Дэниэл Роббинс.

Linux Fundamentals

http://zabika.ru/adpopac/Linux+Fundamentals+(%D0%9F%D0%BE%D1%81%D0%BE%D0%B1%D0%B8%D0%B5+%D0%B4%D0%BB%D1%8F+%D0%BF%D0%BE%D0%B4%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%BA%D0%B8+%D0%BA+%D1%8D%D0%BA%D0%B7%D0%B0%D0%BC%D0%B5%D0%BD%D1%83+lpic+1)+%D0%9F%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%D0%B4%3A+%D0%91%D1%83%D0%B4%D0%B0%D0%BD%D0%BE%D0%B2+%D0%95%D0%B2%D0%B3%D0%B5%D0%BD%D0%B8%D0%B9+aka+r0g3r.+Lpi+101+(%D0%A7%D0%B0%D1%81%D1%82%D1%8C+%D0%BF%D0%B5%D1%80%D0%B2%D0%B0%D1%8F)c/main.html

(Пособие для подготовки к экзамену LPIC 1)

Перевод: Буданов Евгений aka r0g3r.

LPI 101 (Часть первая)

Содержание

- 1 Перед тем как начать
- 2 Введение в bash
- 3 Используем команды Linux
- 4 Создание ссылок и удаление файлов
- 5 Использование шаблонов
- 6 Заключение и список литературы

Перед тем как начать

Об этом руководстве

Добро пожаловать в курс по основам Linux, первое из четырёх руководств посвящённых тому, чтобы подготовить вас к сдаче экзамена №101 в Linux Professional Institute. В этом руководстве мы познакомим вас с bash (стандартной оболочкой в Linux), покажем вам как использовать все преимущества команд Linux, таких как ls, cp и mv, а также объясним, что такое иноды, символические и жёсткие ссылки и многое другое. По окончанию изучения данного руководства у вас будет более полное понимание основ работы операционной системы Linux, а также вы сможете приступить к изучению некоторых базовых задач системного администрирования. По окончанию изучения всей серии руководств (всего восемь статей) у вас появятся знания о том, что необходимо для того, чтобы работать системным администратором в Linux-системах. Кроме этого, у вас появится возможность сдать экзамен LPIC Level 1, если вы этого захотите.

Данное руководство лучше всего подходит тем, кто является новичком в ОС Linux, или для тех, кто хочет освежить или улучшить свои знания в области понимания основ Linux, таких как копирование или перемещение файлов, создание жёстких или символических ссылок, а также использования стандартных команд для обработки текста используя пайпы и перенаправления. На протяжении всего курса, мы будем рады поделиться всевозможными подсказками, советами и хитростями, чтобы сделать обучение более насыщенныи и реалистичным даже для тех, кто уже имеет опыт работы с Linux. Таким образом, начинающим большая часть материала будет совершенно новой для них, а для более опытных пользователей Linux это будет хорошим способом освежить свои знания.

Введение в bash

Оболочка

Если вы уже хотя бы один раз пробовали работать в Linux, то должны знать, что при входе в операционную систему выдаётся вот такое приглашение:

$

В некоторых случаях приглашение которое вы видите может выглядеть немного иначе. Оно может содержать имя машины на которой запущена оболочка, имя текущей рабочей директории или всё сразу. но невзирая на то, как выглядит ваше приглашение, одна вещь там точно должна быть. Программа, которая печатает это приглашение называется оболочкой или шеллом. И, что наиболее вероятно, этой оболочкой будет являться программа которая называется bash.

Запущен ли у вас bash?

Чтобы проверить запущен ли у вас bash, нужно выполнить следующую команду:

$ echo $SHELL /bin/bash

Если строчка выше выдаёт какую-либо ошибку или отличается от данного примера, то скорее всего у вас запущена оболочка отличающаяся от bash. В принципе, большая часть руководства будет работать и в этом случае, но лучше всё же-таки будет если вы запустите bash, чтобы успешно начать подготовку к экзамену 101.

Что же такое bash?

Bash - это акроним образованный от "Bourne-again shell", и являющийся основной облочкой для работы в командной строке в большинстве Linux-систем. Оболочка во время работы выполняет ваши команды, таким образом вы взаимодействуете со своей Linux-системой. Когда вы закончите вводить команды, то можно либо скомандовать шеллу выйти из системы или просто завершить сеанс, таким образом, вы снова увидите строку приглашения. Кстати, вы можете выйти из bash просто нажав клавиши Control+D в строке приглашения.

Используем команду cd

Как вы уже наверное заметили, командная строка в bash не самая интересная вещь в этом мире. Давайте же воспользуемся ей, чтобы поперемещаться по нашей файловой системе. Напечатайте следующую команду без знака "$":

$ cd /

Итак, мы только что сказали bash, что мы хотим работать в / известном как корневой каталог системы. Поскольку все директории (каталоги) имеют иерархическую древовидную структуру, то / является верхушкой дерева, или корнем. Команда cd устанавливает текущую рабочую директорию. Собственно, cd так и расшифровывается "current working directory".

