   lib/    proc/   sys/
dev/    media/  root/   tmp/
etc/    mnt/    sbin/   usr/
```

Команда **ls -F** выводит содержимое корневой точки размещения, но добавляет к именам специальных файлов дополнительные символы. К примеру, добавляет к каталогам символ «/», символическим ссылкам символ «@» и исполняемым файлам символ «*». Преимущество, на данный момент в этой книге, в том, что вы сможете легко различать типы файлов, которые у вас есть. По умолчанию, для команды **ls** в Gentoo активен цветной режим, который подсказывает вам предназначение каждого файла с помощью расцветки. Тем не менее, использование обозначений с символами намного разумнее для иллюстрации в книгах.

Популярный способ представления файловой системы в виде древа. В данном примере показан верхний уровень:

```
  /      +-bin/
+-boot/   +-dev/
+-etc/    +-lib/
+-home/   +-media/
+-mnt/    +-opt/
+-proc/   +-root/
+-sbin/   +-srv/
+-sys/    +-tmp/
+-usr/    +-var/
```

Чем больше переходов, тем больше становится древо, и в скором времени станет труднее отобразить структуру в одиночном виде. Древовидный формат - всё ещё хороший способ представления файловой системы, поскольку он представляет её точно так, как вы видите.


```

/
+—      bin /
+—      ...
+—      home/
|      =—      thomas/
|      |      +—      Documents/
|      |      +—      Movies/
|      |      +—      Music/
|      |      +—      Pictures/      <— You are here
|      |      |      '—      Backgrounds/
|      |      '—      opentasks.txt
|      +—      jane/
|      '—      jack/
+—      lib /
+—      ...
'—      var /

```

Ранее мы уже освещали моменты, касающиеся навигации по древу вкратце: предположим, вы находитесь по адресу `/home/thomas/Pictures`. Чтобы пройти ещё дальше (в каталог `Backgrounds`) необходимо выполнить команду «**cd Backgrounds**». Чтобы перейти назад (в `/home/thomas`) следует выполнить «**cd ..**» (`..` обозначает родительский каталог).

Перед тем как объяснять различные размещения, давайте сперва рассмотрим, как файловая система располагается или хранится на одном носителе (или более одного)...

5.2 Монтирование файловых систем

Предположим, что корень файловой системы у вас расположен на одном разделе, а все 'пользовательские' файлы на другом. Это могло бы означать, что / всё что ниже по иерархии находится на одном разделе, за исключением точки `/home` и всего что в ней, которая расположена на втором разделе.

Выполнение монтирования требует от вас указать размещение файловой системы для точки монтирования (например, точка монтирования — это `/home`), в которой каждый файл в действительности находится в другом месте (в этом примере, всё что исходит от родительского каталога `/home`, находится на втором

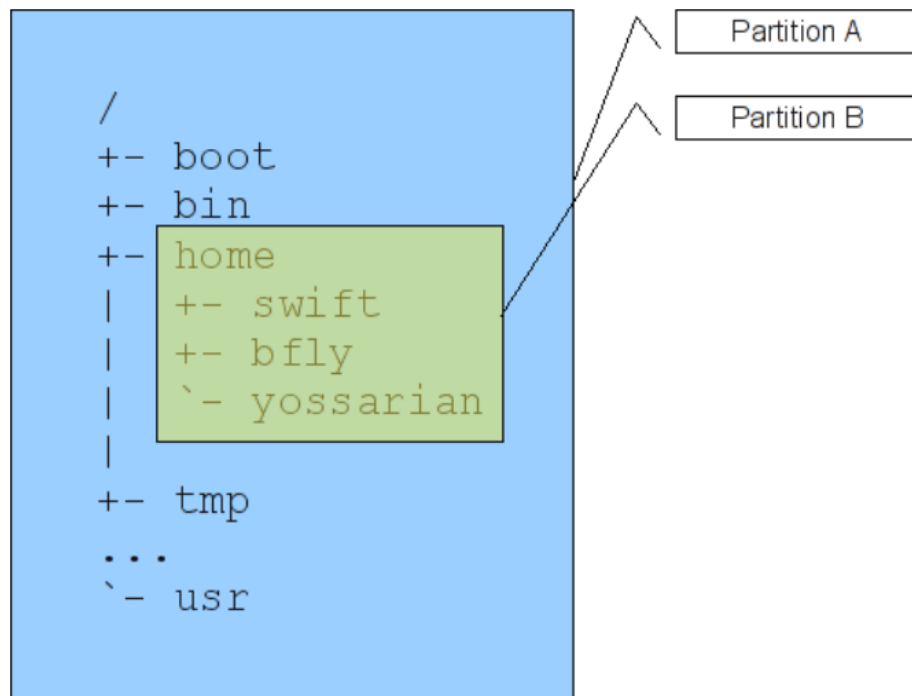


Рис. 5.1: Два раздела, используемые как части структуры файловой системы

разделе). Монтируемому разделу не нужно знать куда он примонтирован. В действительности, он не знает этого. Вы можете монтировать подкаталоги пользовательского каталога в /home (что предпочтительнее), однако лучше всего монтировать их в /srv/export/systems/remote/disk/users. Разумеется, причина по которой вы должны сделать так же — мой пример, но можно сделать это и по-своему.

Команда `mount`, запущенная без каких-либо аргументов покажет список примонтированных файловых систем:

```
$ mount
/dev/sda8 on / type ext3 (rw,noatime)
proc on /proc type proc (rw) sysfs on /sys type sysfs (rw,nosuid,nodev,noexec)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=10240k,mode= devpts
on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode
/dev/sda7 on /home type ext3 (rw,noatime)
none on /dev/shm type tmpfs (rw) /dev/sda1 on
/mnt/data type ext3 (rw,noatime)
usbfs on /proc/bus/usb type usbfs (rw,noexec,nosuid,devmode=0664,devgi
```

Вышеприведенный пример содержит большое количество разных файловых систем, о которых мы ещё ничего не знаем, и говорит нам, что файловая система может бть представлена в следующем виде:

```
/          (on /dev/sda8)
+—  ...
+—  dev/    (special: "udev")
|    +— pts (special: "devpts")
|    ‘— shm (special: "none")
+—  proc/   (special: "proc")
|    ‘— bus/
|        ‘— usb/ (special: "usbfs")