Пути
Чтобы в bash посмотреть, в какой же всё-таки директории вы работаете, достаточно ввести следующую команду:

$ pwd /

В примере выше видно, что / для команды cd является аргументом, который называется *путь*. Он сообщает cd, куда же мы хотим сходить. В большинстве случаев, знак / означает аргмент говорящий нам о том, что это *абсолютный путь* указывающий на местоположение начиная от корневой директории системы.

Абсолютные пути

Давайте взглянем на примеры абсолютных путей в Linux:

/dev /usr /usr/bin /usr/local/bin

Как вы можете увидеть, общим признаком указывающим на то, что путь является абсолютным, является то, что все они начинаются с символа /. Например, если вы укажете путь /usr/local/bin команде cd, то она сначала перейдёт в корень, затем в директорию usr, затем в директорию local, а потом в bin. Таким образом определить, что путь является абсолютным можно по наличию символа / в самом начале пути.

Относительные пути

Есть ещё один тип путей в системе, который называется *относительный*. Bash, команда cd и другие команды всегда интерпретируют такие пути относительно текущей директории. Относительные пути никогда не начинаются с /. Итак, мы сейчас перейдём в директорию usr:

$ cd /usr

Теперь мы можем использовать относительные пути внутри директории usr и сменить путь на /usr/local/bin:

$ cd local/bin $ pwd /usr/local/bin

Используем ..

Относитенльные пути могут содержать одну или несколько .. директорий. Директории обозначаемые как .. - это специальные директории, которые указывают на родительский каталог. В общем, продолжим на том примере, что указан выше:

$ pwd /usr/local/bin $ cd .. $ pwd /usr/local

Как теперь видно, наша текущая директория теперь стала /usr/local. Мы прошли "назад" по дереву каталогов и оказались там, где сейчас и находимся. Дополнительно можно добавить, что мы можем добавлять .. к уже имеющемуся относительному пути, чтобы пройти дальше в глубь директории. Например, вот так:

$ pwd /usr/local $ cd ../share $ pwd /usr/share

Примеры относительных путей

Относительные пути бывают разные. Ниже мы приведём примеры таких путей, а вы может попробовать ввести эти команды, чтобы увидеть, что же у вас выйдет:

$ cd /bin $ cd ../usr/share/zoneinfo $ cd /usr/X11R6/bin $ cd ../lib/X11 $ cd /usr/bin $ cd ../bin/../bin

Надеемся, что у вас всё получилось. :)

Введение в .

Перед тем как мы закончим изучение принципа работы команды cd, мы рассмотрим некоторые вещи имеющих весьма важное значение. Во-первых, существует ещё один тип специальных директорий, который обозначается как . (символ точки). Данный символ означает текущую директорию. Даже если . не используется командой cd, то ей пользуются некоторые другие программы. Например, это бывает необходимо для запуска некой программы находящейся в текущей директории:

$ ./myprog

В примере выше показано, что программа myprog, которая находится в текущей директории вызвана для выполнения.

Команда cd и домашняя директория
Если мы хотим перейти в домашний каталог, то нам достаточно сделать следующее:

$ cd

Если ввести команду cd без каких-либо аргументов, то команда вернёт вас в домашний каталог того пользователя, под которым вы сидите. Обычно это выглядит как /home/имярек. Если же вы сидите из под пользователя root, то вы попадёте в каталог корневой, то есть в /.
А как быть, если мы хотим указать какой-то файл в домашней директории? Например, мы хотим передать аргумент в виде файла команде myprog. Если всё необходимое уже лежит в вашем домашнем каталоге, то надо проделать следующее:

$ ./myprog /home/имярек/myfile.txt

Правда, использовать абсолютные пути очень неудобно. Чтобы облегчить работу, мы можно использовать символ ~ (тильда) чтобы сделать то же самое гораздо проще:

./myprog ~/myfile.txt

Домашние директории других пользователей
Bash использует символ ~ чтобы указывать на вашу домашнюю директорию, но на самом деле ,вы можете использовать этот символ, чтобы перейти в домашние директории других пользователей. Например, мы хотим передать файл с названием fredsfile.txt в домашний каталог пользователя с именем Fred, то нам надо напечатать в командной строке следующее:

$ ./myprog ~fred/fredsfile.txt

Используем команды Linux

Знакомимся с командой ls
Ну, теперь давайте пробежимся по такой команде как ls. Хоть и вполне допускаем, что вы

уже знаете про такую команду и даже знаете то, что будет выведен список содержимого, если напечатать ls в командной строке находясь в текущей рабочей директории:

$ cd /usr $ ls X11R6 doc i686-pc-linux-gnu lib man sbin ssl bin gentoo-x86 include libexec portage share tmp distfiles i686-linux info local portage.old src