+—  sys/    (special: "sys")
+—  home/   (on /dev/sda7)
‘—  mnt/
    ‘— data/ (on /dev/sda1)
```

Игнорируя специальные точки монтирования, можно увидеть корень файловой системы на устройстве `/dev/sda8`. С другой стороны, `/home`, файловая система этой точки хранится на `/dev/sda7`, а данные `/mnt/data`, в свою очередь в `/dev/sda1`. Позже мы подробнее разберем синтаксис специфичных устройств.

Концепция монтирования позволяет программам знать, где структурированы ваши данные. С точки зрения приложения (или пользователя), файловая система является единым деревом. Под «капотом» же, она может состоять из одного раздела, дюжины разделов, сетевого хранилища, съемных носителей и др.

5.3 Файловые системы

Каждый носитель содержит файлы, которые структурированы внутренне. То, как выглядит их структура, является частью используемой файловой системы. Пользователи Windows могут вспомнить, что в самом начале, перед тем как мигрировать на одну из множества ревизий NTFS, в Microsoft Windows применялись файловые системы FAT16 и позднее FAT32. Подобно этому примеру, Linux также имеет свои файловые системы.

Однако система не требует, чтобы все разделы содержали одну из возможных файловых систем (как например в упомянутой выше системе от Microsoft «поддерживается только NTFS»): до тех пор, пока она понимает их, а используемая файловая система поддерживает такие возможности, как права владельца и разрешения, вы вольны выбирать любую из них. В действительности, множество

дистрибутивов имеет установщик, предлагающий выбрать одну из файловых систем. Следующий список — лишь небольшой, но вкратце описывающий наиболее популярные из файловых систем с перечислением их преимуществ и недостатков...

- Файловая система *ext2* — старая, но всё ещё используемая в Linux. Она представляет собой расширенную файловую систему второй версии и довольно проста. Она использовалась почти с момента появления Linux на свет и достаточно устойчива к фрагментации данных — что тем не менее, верно почти для всех файловых систем Linux. Однако медленно замещается журналируемыми.
- Файловая система *ext3* — усовершенствованная версия *ext2*, кроме других возможностей имеет концепцию журналирования.
- Файловая система *ext4* — усовершенствованная версия *ext3*, кроме всего прочего имеет поддержку очень больших систем/файлов, экстендов (смежные физические блоки), предварительного выделения, отложенного выделения и др. *Ext4* является обратно совместимой с *ext3* до тех пор, пока не используются экстенды. Неоднократно замечена как файловая система, выбираемая по умолчанию среди большинства администраторов и дистрибутивов.
- Файловая система *reiserfs* разработана с нуля. Также предоставляет журналирование, но в основном сфокусирована на производительности. Предоставляет быстрый доступ к точке размещения с тысячами файлов (*ext2* и *ext3* в подобной ситуации достаточно медленные), экономит объем занимаемого пространства для небольших файлов (некоторые другие файловые системы резервируют весь блок для каждого файла, тогда как *reiserfs* способен разделять блоки с несколькими файлами, делая их общими). Несмотря на это была популярной несколько лет назад, в те годы ей не хватало поддержки (губительные ошибки оставались незамеченными некоторое время) и она реже предлагалась дистрибутивами. Её наследница, *reiser4*, всё ещё находится на раннем этапе развития, и вследствие ареста главного разработчика Ганса Рейзера, не слишком активно разрабатывалась.
- Файловая система *btrfs* выглядит многообещающей. Нацелена на большие объемы данных, объединение нескольких устройств, снапшоты и многое другое. Несмотря на то, что в первую очередь предназначалась для использования на предприятиях, она также предлагает интересные возможности и

для домашних пользователей, такие как увеличение или уменьшение размера на лету (на уровнях файловой системы и хранилища), резервирование объектного уровня, прозрачную компрессию и клонирование.

- Файловая система *xfs* представляет собой высокопроизводительное решение с поддержкой журналирования, готовое для применения на предприятиях. Предлагает высокую параллельную пропускную способность, в связи с чем является распространенным выбором среди предприятий.
- Файловая система *zfs* является многофункциональной и предлагает контрольное суммирование на блочном уровне, компрессию, снапшоты, возможность «копирования на лету», дедупликацию, очень большие тома, удалённую репликацию и многое другое. Ранее была портирована с ОС (Open)Solaris на Linux, где и обрела своё место.

Файловая система отслеживает операции с записью файлов, производя метки (о добавлении новых файлов или изменении их содержимого) в свой журнал перед выполнением соответствующих операций. Затем выполняет запись данных на самой файловой системе, после чего удаляет метку об этом. Данный набор операций обеспечивает способность восстановить целостность файловой системы с помощью проверки журнала или удаления незавершенной записи в том случае, если операции над файловой системой внезапно прерываются (к примеру, из-за сбоя питания): таким образом она всегда находится в консистентном состоянии.

Обычно нет возможности переключаться между файловыми системами (за исключением ext2<>ext3), однако большинство из них достаточно развиты, поэтому вам не нужно паниковать по поводу выбора «правильной файловой системы».

Теперь, если мы взглянем на следующий вывод команды `mount`, то заметим, что часть строки говорит нам о том, какого «типа» точка монтирования. Следовательно, это и есть тип файловой системы, используемой для отдельного монтирования.

```
$ mount
rootfs on / type rootfs (rw)
sysfs on /sys type sysfs (rw,seclabel,relatime)
selinuxfs on /sys/fs/selinux type selinuxfs (rw,relatime)
/dev/md3 on / type ext4 (rw,seclabel,noatime,no
/dev/md4 on /srv/virt type ext4 (rw,noatime,data=ordered
proc on /proc type proc (rw,nosuid,nodev,noexec
tmpfs on /run type tmpfs (rw,rootcontext=system_
udev on /dev type devtmpfs (rw,seclabel,nosuid,relative
```