Указав опцию -a, вы можете видеть все файлы в директории, включая даже те, что являются скрытыми. Например те, что начинаются с . или .. . Как видно из нижеследующего примера, команда ls -a показывает даже специальные ссылки . и .. на директории:

$ ls -a . bin gentoo-x86 include libexec portage share tmp .. distfiles i686-linux info local portage.old src X11R6 doc i686-pc-linux-gnu lib man sbin ssl

Длинные списки

Вы также можете указывать, выводить один или несколько файлов и директорий при использовании ls. Если вы укажете ls отображать только файлы, он будет показывать именно файлы. Если вы хотите видеть каталоги, то команда ls покажет *содержимое* каталога. С помощью опции -l очень удобно смотреть разрешения, владельцев, время модификации и информацию о размере в списке директории.

В примере ниже я покажу, как выглядит полный список содержимого моего каталога используя опцию -l:

$ ls -l /usr drwxr-xr-x 7 root root 168 Nov 24 14:02 X11R6 drwxr-xr-x 2 root root 14576 Dec 27 08:56 bin drwxr-xr-x 2 root root 8856 Dec 26 12:47 distfiles lrwxrwxrwx 1 root root 9 Dec 22 20:57 doc -> share/doc drwxr-xr-x 62 root root 1856 Dec 27 15:54 gentoo-x86 drwxr-xr-x 4 root root 152 Dec 12 23:10 i686-linux drwxr-xr-x 4 root root 96 Nov 24 13:17 i686-pc-linux-gnu drwxr-xr-x 54 root root 5992 Dec 24 22:30 include lrwxrwxrwx 1 root root 10 Dec 22 20:57 info -> share/info drwxr-xr-x 28 root root 13552 Dec 26 00:31 lib drwxr-xr-x 3 root root 72 Nov 25 00:34 libexec drwxr-xr-x 8 root root 240 Dec 22 20:57 local lrwxrwxrwx 1 root root 9 Dec 22 20:57 man -> share/man lrwxrwxrwx 1 root root 11 Dec 8 07:59 portage -> gentoo-x86/ drwxr-xr-x 60 root root 1864 Dec 8 07:55 portage.old drwxr-xr-x 3 root root 3096 Dec 22 20:57 sbin drwxr-xr-x 46 root root 1144 Dec 24 15:32 share drwxr-xr-x 8 root root 328 Dec 26 00:07 src drwxr-xr-x 6 root root 176 Nov 24 14:25 ssl lrwxrwxrwx 1 root root 10 Dec 22 20:57 tmp -> ../var/tmp

Первая колонка в примере отображает информацию о разрешениях для каждого пункта в списке. Чуть попозже я попробую немного объяснить, как правильно читать эту информацию. Следующая колонка указывает число ссылок на каждый файловый объект в системе. К этому пункту мы тоже вернёмся позднее. Колонки третья и четвёртая указывают имя владельца и группу соответственно. Пятая колонка описывает размер объекта. Шестая - указывает время последней модификации или mtime объекта. Последняя колонка - имя. Если вы видите в названии вот такой символ ->, это означает, что объект является символической ссылкой, а то что записано после этого символа, указание на то, куда эта ссылка указывает.

Просмотр содержимого директории

Иногда нам надо посмотреть содержимое каталога немного по-другому, чем обычно. Для такого случая существует опция -d, которая говорит нам, что надо показывать любые директории, которые обычно выглядят вот так:

$ ls -dl /usr /usr/bin /usr/X11R6/bin ../share drwxr-xr-x 4 root root 96 Dec 18 18:17 ../share drwxr-xr-x 17 root root 576 Dec 24 09:03 /usr drwxr-xr-x 2 root root 3192 Dec 26 12:52 /usr/X11R6/bin drwxr-xr-x 2 root root 14576 Dec 27 08:56 /usr/bin

Рекурсивный просмотр и список инод

Итак, вы научились использовать опцию -d, чтобы смотреть содержимое каталога. Но вы также можете ещё использовать опцию -R, чтобы сделать прямо противоположное! Смотреть не изнутри директории, а рекурсивно просмотреть местонахождение всех файлов и каталогов внутри необходимой вам директории. В примере мы не будем включать вывод команды ls -R, так как он получится крайне большим, но вы всегда можете попробовать его сами. Например, можете сначала проверить работу команды ls -R, а затем ls -Rl, чтобы понять как эти команды работают.

И, наконец, существует опция -i, показывающая номер иноды объекта файловой системы в списке директории:

$ ls -i /usr 1409 X11R6 314258 i686-linux 43090 libexec 13394 sbin 1417 bin 1513 i686-pc-linux-gnu 5120 local 13408 share 8316 distfiles 1517 include 776 man 23779 src 43 doc 1386 info 93892 portage 36737 ssl 70744 gentoo-x86 1585 lib 5132 portage.old 784 tmp

Введение в иноды

Каждый объект в файловой системе имеет уникальный индекс, который носит название *инода*. Звучит банально, но понять что такое инода крайне необходимо, чтобы разобраться во многих системных операциях. Например, давайте рассмотрим уже обсуждавшиеся выше . и .. ссылки на которые есть в любой директории. Чтобы полностью разобраться, что же такое .. на самом деле, давайте взглянем на номер иноды каталога /usr/local:

$ ls -id /usr/local 5120 /usr/local

Номером иноды каталога /usr/local является число 5120. А теперь, давайте посмотрим номер иноды у каталога /usr/local/bin/..:

$ ls -id /usr/local/bin/.. 5120 /usr/local/bin/..

Как уже стало видно, номер иноды директорий /usr/local и /usr/local/bin/.. один и тот же! [Отсутствующий здесь кусок пока не понял как переводить] Раньше мы были уверены, что /usr/local это директория сама по себе. Теперь вдруг мы узнали, что инода с номером 5120 на самом деле директорией и является и имеет там две записи именуемые ссылками, которые указывают на эту иноду. Оба каталога usr/local и /usr/local/bin/ имеют один и тот же номер 5120. Таким образом, инода с номером 5120 находится только водном месте на диске, но имеет множество ссылок на себя. Инода 5120 является реальной записью на диске.

Другими словами, мы можем просмотреть сколько ссылок имеется на иноду 5120 с помощью команды 'ls -dl':

$ ls -dl /usr/local drwxr-xr-x 8 root root 240 Dec 22 20:57 /usr/local

если мы взглянем на вторую колонку то слева, то как раз увидим, что /usr/local (инод 5120) упоминается 8 раз. У меня лично имеются такие пути, указывающие на данную иноду:

/usr/local /usr/local/. /usr/local/bin/.. /usr/local/games/.. /usr/local/lib/.. /usr/local/sbin/.. /usr/local/share/.. /usr/local/src/..

Команда mkdir

Настала пора взглянуть на команду mkdir, с помощью которой вы можете создавать директории. С помощью следующего примера мы создадим три каталога tic, tac и toe внутри /tmp:

$ cd /tmp $ mkdir tic tac toe

По умолчанию mkdir не создаёт родительские каталоги если вам это необходимо, поскольку необходимо, чтобы каждый последующий элемент уже существовал в системе. Так что если вам всё же вдруг захочется создать директории won/der/ful, то вам придётся вызывать команду mkdir три раза подряд:

$ mkdir won/der/ful mkdir: cannot create directory `won/der/ful': No such file or directory $ mkdir won $ mkdir won/der $ mkdir won/der/ful

Но команда mkdir имеет полезную опцию -p позволяющую создавать недостающие родительские каталоги:

mkdir -p easy/as/pie

ну вот, для начала, пожалуй всё. Если захотите узнать немного побольше о команде mkdir наберите в консоли man mkdir, чтобы почитать документацию к ней. Аналогично можно поступить с уже рассматривавшейся здесь командой ls и другими, за исключением команды cd, которая встроена в bash.

Команда touch

Итак, рассмотрим взглянем на команды cp и mv. Первая команда предназначена для копирования, вторая - для переноса или переименования файлов или директорий. Но перед тем как бы глянем эти команды в действии, рассмотрим ещё одну. Это команда touch, с помощью которой мы можем создать файл, например, в /tmp:

$ cd /tmp $ touch copyme

Команда touch обновляет так называемый mtime, если файл уже существует (вспомните про шестую колонку в выводе ls -l). Если такого файла нет, то touch создаёт его, при этом файл /tmp/copyme будет иметь нулевой размер.

Команда echo

Раз теперь у нас появился существующий файл, то давайте внесём в него какие-нибудь данные. Чтобы это сделать, необходимо воспользоваться командой echo, которая имеет различные аргументы и умеет выводить их прямо в потоковый вывод. Сначала, выполним команду как есть:

$ echo "firstfile" firstfile

Теперь сделаем то же самое, но уже с перенаправлением вывода:

$ echo "firstfile" > copyme

Мы сказали шеллу вывести вывод команды echo прямо в файл имеющим название copyme. Если бы этот файл существовал ранее то его содержимое было бы перезаписано. В случае его отсутствия файл создаётся с нуля и в него пишутся данные. Напечатав в шелле ls -l мы увидим, что у нас появился файл copyme размером 10 байт, содержащий внутри слово firstfile и символ новой строки:

$ ls -l copyme -rw-r--r-- 1 root root 10 Dec 28 14:13 copyme

Команды cat и cp

Чтобы просмотреть содержимое файла в терминале необходимо воспользоваться командой cat:

$ cat copyme firstfile

А теперь воспользуемся самым простым вызовом команды cp, чтобы создать файл copiedme из уже имеющегося файла с именем copyme:

$ cp copyme copiedme

После некоторого изучения видим, что мы действительно создали разные файлы. Взгляните на их разные номера инод:

$ ls -i copyme copiedme 648284 copiedme 650704 copyme

Команда mv А теперь давайте воспользуемся командой mv, чтобы переименовать файл copiedme в movedme. Заметьте, что номер иноды хоть и остался прежним, но имя файла изменилось:

$ mv copiedme movedme $ ls -i movedme 648284 movedme

Номер иноды перемещённого файла останется одним и тем же до тех пор, пока перемещённый файл находится на той же самой файловой системе, что и исходный файл. Почему так происходит мы узнаем ближе к третьей части серии данных руководств.

Поскольку мы сейчас затронули команду mv, то давайте рассмотрим другие способы использования данной команды. Помимо уже известного нам способа для переименования файлов, mv умеет перемещать один или несколько файлов в другое место в дереве каталогов. Например, давайте переместим файл /var/tmp/myfile.txt в /home/drobbins (который является моей домашней директорией). Для этого я должен ввести следующее:

$ mv /var/tmp/myfile.txt /home/drobbins

После введения этой команды myfile.txt будет перемещён /home/drobbins/myfile.txt. Если /home/drobbins находится на отличной от /var/tmp файловой системе, то mv перехватит копирование myfile.txt в новое место и сотрёт его в старом. Ну, и как вы уже наверное догадались, при перемещении между разными Файловыми системами у файла myfile.txt меняется номер инода. Это связано с тем, что номера инод на каждой файловой системе свои.

Мы так же можем использовать команду mv, чтобы копировать несколько файлов в одно место. Например, чтобы перенести файлы myfile1.txt и myarticle3.txt в /home/drobbins я должен буду ввести следующее:

$ mv /var/tmp/myfile1.txt /var/tmp/myarticle3.txt /home/drobbins

Создание ссылок и удаление файлов

Жесткие ссылки (хардлинки)

Мы используем термин ссылка в значении связи по их отношению к родительским записям (или именами, если назвать их более привычно) и инодами (номерами индексов нижележащей файловой системы, которую мы, обычно, можем игнорировать). Существует два типа ссылок доступных в Linux. Для начала мы разберём тип ссылок именуемых *жёсткими ссылками*. Существующая инода может иметь любое количество жёстких ссылок. Инода продолжает существовать на файловой системе даже тогда, когда исчезают все жёсткие ссылки. Когда последняя жёсткая ссылка исчезает и нет ни одной программы, которая удерживала бы открытый файл, операционная система Linux автоматически удаляет этот файл. новая жесткая ссылка может быть создана следующим образом:

$ cd /tmp $ touch firstlink $ ln firstlink secondlink $ ls -i firstlink secondlink 15782 firstlink 15782 secondlink

Как можно увидеть в примере, жёсткая ссылка работает на уровне инод указывая на обычный файл. В Linux-системах жёсткие ссылки имеют некоторые ограничения. Одно из них, вы не можете создавать жёсткие ссылки на директории, только на файлы. Что в общем-то верно. Если подумать, то . и .. это созданные операционной системой жёсткие ссылки на каталоги, но при этом вы сами (даже если вы сидите из под пользователя root) не можете создавать свои жёсткие ссылки на каталоги. Второе ограничение - жёсткие ссылки не различают файловых систем. Это означает, что вы не можете создавать ссылку на /usr/bin/bash в /bin/bash, если каталоги / и /bin находятся на разных файловых системах.

Символические ссылки

На практике, символические ссылки (или *симлинками*) используются куда чаще жёстких ссылок. Симлинки - это особый тип файлов, где ссылка указывает на совершенно другой файл, который ко всему прочему, может находится даже в другой иноде. Симлинки не защищают файл от удаления, так что если исходный файл был по некоторым причинам удалён, то симлинк будет испорчен или им нельзя будет пользоваться.

Симлинки создаются с помощью всё той же командой ln, но с использованием опции -s:

$ ln -s secondlink thirdlink $ ls -l firstlink secondlink thirdlink -rw-rw-r-- 2 agriffis agriffis 0 Dec 31 19:08 firstlink -rw-rw-r-- 2 agriffis agriffis 0 Dec 31 19:08 secondlink lrwxrwxrwx 1 agriffis agriffis 10 Dec 31 19:39 thirdlink -> secondlink

Символические ссылки можно отличить от обычных файлов при выводе команды ls -l тремя способами. Во-первых, самый первый столбец в листинге может содержать букву l, что чётко идентифицирует его как символическую ссылку. Во-вторых, размер символической ссылки это количество символов в названии исходного файла (в примере это слово secondlink). В-третьих, последняя колонка в списке файлов содержит уже упоминаемый символ -> указывающий на целевой файл.