```

/dev/mapper/volgrp-nfs on /srv/nfs type ext4 (rw,noatime,data=journal
/dev/mapper/volgrp-usr on /usr type ext4 (rw,noatime,data=journal
/dev/mapper/volgrp-home on /home type ext4 (rw,noatime,nosuid,node
/dev/mapper/volgrp-opt on /opt type ext4 (rw,noatime,data=journal
/dev/mapper/volgrp-var on /var type ext4 (rw,noatime,data=journ
/dev/mapper/volgrp-vartmp on /var/portage type ext4 (rw,noatime,data=ordered
mqueue on /dev/mqueue type mqueue (rw,seclabel,nosuid,nod
devpts on /dev/pts type devpts (rw,seclabel,nosuid,noe
shm on /dev/shm type tmpfs (rw,seclabel,nosuid,nod
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec
debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec
none on /selinux type selinuxfs (rw)
tmpfs on /var/tmp type tmpfs (rw,nosuid,noexec,nodev
tmpfs on /tmp type tmpfs (rw,nosuid,noexec,nodev
rc-svcdird on /lib64/rc/init.d type tmpfs (rw,nosuid,nodev,noexec
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,nodev,noexec,nos
rpc_pipefs on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
nfsd on /proc/fs/nfsd type nfsd (rw,nodev,noexec,nosui

```

Как вы видите, все разделы (строки неспециального назначения) имеют тип `ext4`. Но что если выбрать другие файловые системы?

`Proc` — файловая система, которой не существует на устройстве, однако она является чем-то наподобие врат или шлюза доступа к ядру *Linux*. Всё, что вы видите ниже по иерархии `/proc` — это то, что ядро отображает, когда просматривается её содержимое. Это способ коммуникации с ядром (и наоборот), использующий очень простой интерфейс: чтение и запись файлов, что-то, что хорошо поддерживается.

Я расскажу вам о `/proc` больше в этой главе.

- `proc` известен как псевдофайловая система: она не содержит реальных файлов, но хранит в себе данные, касающиеся рабочих процессов.
- `sysfs` - специальная система, такая как `proc`: она не существует на устройстве, и тоже является подобием шлюза доступа к ядру *Linux*. Отличается от `proc` тем, как спроектирована и структурирована: `sysfs` более структурирована и рассчитана на компьютерный разбор файлов и директорий, тогда как `proc` имеет структуру и предназначение для более удобного их чтения или записи пользователем.

– В перспективе `proc` исчезнет (однако, ключевого срока не существует,

поскольку множество людей предпочитает простой способ с использованием `/proc`, что даёт им необходимые данные) и будет заменена файловой системой `sysfs`.

- `tmpfs` — временная файловая система, содержимое которой хранится в памяти вместо постоянного хранилища. Таким образом, она используется довольно часто (оперативная память намного быстрее даже самых производительных SSD и жестких дисков). Скажем так, обычно, `tmpfs` может выгружать страницы памяти в область подкачки, что незамедлительно делает эту часть файловой системы `tmpfs` медленней (так как перед использованием они должны считываться с диска).
 - В Linux `tmpfs` используется для таких нужд, как общая память в `/dev/shm` и `/tmp`.
- Файловая система `devtmpfs` имеет сходство с `tmpfs`, но содержит файлы устройств, управляемых ядром. Она воплощает в жизнь возможность запуска и контроля важных файлов устройств до передачи их `udev` (менеджеру устройств).
- `devpts` — еще одна псевдофайловая система, подобная `proc` и `tmpfs`. Содержит файлы устройств, используемых для эмуляции терминала (например, предоставляет доступ к консоли в графической среде при помощи `xterm`, `uterm`, `eterm` или другой программы, эмулирующей терминал). Ранее, упомянутые файлы устройств создавались статически, из-за чего большинство дистрибутивов приходилось снабжать большим количеством файлов устройств, эмулирующих терминал (поскольку было затруднительным знать, сколько таких эмуляций пользователь мог бы запустить со временем). Чтобы лучше управлять ими, была разработана псевдофайловая система, которая создает и уничтожает файлы устройств по мере необходимости.
- `usbfs` также является псевдофайловой системой и может быть сравнима с `devpts`. Также содержит файлы, которые создаются в момент подключения или извлечения устройств USB. Тем не менее, в отличие от `devpts` вместо файлов устройств она создает псевдофайлы, которые могут быть использованы для взаимодействия с устройством USB.
 - Так как большинство из них является стандартными устройствами USB (принадлежащие определенным классам, как стандартные устройства хранения USB) для Linux разработан фреймворк, позволяющий програм-

мам работать с устройствами USB, основываясь на их характеристиках посредством файловой системы `usbfs`.

- `selinuxfs` — еще одна псевдофайловая система, представляющая подсистему SELinux в ядре Linux. Используется библиотеками SELinux для взаимодействия с сервером безопасности SELinux, запрашивающий имеющиеся политики SELinux и многое другое. Эта файловая система не монтируется на системах Linux, в которых не включена подсистема SELinux.
- `mqueue` — псевдофайловая система, используемая для поддержки межпроцессных сообщений (очередь сообщений POSIX)
- `binfmt_misc` — псевдофайловая система, используемая для регистрации исполняемых файлов. Посредством `binfmt` ядро Linux способно выполнять произвольные форматы исполняемых файлов, распознавая зарегистрированные исполняемые форматы и передавая их приложениям, находящимся в пользовательском пространстве.

Существует множество специальных файловых систем (некоторые из них даже упомянуты выше в выводе команде `mount`), однако я оставлю это для интересующегося читателя, который найдет больше информации об этих файловых системах.

5.4 Разделы и диски

Каждое устройство, подключенное к оборудованию (кроме сетевого интерфейса) и доступное в Linux, представляется в виде файлов устройств, размещенных в `/dev`. Разделы и диски — не исключение. Возьмем к примеру жёсткий диск Serial ATA.

Драйвер диска SATA использует внутренний слой SCSI для отображения и доступа к данным. Собственно, устройство SATA представляется как устройство SCSI. Первый диск SATA в вашей системе отображается как устройство `/dev/sda`, а его первый раздел станет `/dev/sda1`. Вы можете расшифровать `sda1` как «1-ый раздел (1) на первом (a) устройстве scsi (sd)».

```
~$ ls -l /dev/sda1
brw-rw— 1 root disk 8, 1 Nov 12 10:10 /dev/sda1
```


Обычный диск типа ATA (или DVD-ROM) отобразится как `/dev/hda` (сокращение `hd` означало жёсткий диск, но теперь применяется для распознавания устройства ATA).

```
$ ls -l /dev/hda  
brw-rw---- 1 root cdrom 3, 0 Apr 23 21:00 /dev/hda
```

В стандартной инсталляции Gentoo, менеджер устройств (называется `udev`) создает файлы устройств, обнаруженных оборудованием. К примеру, в моей системе, разделы первого устройства SATA могут быть представлены в следующем виде:

```
$ ls -l /dev/sda*  
brw-r----- 1 root disk 8, 0 Sep 30 18:11 /dev/sda  
brw-r----- 1 root disk 8, 1 Sep 30 18:11 /dev/sda1  
brw-r----- 1 root disk 8, 2 Sep 30 18:11 /dev/sda2  
brw-r----- 1 root disk 8, 5 Sep 30 18:11 /dev/sda5  
brw-r----- 1 root disk 8, 6 Sep 30 18:11 /dev/sda6  
brw-r----- 1 root disk 8, 7 Sep 30 18:11 /dev/sda7  
brw-r----- 1 root disk 8, 8 Sep 30 18:11 /dev/sda8
```

Внутри каталога `/dev` также находятся символические ссылки (указатели) на файлы устройств, которые могут использоваться для идентификации разделов или дисков другим способом. К примеру, чтобы отобразить список блочных устройств, используя их UUID (Универсальный Уникальный Идентификатор):