Симлинки в деталях Символические ссылки гораздо более гибкие и удобные в применении, в отличие от жёстких ссылок. Вы можете создавать по своему желанию символическую ссылку на любой файл в системе включая каталоги. Поскольку симлинки основаны на путях, а не инодах, вы прекрасно можете создавать ссылки на объекты находящиеся на совершенно другой файловой системе. Однако есть пара вещей о которых следовало бы знать.

Представим ситуацию, когда вы хотите создать ссылку в /tmp указывающую на /usr/local/bin. Скажем, сделаем это вот так:

$ ln -s /usr/local/bin bin1 $ ls -l bin1 lrwxrwxrwx 1 root root 14 Jan 1 15:42 bin1 -> /usr/local/bin

Или по-другому:

$ ln -s ../usr/local/bin bin2 $ ls -l bin2 lrwxrwxrwx 1 root root 16 Jan 1 15:43 bin2 -> ../usr/local/bin

Как вы заметили, обе символические ссылки ведут в одну и ту же директорию, но во втором случае символическая ссылка "переехала" в другую директорию и была нарушена потому, что относителный путь к ней путь стал таким:

$ ls -l bin2 lrwxrwxrwx 1 root root 16 Jan 1 15:43 bin2 -> ../usr/local/bin $ mkdir mynewdir $ mv bin2 mynewdir $ cd mynewdir $ cd bin2 bash: cd: bin2: No such file or directory

Так как пути /tmp/usr/local/bin не существует, мы не можем менять содержимое в bin2. Другими словами, симлинк был испорчен.

По этой причине я бы иногда советовал избегать создания символических ссылок с относительными путями в названии. Тем не менее, есть много случаев, когда ссылки на относительные пути бывают полезны. К примеру, представим ситуацию, когда вы хотите создать альтернативное имя для программы располагающейся в /usr/bin:

ls -l /usr/bin/keychain -rwxr-xr-x 1 root root 10150 Dec 12 20:09 /usr/bin/keychain

Как суперпользователь root, вы можете создать альтернативное имя для файла keychain. Например, kc. В этом примере у нас имеется доступ от лица суперпользователя, что видно по тому, что приглашение в командной строке изменилось на значок #. Доступ в лице суперпользователя нам крайне необходим, так как мы не имеем права создавать файлы в /usr/bin работая из под обычного пользователя. Ну, а имея доступ из под пользователя root мы можем проделать примерно такое:

cd /usr/bin # ln -s /usr/bin/keychain kc # ls -l keychain -rwxr-xr-x 1 root root 10150 Dec 12 20:09 /usr/bin/keychain # ls -l kc lrwxrwxrwx 1 root root 17 Mar 27 17:44 kc -> /usr/bin/keychain

В этом примере мы создали символическую ссылку kc указывающую на файл

/usr/bin/keychain.

Но это же решение может создать определённые проблемы, если мы захотим переместить оба файла /usr/bin/keychain и /usr/bin/kc в /usr/local/bin:

mv /usr/bin/keychain /usr/bin/kc /usr/local/bin # ls -l /usr/local/bin/keychain -rwxr-xr-x 1 root root 10150 Dec 12 20:09 /usr/local/bin/keychain # ls -l /usr/local/bin/kc lrwxrwxrwx 1 root root 17 Mar 27 17:44 kc -> /usr/bin/keychain

Поскольку мы использовали абсолютные пути при [создании символической ссылки](), то наш симлинк kc всё ещё указывает на /usr/bin/keychain которого уже вообще-то не существует, поскольку мы перенесли usr/bin/keychain в /usr/local/bin. Что означает, что наш kc стал испорченным симлинком.

Поскольку относительный и абсолютный путь при создании символических ссылок имеет свои достоинства, то надо понимать, какой из этих типов вам лучше всего использовать в конкретной ситуации. В большинстве случаев, симлинки [имеющих как абсолютные](), так и относительные пути работают вполне хорошо. Следующий пример наглядно покажет, что симлинк будет хорошо работать даже если оба файла будут перемещены:

cd /usr/bin # ln -s keychain kc # ls -l kc lrwxrwxrwx 1 root root 8 Jan 5 12:40 kc -> keychain # mv keychain kc /usr/local/bin # ls -l /usr/local/bin/keychain -rwxr-xr-x 1 root root 10150 Dec 12 20:09 /usr/local/bin/keychain # ls -l /usr/local/bin/kc lrwxrwxrwx 1 root root 17 Mar 27 17:44 kc -> keychain

Теперь мы можем запустить программу keychain введя в консоли /usr/local/bin/kc. /usr/local/bin/kc ссылается на программу keychain, которая лежит в той же директории что и kc.

Команда rm

Ну вот. Мы узнали про такие команды как mv, cp и ln. Настало самое время узнать как удалять объекты из файловой системы. Обычно это делается с помощью команды rm. Чтобы удалить файлы достаточно указать их в командной строке:

$ cd /tmp $ touch file1 file2 $ ls -l file1 file2 -rw-r--r-- 1 root root 0 Jan 1 16:41 file1 -rw-r--r-- 1 root root 0 Jan 1 16:41 file2 $ rm file1 file2 $ ls -l file1 file2 ls: file1: No such file or directory ls: file2: No such file or directory

Примечание для пользователей Linux. Дело в том, что если вы вводите команду rm, то файлы удаляются навсегда. Поэтому начинающим системным администраторам было бы нелишне использовать опцию -i, это включает интерактивный режим и появляется запрос перед удалением файлов. Например, вот так:

$ rm -i file1 file2 rm: remove regular empty file `file1'? y rm: remove regular empty file `file2'? y

В примере показанном выше, rm послала запрос, действительно ли вы хотите удалить указанные файлы. В случае положительного ответа необходимо нажать "y" и клавишу Enter дважды. Если бы была нажата клавиша n, то ничего удалено бы не было. В том случае, если б что-то пошло не так, достаточно нажать клавиши Control-D, чтобы прервать выполнение команды rm -i полностью до того, пока системе были нанесены какие-либо повреждения.

Если вы используете команду rm достаточно часто, то есть смысл добавить следующую строку в ваш файл ~/.bashrc используя ваш любимый текстовый редактор. Затем необходимо выйти и снова войти в систему. Теперь каждый раз, когда вы будете набирать rm, у вас будет автоматически выполняться удаление в интерактивном режиме:

alias rm="rm -i

Команда rmdir

Удалить каталог, вы можете двумя способами. Вы можете удалить всё содержимое внутри каталога, а затем удалить сам каталог с помощью команды rmdir:

$ mkdir mydir $ touch mydir/file1 $ rm mydir/file1 $ rmdir mydir

Этот метод обычно обзывается как "способ удаления файлов для лохов". Продвинутые же пользователи, а также системные администраторы используют команду rm -rf, чтобы сделать то же самое.

Самый хороший способ удалить содержимое директории, это использование *принудительной рекурсии* в команде rm, которая сообщает ей, что необходимо удалить не только содержимое директории, но и её саму:

$ rm -rf mydir

Обычно, этот способ наиболее предпочтителен, когда надо удалить целое дерево каталогов. Будьте внимательны используя rm -rf. Ведь её сила может быть направлена как в злых, так и добрых целях :)

Использование шаблонов

Введение в использование поисковых шаблонов

Во время вашего ежедневного использования Linux, вы, наверное, много раз делали часто повторяющуюся работу состоящую как правило из одной команды (например, rm) со множеством файлов хотя бы однажды. В таких случаях часто получаются утомительные вещи вроде такого:

$ rm file1 file2 file3 file4 file5 file6 file7 file8

Чтобы решить эту проблему, мы предлагаем вам ознакомиться со встроенной в Linux поддержкой поисковых шаблонов. Оно также называется как "подстановка" (так уж исторически сложилось), позволяющая указывать множество файлов используя поисковые шаблоны. Bash и другие команды в Linux, интерпретируют эти шаблоны во время поиска по диску и находя любые файлы подпадающие под этот шаблон. Таким образом, если у вас в каталоге лежат файлы начинающиеся с имени file1 и заканчивающиеся файлом с именем file8, вы можете удалить их с помощью такой команды:

$ rm file[1-8]

Или если вы просто хотите удалить файлы начинающиеся на file, то можно сделать ещё проще:

$ rm file*

Символ * в данном шаблоне обрабатывается как любой символ или их сочетание, а то и вовсе как их отсутствие. Подобные подстановки могут быть в дальнейшем использованы не только для более простого удаления файлов. Что, кстати, мы и рассмотрим ниже.

Изучаем шаблоны поиска исключений

Если вам бы вам вдруг понадобился понабился список всех файлов в /etc начинающихся на g, вы бы наверняка ввели нечто подобное:

$ ls -d /etc/g* /etc/gconf /etc/ggi /etc/gimp /etc/gnome /etc/gnome-vfs-mime-magic /etc/gpm /etc/group /

Но что делать, если у вас ничего не нашлось? В следующем примере мы попробуем поискать файлы в директории /usr/bin, которые начинаются на asdf и заканчиваются jkl. Причём прошу заметить, что в поиск будет включён файл asdfjkl, который тоже может нам встретиться.

$ ls -d /usr/bin/asdf*jkl ls: /usr/bin/asdf*jkl: No such file or directory

Вот что же у нас произошло. В обычной ситуации мы задаём некий паттерн, в котором он ищет совпадения по одному или нескольким файлах имеющихся в файловой системе, в то время как *bash подменяет их на разделяемый пробелами список всех подходящих файлов*. Однако, если ничего не находится из того, что могло бы подпасть под этот поисковый шаблон, то *bash оставляет все аргументы и шаблоны без изменений*. Таким образом, если команда ls ничего не вывела подходящего под критерий /usr/bin/asdf*jkl, то она выдаёт ошибку. Основным моментом в такой ситуации является то, что шаблон подстановки срабатывает только если находит нечто удовлетворяющее условиям поиска на диске. *В противном случае, всё остаётся как есть, и передаётся запущенной программе в своём изначальном виде.*

Синтаксис шаблонов поиска: символы * и ?