```
$ ls -l /dev/disk/by-uuid
total 0
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 1628b93d-3448-4b8c-b72b-1d68e89bd2fa
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 5550c45f-9660-44f2-8e86-05a612d028a
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 77eb40be-f571-49c6-bbb0-a12677615fe
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 9644f675-6eaf-4974-9e1a-0b8eafa931a
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 9beda062-6e15-4323-9ad1-53b6a9e3967
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 9e7a1178-b0ad-4cd8-8977-a471a5d2b79
lrwxrwxrwx. 1 root root 9 Dec 16 20:01 b06fa545-0d5a-4c9a-97cb-83b4e1799f9a
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 b80c76a3-d52f-4006-9bf5-62f4d7edc79
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 bdb15de1-3430-47b4-9e63-ee58557f1d1
lrwxrwxrwx. 1 root root 9 Dec 16 20:01 c44503ce-e52e-452c-b4bc-767ddd1d3b27
lrwxrwxrwx. 1 root root 9 Dec 16 20:01 d8a2bb27-15da-49bb-b205-3160c307835c
```

Преимущество использования этих UUID заключается в своеобразной идентификации раздела или диска. Если диски являются многофункциональными или мы говорим о съёмных устройствах, то используя UUID мы можем знать, что смотрим на правильный раздел (и не на другой диск, который, например, назван /dev/sda2).

5.5 Команда 'mount' и файл fstab

Монтирование носителя с файловой системой выполняется командой `mount`. Чтобы производить это действие достаточно хорошо, необходима некоторая информация, такая как точка монтирования, тип файловой системы, устройства и дополнительные параметры монтирования.

К примеру, команда `mount`, которая монтирует устройство /dev/sda7, содержащее файловую систему `ext3` в точку /home, выглядит так:

```
# mount -t ext3 /dev/sda7 /home
```

Также можно увидеть, что это действие напоминает «прикрепление» определённого устройства к какой-либо области файловой системы, что эффективно расширяет её большим количеством файлов, директорий и прочих данных.

Однако, если в системе имеется несколько разных разделов, необходимость

вводить команды снова и снова может показаться глупой шуткой. Это одно из причин наличия в Linux специального файла, содержащего список файловых систем. Называется он `/etc/fstab` и содержит всю информацию, которая может быть необходима, чтобы успешно монтировать устройство. Пример содержимого файла `fstab` приведен ниже:

```
/dev/sda8 /          ext4    defaults ,noatime    0 0
/dev/sda5 none      swap    sw                0 0
/dev/sda6 /boot      ext4    noauto ,noatime      0 0
/dev/sda7 /home      ext4    defaults ,noatime    0 0
/dev/sdb1 /media/usb  auto    user ,noauto ,gid=users 0 0
```

Структура файла следующая:

1. Устройство, предназначенное для монтирования (также поддерживаются метки, но об этом мы поговорим позже)
2. Размещение монтируемого устройства (точка монтирования)
3. Тип файловой системы или `auto` (автоматический), если вы хотите, чтобы Linux автоматически определял её
4. Дополнительные параметры (используйте «`defaults`», если не нуждаетесь в какой-либо специфичной опции), такие как «`noatime`» (не регистрировать время доступа к файловой системе для улучшения её производительности) и «`users`» (позволять обычным пользователям монтировать или размонтировать устройство)
5. Номер дампа (можно оставить 0)
6. Порядок проверки файлов (также можно оставить 0)

Благодаря этому файлу, предыдущий пример с командой `mount` больше не требуется (так как монтирование выполняется автоматически), однако в том случае, если операция монтирования ещё не выполнена, команда упрощается до:

```
# mount /home
```

Если вам когда-либо потребуется удалить носитель из системы, используйте команду `umount`:

```
# umount /home
```

Это представляет особый интерес в случае со съёмными носителями: если необходимо получить доступ к носителю CD или DVD (или даже к USB-накопителю), перед этим сперва следует монтировать носитель с файловой системой. Аналогичным образом, перед удалением устройства из системы, его сперва необходимо отмонтировать:

```
# mount /media/dvd
```

(Устройство DVD теперь примонтировано и доступно)

```
# umount /media/dvd
```

(Устройство DVD теперь более не доступно в файловой системе и может быть извлечено из лотка).

Разумеется, современные операционные системы Linux снабжены инструментами, которые автоматически монтируют устройства для доступа по файловой системе и отмонтируют при их извлечении. По умолчанию, Gentoo Linux не предлагает их (их следует установить самостоятельно).

5.6 Размещение раздела подкачки

У вас есть (или возможно будет) раздел, выделенный специально для страничной подкачки: Linux использует этот раздел, в нем при недостатке физической памяти хранятся данные обо всех запущенных процессах (и их ресурсах). Когда это происходит, операционная система начинает помещать эти данные (которые, возможно, не будут использованы в скором времени) на диск, освобождая физическую память.

Этот раздел такой же как и другие, однако вместо обычной используемой пользователями файловой системы, он содержит свою особенную для нужд памяти и опознается в таблице разделов как раздел подкачки.

```
# fdisk -l /dev/sda
```

```
Disk /dev/sda: 60.0 GB, 60011642880 bytes
```

255 heads, 63 sectors/track, 7296 cylinders
 Units = cylinders of 16065 * 512 = 8225280 bytes
 Disk identifier: 0x8504eb57

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1*		1	1275	10241406	83	Linux
/dev/sda2		1276	7296	48363682+	5	Extended
/dev/sda5		1276	1525	2008093+	82	Linux swap / Solaris
/dev/sda6		1526	1532	56196	83	Linux
/dev/sda7		1533	2778	10008463+	83	Linux
/dev/sda8		2779	7296	36290803+	83	Linux

Раздел подкачки указан в файле `/etc/fstab` и задействован при загрузке.

Чтобы просмотреть текущие активные разделы подкачки (или файлы, файлы подкачки также поддерживаются), выведите содержимое файла `/proc/swaps` на экран или запустите команду `swapon -s` :

```
# cat /proc/swaps
```

Filename	Type	Size	Used	Pri
/dev/sda1	partition	1048572	108	-1

5.7 Размещение файловых систем Linux

Как было сказано ранее, каждое размещение (или область) файловой системы Linux имеет свое назначение. Мы уже рассказывали об одном из них, без подробностей о том, что они являются стандартными размещениями, такими как `/home`, в котором хранятся каталоги локального пользователя. Стандарт файловой системы Linux покрывает все эти области, однако эта глава была бы неполной, если бы мы не упомянули их.

5.7.1 Системные размещения

Системные размещения представляют собой области, которые нельзя поместить на другой носитель с файловой системой, так как они требуются самой команде `mount` для корректного функционирования:

- `/bin` обычно содержит исполняемые программы, необходимые для инициализации и запуска системы. Однако в последнее время, всё больше и больше дистрибутивов перемещает все свои приложения в `/usr/bin` и использует символические ссылки для перехода в новую структуру.
- `etc` содержит все конфигурационные файлы системы (кроме специфичных для пользователя)
- `/lib` обычно содержит системные библиотеки, необходимые для успешной загрузки системы и выполнения команд, находящихся в `/bin`. Недавно они также были перемещены в `/usr/lib`.
- `/sbin` подобно `/bin` содержит исполняемые программы. Тем не менее, если `/bin` хранит программы, которые могут использовать даже пользователи, то `/sbin` содержит программы, предназначенные для административных нужд.

5.7.2 Пользовательские размещения

Пользовательские размещения — области, которые содержат файлы, используемые в обычных операциях в рабочей системе (такие как данные приложений и сами приложения). Они могут размещаться на отдельном носителе, и если это необходимо, потребуется настроить так называемый «начальный диск оперативной памяти» для загрузки системы. Подробнее на этом мы остановимся позже. Областью пользовательских данных является `/usr` (который достался нам от системных ресурсов Unix).

- `/usr` — корень всех пользовательских размещений (и обычно точка монтирования для отдельного носителя)
- `/usr/X11R6` содержит все файлы, необходимые для графического оконного сервера (X11); они разделены на двоичные файлы (`bin/`), библиотеки (`lib/`) и определения для заголовков (`/include`) для программ, рассчитанных на использование в системе X11.
- `/usr/bin` содержит все исполняемые файлы программ
- `/usr/lib` содержит все библиотеки для вышеупомянутых программ
- `/usr/share` содержит полные данные различных приложений (таких как графические элементы, документация, ...)

- /usr/local чаще всего может быть отдельной точкой монтирования, также содержит программы, специфичные для локальной системы (/usr может использоваться совместно различными системами в больших окружениях)
- /usr/sbin, аналогично /usr/bin, является местом для исполняемых файлов программ, но как /bin и /sbin, содержит программы, необходимые для административных нужд системы.

5.7.3 Общие размещения

Общие размещения — это всё что может размещаться на отдельном носителе...

- /home содержит каталоги всех локальных пользователей
- /boot содержит статические файлы, относящиеся к загрузке, обычно после загрузки системы они уже не требуются (к примеру, сюда включены конфигурация загрузчика и образ ядра)
- /media содержит точки монтирования для различных подключаемых устройств хранения (как диски USB, DVD, ...)
- /mnt — размещение для временных точек монтирования (читайте: те, которые не стоит указывать в файле fstab)
- /opt содержит дополнительные пакеты и обычно используется для установки приложений, которые не поддерживаются пакетным менеджером (как те, находящиеся в /usr) или сборки специфичных для системы приложений (/usr/local).
- /tmp содержит временные файлы системных утилит. Каталог может быть очищен при загрузке.
- /var содержит данные, размеры которых постоянно меняются. Это могут быть журнальные файлы (логи), кэш приложений и т.д.

5.8 Специальные файловые системы, предоставляемые ядром

Некоторые области файловой системы не хранятся на диске или разделе, но создаются и управляются на лету ядром Linux.

- `/proc` содержит информацию о запущенной системе, ядре и процессах
- `/sys` содержит сведения о доступном оборудовании и задачах ядра
- `/dev` содержит файлы устройств

Нередко в этих областях содержатся другие примонтированные (псевдо) файловые системы.

5.9 Корневая файловая система /

Как упомянуто ранее, корневая файловая система «/» является родительской для всего её содержимого. Это первое, что монтируется когда ядро загружается (если только не используется начальный диск оперативной памяти), и система не сможет правильно функционировать, если ядро обнаружит повреждения в этой файловой системе. Также, из-за особенности процесса загрузки, она обычно доступна для записи (так как в процессе загрузки необходимо хранить информацию о состоянии запуска каких-либо процессов и т.п.). Некоторые её области настоятельно рекомендуется располагать в корневой файловой системе (никогда не монтируйте другую файловую систему поверх этой области). Вот они:

- `/bin` и `/sbin` содержат двоичные файлы (команды) или ссылки на двоичные файлы, которые необходимы, чтобы приготовить систему к тому состоянию, когда она сможет монтировать другие файловые системы. Несмотря на то, что этой функциональности с каждым разом становится всё меньше, она всё ещё может сломать систему, если вы создадите отдельные точки монтирования для этих (небольших) областей.
- `/lib` содержит библиотеки, необходимые для команд `/bin`.
- `/etc` содержит конфигурационные файлы системы, которые не требуются в процессе загрузки системы.

Яркий пример конфигурационного файла, находящегося в `/etc` — `fstab` (который содержит информацию о других файловых системах для их монтирования при загрузке).

5.10 Область произвольных данных /var

Размещение `var` содержит различные данные. Как вы должны заметить, это место используется довольно часто на протяжении всего жизненного цикла инсталляции. Оно содержит журнальные файлы, кэши данных, временные файлы и т.д. Для многих это повод дать /var отдельную файловую систему: её использование гарантирует, что переполнение /var не нанесет вред корневой файловой системе (поскольку она находится на другом разделе).

5.11 Пользовательская область /usr

Каталог `usr` содержит файлы приложений для ежедневного пользования. Его особое свойство заключается в том, что если вы не обновляете систему, он останется неизменным. Другими словами, у вас должен быть доступ к этой области «только для чтения». Большинство дистрибутивов, однако, больше не поддерживает эту возможность и предусматривает, что /usr доступен для записи администратором всё время. Размещение /usr на другой файловой системе также даёт и другие преимущества (и всё равно кое-что может быть надуманным ;-). Если вы выполняете административные задачи, можно отмонтировать /usr, что не позволит пользователям запускать любые программы во время административного окна. Размещая /usr (и некоторые другие точки монтирования) на отдельном носителе, можно содержать корневую файловую систему в меньшем объеме, что снижает риск её повреждения, которое могло бы сделать невозможной загрузку системы. Вы можете использовать файловую систему, оптимизированную для быстрого считывания файлов (запись не требует особой отзывчивости). Преимуществ данного решения в настоящее время становится всё меньше. Вместо этого, дистрибутивы фокусируются на использовании начальных файловых систем в оперативной памяти (миниатюрная файловая система в памяти, используемая для запуска системы), о которых мы поговорим позже в этой книге.

5.12 Домашний каталог /home

Наконец, каталог /home. Он содержит каталоги пользователей. Внутри них пользователи имеют полный доступ на запись. За ними у пользователей есть права только для доступа в режиме чтения (или даже полностью отсутствуют). Структура внутри этого каталога также не привязана к каким-либо конкретным правилам. Поэтому пользовательский каталог /home находится под ответственностью

самих пользователей. Однако это также означает, что пользователи станут наполнять этот каталог по своему усмотрению, и если он находится не на отдельном разделе, может произойти переполнение корневой файловой системы. Поэтому, хорошая идея - использовать отдельную файловую систему под /home. Другое преимущество использования отдельного раздела для /home можно извлечь в случае, когда вы решаете переключаться между дистрибутивами: можно повторно использовать /home для других дистрибутивов Linux (или после переустановки вашего дистрибутива).

5.13 Разрешения и атрибуты

По умолчанию, Linux поддерживает систему разрешений под названием дискреционное управление доступом (DAC), привилегии которой базируются на владельце файла и распознавании пользователя. Тем не менее, существующие проекты предоставили мандатное управление доступом (MAC) для Linux, его привилегии основаны ролях и используется там, где администратор может усилить политики безопасности по файлам и процессам. Большинство проектов MAC, нацеленных на безопасность (такие как RSBAC [<http://www.rsbac.org>], LIDS [<http://www.lids.org>] и grSecurity [<http://www.grsecurity.net>]) пока не являются частью стандартного ядра Linux. Я расскажу вам о стандарте и механизме дискреционного управления доступом используется в большинстве дистрибутивов Linux. Мы также не обсудили SELinux [<http://www.nsa.gov/selinux>], который является часть ядра. Если вы интересуетесь системами, усиленными SELinux, советую использовать Gentoo Hardened [<http://www.gentoo.org/proj/en/hardened>], поддерживающий SELinux. Существует также настольная книга Gentoo Hardened [<http://www.gentoo.org/proj/en/hardened/selinux/selinux-handbook.xml>], которую стоит прочесть.

5.14 Чтение, запись и исполнение

Файловая система Linux поддерживает различные флаги разрешения для каждого файла или каталога. Флаг следует рассматривать как возможность или привилегию, которая может быть активна или неактивна и устанавливаться отдельно от других флагов. Большинство из используемых флагов в файловой системе — на чтение (r), запись (w) и исполнение. Их свойства отличаются в зависимости от объекта, к которому они применяются. Однако поддержка этих флагов не делает систему защищённой: вы собираетесь комбинировать эти привилегии, ос-

новываясь на том, кто работает с этим файлом. К примеру, конфигурационные файлы системы должны быть читаемы только администратором (или администраторами), некоторые из них могут быть нечитаемыми даже для пользователя (например, файл, содержащий пользовательские пароли). Для доступа к этому Linux поддерживает своего рода привилегии трех типов назначения:

- владелец файла (1-ая группа привилегий)
- группа, владеющая файлом (2-ая группа привилегий)
- все остальные (3-ая группа привилегий)

Здесь вы можете установить один набор привилегий для владельца файла, другой набор для группы (что означает любой, кто является членом этой группы, к которой применяются эти привилегии) и третий для всех остальных. В случае с файлом,

- привилегия read (чтение) информирует систему о том, что файл можно прочитать (просмотреть)
- привилегия write (запись) информирует систему о том, что файл можно записать (отредактировать)
- привилегия execute (исполнение) информирует систему о том, что файл является командой, которую можно выполнить

Взгляните на пример с выводом команды `ls -l`:

```
$ ls -l /etc/fstab
-rw-r—r— 1 root root 905 Nov 21 09:10 /etc/fstab
```

В приведенном выше примере, файл `fstab` доступен для записи пользователем `root` (`rw-`) и просмотра любым другим пользователем (`r-`).

В случае с каталогом,

- привилегия read (чтение) информирует систему о том, что содержимое каталога можно прочитать (просмотреть)
- привилегия write (запись) информирует систему о том, что содержимое каталога можно изменить (добавлять или удалять файлы или подкаталоги)
- привилегия execute (исполнение) информирует систему о том, что можно перейти в каталог (используя команду `cd`)

Взгляните на пример с выводом команды `ls -ld`:

```
$ ls -ld /etc/cron.daily
drwxr-x— 2 root root 4096 Nov 26 18:17 /etc/cron.daily/
```

В приведенном выше примере, каталог `cron.daily` доступен для просмотра (r), записи (w) и входа (x) пользователем `root`. Члены группы `root` имеют права на чтение и вход (r-x), тогда как все остальные пользователи не имеют прав для просмотра, записи и входа в каталог (—) Привилегия на просмотр Чтобы просмотреть доступные привилегии для файла, можно использовать поддержку длинного листинга команды `ls`. К примеру, чтобы просмотреть права системного файла `passwd` (который содержит сведения об учетной записи пользователя), выполните:

```
$ ls -l /etc/passwd
-rw-r—r— 1 root root 3108 Dec 26 14:41 /etc/passwd
```

Файл имеет разрешения на чтение/запись для пользователя `root` и на чтение для всех остальных. Первый символ в этом выводе отображает тип файла:

- `'-'`: обычный файл `'d'`: каталог `'l'`: символическая ссылка `'b'`: блочное устройство (как `/dev/sda1`) `'c'`: символьное устройство (как `/dev/console`) `'p'`: именованный канал `'s'` сокет доменов `unix`

Отображаемая информация о привилегиях разделена на три части: одна для владельца файла, другая для группы, владеющей файлом и последняя для всех остальных. Таким образом в данном примере, мы можем интерпретировать вывод `'-rw-r—r—'` как:

1. файл является регулярным
2. владелец (`root` — см. третье поле в выводе) имеет права на чтение и запись
3. члены владеющей группы (также `root` — см. четвертое поле в выводе) имеет права на чтение
4. любой пользователь имеет права на чтение

Другим примером может стать каталог `/var/log/sandbox`. В том случае, мы также используем аргумент `ls -ld`, что вывести сведения о каталоге вместо вывода его содержимого:

```
$ ls -ld /var/log/sandbox
drwxrwx— 2 root portage 4096 Jul 14 18:47 /var/log/sandbox
```

В этом случае:

1. файл является каталогом
2. владелец (root) имеет права для чтения, записи и запуска
3. члены владеющей группы (portage) также имеют права для чтения, записи и запуска
4. остальные пользователи не имеют никаких прав и не могут что-либо совершать.

Другим способом получить сведения о правах доступа является использование команды stat:

```
$ stat /etc/passwd
File:          /etc/passwd'
Size:          3678 Blocks: 8
IO Block:      4096 regular file
Device:        808h/2056d
Inode: 3984335
Links: 1
Access:        (0644/-rw-r--r--) Uid: ( 0/ root)
Gid:           ( 0/ root)
Access: 2013-03-18 21:46:06.000000000 +0100
Modify: 2013-03-18 21:46:06.000000000 +0100
Change: 2013-03-18 21:46:06.000000000 +0100
```

В выводе команды stat можно заметить не только такие же флаги доступа, которые упоминались выше (в этом случае -rw-r--r-), но и число. Число определяет такие же права в сокращенном обозначении. Чтобы уметь читать и понимать значение этого числа, необходимо знать верные значения каждого из чисел:

- права на исполнения обозначают число 1
- права на запись обозначают число 2
- права на чтение обозначают число 4

Чтобы получить права доступа отдельной группы (владельца, группы или остальных), добавьте числа вместе. Для файла с привилегиями (-rw-r--r-) это выведет номер 644:

- $6 = 4 + 2$, означает права для чтения и записи владельца

- 4 = 4, означает права для чтения группы
- 4 = 4, означает права для чтения для остальных пользователей

Первое число 0, как мы заметили в выводе `stat`, обозначает файл, не имеющий особых привилегий.

5.15 Особые привилегии

Кроме того, существует несколько особых привилегий внутри Linux. Это флаг ограничивающий удаление или бит закрепления (sticky bit), опознанные ранее. Если задать его каталогу, это предотвратит доступ на запись в него другими пользователями, однако не для удаления файла (по умолчанию доступ на запись для каталога означает, что вы можете удалить файлы, находящиеся внутри него в зависимости от их разрешений). Наиболее известный пример использования этого флага для каталога `/tmp`:

```
$ stat /tmp
File:          '/tmp'
Size:          28672
Blocks:        56 IO
Block:         4096 directory
Device:        808h/2056d
Inode:         3096577
Links:         759
Access:        (1777/drwxrwxrwt)
Uid:           ( 0/ root)
Gid:           ( 0/ root)
Access:        2010-01-10 17:44:04.000000000 +0100
Modify:        2013-04-02 00:04:36.000000000 +0200
Change:        2013-04-02 00:04:36.000000000 +0200
```

Другая особая привилегия, которую мы опознали ранее — флаг `setuid` или `setgid`. При применении этого флага к файлу (не сценарию!), он становится исполняемым и выполняется с правами владельца (`setuid`) или владеющей группы (`setgid`) вместо выполнения с правами пользователя, запустившего файл. Это означает, что пользователи, не имеющие привилегий суперпользователя всё ещё могут запускать команды с повышенными (от `root`) привилегиями в том случае, если эти команды имеют флаг `setgid`. По этой причине число исполняемых файлов, имеющих бит `setuid/setgid` должно быть ограничено и проверяться на воз-

можные риски нарушения безопасности. Прекрасный пример с данным флагом приведен для `/bin/mount`:

```
$ stat /bin/mount
File:  ‘ /bin/mount’
Size:  59688 Blocks: 128
IO Block:          4096 regular file
Device:           808h/2056d Inode: 262481
Links:    1
Access:   (4711/-rws—x—x)
Uid:      ( 0/ root)
Gid:      ( 0/ root)
```

5.15.1 Изменение привилегий

Чтобы изменить привилегии файла или каталога, следует использовать команду `chmod` (change mode). Её синтаксис достаточно прост, чтобы хорошо запомнить. Итак, ключевые разрешения:

- ‘u’ для пользователя,
- ‘g’ для группы,
- ‘o’ для остальных пользователей (прочих)

Затем, вы можете установить (=), добавить (+) или удалить (-) привилегии. К примеру, сделать файл `/etc/passwd` доступным для записи членами владеющей группы:

```
# chmod g+w /etc/passwd
```

Вы также можете комбинировать привилегии. Например, если хотите удалить привилегии для записи владеющей группы и удалить привилегии для чтения у других:

```
# chmod g-w,o-r /etc/passwd
```

Наконец, вы можете также использовать цифровой индекс, если желаете:

```
# chmod 644 /etc/passwd
```

5.15.2 Изменение (смена) владельца

Когда вам необходимо сменить владельца файла или каталога, используйте команду `chown` (change owner) или `chgrp` (change group). Например, чтобы сменить владельца файла на пользователя «jack», выполните следующее:

```
# chown jack template.txt
```

Чтобы сменить владельца файла, необходимо стать `root` (применить привилегии суперпользователя) — это не поможет, если вы уже являетесь текущим владельцем. Однако, это правило не распространяется для группы: если вы член группы назначения, то можете изменить владеющую группу:

```
$ ls -l bar
-rw-r--r-- 1 swift users 0 May 13 20:41 bar
$ chgrp dialout bar
$ ls -l bar
-rw-r--r-- 1 swift dialout 0 May 13 20:41 bar
```

Если же необходимо сменить и владельца и группу, можно использовать лишь одну команду `chown`: просто разделите владельца и группу назначения двоеточием, как тут:

```
# chown jack:dialout template.txt
```

5.16 Атрибуты

Некоторые файловые системы позволяют применять к файлам дополнительные атрибуты. Эти атрибуты могут иметь воздействие на разрешения или использование этих файлов либо на то, как операционная система работает с этими файлами. Их использует немного дистрибутивов, так как не все файловые системы их поддерживают.

5.16.1 Листинг или изменение атрибутов

Чтобы просмотреть атрибуты файла, можно использовать команду `lsattr` (list attributes); а для их изменения применить команды `chattr` (change attributes). Дистрибутив *Gentoo* не имеет образца файла для примера, поэтому давайте сперва создадим его:

```
# touch /tmp/foo
# chattr +aS /tmp/foo
```


Теперь давайте посмотрим, что "скажет" `lsattr`:

```
# lsattr /tmp/foo  
s-S—a———— /tmp/foo
```

Ничего удивительного, в выводе присутствует та самая команда `chattr`. Но что он означает? Хорошо, `man chattr` даёт нам необходимую информацию, но здесь она сокращена:

- `s`: когда файл удалён, его блоки обнуляются и записываются обратно на диск (в отличие от примера с обычными файлами, где присутствует только ссылка на удалённый файл)
- `S`: Когда изменения применяются к файлу, они немедленно синхронизируются на диск (кэширование памяти не допускается)
- `a`: Файл может быть только прикрепляемым (данные добавляются к файлу); применение изменений к текущему содержанию не допускается. Очень полезно для журнальных файлов (логов).

Другой, весьма интересный атрибут — неизменяемый флаг (`immutable flag`). Он не позволяет удалять, изменять, модифицировать, переименовывать или переносить файл.

5.16.2 Элементы POSIX ACL

Следующей ступенью является дискреционное управление доступом для файлов в Linux (пользователь, группа и другие). Возможно добавить к файлам больше элементов контроля доступа через POSIX ACL. С помощью команды `getfacl` можно отобразить элементы доступа к файлу или каталогу вместе с элементами управления POSIX (если это применимо). Команда `setfacl` используется для их добавления или удаления из файла или каталога. К примеру, чтобы дать пользователю `minidlna` минимальный доступ к файлу, который изначально не имел его: Поддержка элементов POSIX ACL, в свою очередь, требует наличия поддержки специфичной файловой системы, тем самым, её возможно придется включить в ядре. Кроме того, она должна быть смонтирована с параметром «`acl`».

5.16.3 Стандартные расширенные атрибуты

Файлы могут иметь дополнительные расширенные атрибуты, присвоенные им. POSIX ACL, например, использует расширенные атрибуты под названием

system.posix_acl_access, с помощью которых SELinux использует расширенный атрибут по имени *security.selinux*. Расширенные атрибуты являются метаданными, присвоенными файлам, которые используются одним или более приложениями для отдельной функции. Когда расширенный атрибут находится в безопасном пространстве имён (его имя начинается на «security.»), он является атрибутом, особо важным с точки зрения безопасности и может быть модифицирован лишь администраторами или пользователями с корректными привилегиями. Чтобы отобразить список всех расширенных атрибутов, присвоенных файлу, используйте *getfattr*:

```
$ getfattr -m . -d TMPFILE
# file: home/swift/TMPFILE
security.selinux="staff_u:object_r:user_home_t"
```

Обычно пользователям не нужно изменять расширенные атрибуты напрямую; вместо этого за ними следит приложение или приложения, которые их поддерживают. Но всё ещё остается возможность изменять эти атрибуты напрямую (повторюсь ещё раз, если у вас имеются корректные разрешения), используя *setfattr*.

5.17 Обнаружение файлов

Когда в наличии множество областей с данными, может быть затруднительным найти отдельный файл. Большую часть времени вы желаете найти файл в домашнем каталоге (поскольку это единственное место, где у вас есть привилегии на запись). Однако в некоторых случаях, может быть необходимо найти (обнаружить) файл, расположенный где-либо в недрах системы. К счастью, для этих нужд существует несколько команд.

mlocate

Команда *locate* управляет и использует базу данных файлов для помощи в поиске конкретного файла. Перед тем как использовать *locate*, сперва необходимо установить её (пакет называется *sys-apps/mlocate*), а затем создать базу данных. Кроме того, поскольку она не обновляется автоматически при каждом внесении изменений в систему, вам необходимо запускать эту команду всякий раз, а после запускать:

```
# updatedb
```

Популярный способ поддерживать базу данных в актуальном состоянии — это использовать системный планировщик (называется cron), о котором мы упоминали ранее. Когда база данных построена и что-то в ней обновилось, можно установить местонахождение файла в файловой системе, используя утилиту locate:

```
# locate make.conf
/etc/portage/make.conf
(...)
/usr/portage/local/layman/make.conf
```

Как видите, команда locate возвращает список всех файлов, которые она нашла, где строка используется для имени файла, даже если оно отличается. Название mlocate принадлежит проекту, который сопровождает пакет. Ранее по историческим причинам, пользователи выбирали пакет slocate в качестве инструмента с функциональностью locate.

5.17.1 find

Команда find самая значимая и мощная. В отличие от locate, она возвращает лишь актуальную информацию (тем самым, не использует базу данных). Это делают поиск посредством find несколько медленным, однако его мощь заключается не в скорости, а в параметрах для поиска файла.

5.17.2 Регулярные паттерны find

Наиболее простая конструкция find обнаруживает отдельный файл в одном или более каталогах. К примеру, чтобы найти файлы или подкаталоги с именем dhcpd.conf внутри каталога /etc (точные совпадения), выполните:

```
$ find /etc -name dhcpd.conf
/etc/dhcp/dhcpd.conf
```

Чтобы найти файлы (не каталоги) с выражением dhcpd в имени файла, также в каталоге /etc выполните:

```
$ find /etc -type f -name '*dhcpd*'
/etc/conf.d/dhcpd
/etc/init.d/dhcpd
/etc/udhcpd.conf
/etc/dhcp/dhcpd.conf
```

Чтобы найти файлы в каталоге /etc, которые были модифицированы за последние 7 дней (то есть: «меньше чем 7 дней назад»):

```
$ find /etc -type f -mtime -7  
/etc/mtab  
/etc/adjtime  
/etc/wifi-radar.conf  
/etc/genkernel.conf
```

Вы даже можете найти файлы, основываясь на их владельце. К примеру, найти файлы в каталоге /etc, которые не принадлежат пользователю root:

```
$ find /etc -type f -not -user root
```

5.17.3 Комбинирование паттернов find

Также можно комбинировать паттерны find. К примеру, найти файлы, модифицированные в течение последних 7 дней, но не содержащие расширения .conf в своём имени:

```
$ find /etc -type f -mtime -7 -not -name '*.conf'  
/etc/mtab  
/etc/adjtime
```

Или найти те же файлы, но их имена не должны содержать слово mtab:

```
$ find /etc -type f -mtime -7 -not \( -name '*.conf' -or -name mtab )  
/etc/adjtime
```

5.17.4 Работа с результатом

С помощью find вы также можете выполнять задачи с результатом. К примеру, если необходимо просмотреть вывод «ls -l» для файлов, найденных командой find, можно добавить параметр -exec. Строка после -exec должна содержать последовательности двух специальных символов:

- ' ' представляет файл, найденный командой find. Команда, которой передан параметр -exec, выполняется, и ' ' подставляется файлом.
- завершает команду при условии -exec.

```
$ find /etc -type f -mtime -7 -exec ls -l '{}' \;
```

В Интернете также можно обнаружить и следующую конструкцию:

```
$ find /etc -type f -mtime -7 | xargs ls -l '{}'
```

Она даёт тот же результат, но её поведение немного отличается. При использовании параметра `-exec` команда `find` выполняется для каждого файла, которого встретит. Конструкция `xargs` попытается исполнить команду как можно скорее, основываясь на ограничениях аргумента. К примеру, если команда `find` возвращает 10000 файлов, то с параметром `-exec` она исполняется 10000 раз единожды за файл. С помощью `xargs`, эта команда может быть исполнена только несколько десятков раз. Это возможно, так как `xargs` применяет одну команду для нескольких файлов, таким образом, предположим, что она может справиться с несколькими файлами. Пример запуска `find -exec`:

```
ls -l file1
ls -l file2
...
ls -l file10000
```

Пример запуска с `xargs`:

```
ls -l file1 file2 ... file4210
ls -l file4211 file4212 ... file9172
ls -l file9173 file9174 ... file10000
```

Упражнения

1. Создайте иерархию каталогов где-нибудь во временной области, где вы можете производить запись, например в подкаталоге `tmp` домашнего каталога, следующим образом:

```
$ mkdir -p tmp/test/to/work/with/privileges
```

Теперь, рекурсивно удалите (снимите) привилегии с любого пользователя (кроме владельца или группы) внутри этой структуры.

2. Проверьте привилегии каталога `/tmp`. Как вы установили разрешения для собственного каталога `tmp`?