Изучив то, как работает алгоритм подстановки имён, нам следует взглянуть на то, как работает шаблон поиска. Для этого мы можем использовать самые разные символы расширяющие круг поисков: Символ * использует совпадение по любому количеству символов от нуля до бесконечности. Что на практике означает примерно следующее: "Мы можем использовать всё что угодно включая ничего". Например:

- /etc/g* отобразит все файлы в каталоге /etc которые начинаются на букву g, или же сам файл, который называется g.
- /tmp/my*1 отображает все файлы в каталоге /tmp которые начинаются на my и заканчиваются на 1, включая файл с названием my1.

Символ ? подменяет один-единственный символ. К примеру:

- myfile? покажет люьой файл имеющий в своём имени myfile и имеющий в конце своего названия любой символ.
- /tmp/notes?txt должен вывести вам оба варианта, как /tmp/notes.txt так и /tmp/notes_txt, если таковые существуют.

Синтаксис шаблонов поиска: символ []

Данный символ при использовании довольно похож по результату работы на символ ?, только со своей спецификой. Чтобы использовать данный тип шаблона, необходимо вписать нужные символы для поиска внутрь этих скобок. Самое интересное, что в данном случае вы можете использовать для поиска диапазон значений, а также всячески это комбинировать. К примеру:

- myfile[12] покажет в результатах поиска файлы myfile1 и myfile2. Данный шаблон может быть расширен как минимум, на столько символов, сколько их отыщется у вас на в текущей директории.
- [Cc]hange[Ll]og - покажет вам в результатах поиска Changelog, ChangeLog, changeLog, и даже changelog. Как тут вам уже стало ясно, внутри квадратных скобок указаны

различные регистры букв, что бывает полезным при поиске слов имеющих в своём составе строчные или прописные буквы, но вы не знаете какие именно.

- ls /etc/[0-9]* просто покажет список имён файлов в /etc начинающихся с цифр.
- ls /tmp/[A-Za-z]* покажет список всех файлов в /tmp которые начинаются на строчную или прописную букву.

Конструкция вида [!] довольна похожа на уже рассмотренный выше [], с тем отличием что он делает прямо противоположное. То есть наоборот исключает из поиска указанные в данной конструкции символы для поиска указанные после символа !. Пример:

- rm myfile[!9] удалит с диске все файлы с названием myfile с любым символом в конце имени, кроме файла с именем myfile9.

Некоторые пояснения по работе шаблонов поиска

В данном разделе мы дадим некоторые разъяснения касающиеся работы данных шаблонов. Поскольку bash обрабатывает символы которые используются при работе с шаблонами (?, [,], и *), поэтому вам следует проявлять внимательность при вводе команд, которые в качестве аргументов для использования содержат данные символы. Например, вы хотите создать файл используя при этом строку [fo]*, но команда которую вы введёте не сделает того, чего вы от неё хотите:

```
$ echo [fo]* > /tmp/mynewfile.txt
```

Если бы [fo]* искала совпадения с любыми указанными в шаблоне файлами, то вы бы нашли их список в файле, но вместо списка файлов вам нужен простой текст [fo]* в /tmp/mynewfile.txt. Как быть? Всё, что вам необходимо - это окружить одинарными кавычками текст [fo]*, чтобы оболочка не обрабатывала его как шаблон:

```
$ echo '[fo]*' > /tmp/mynewfile.txt
```

Используя данный способ вы обнаружите, что только что созданный файл содержит буквы [fo]*, как вы того и хотели. Как вариант, можно ещё использовать выделение текста с помощью символов обратной косой черты:

```
echo \[fo\]\* > /tmp/mynewfile.txt
```

Оба перечисленных способа в виде обрамления символами обратной косой черты или одинарными кавычками делают одно и то же. И да, уж коли мы заговорили о символе обратной косой черты, то настало самое время упомянуть о порядке обработки символа \, когда вы можете выделить текст одинарными кавычками, или \\ вместо них, который будет обрабатываться как \.

Примечание:

Двойные кавычки работают похожим с одинарными кавычками образом, с тем исключением, имеют гораздо более ограниченную область применения. Использовать одинарные кавычки лучше всего тогда, когда вы хотите поместить обычный текст в некую команду. Для большего понимания о чём речь, и вообще о шаблонах, рекомендую набрать в консоли man 7 glob. Чтобы узнать более подробную информацию о закавычивании в bash, то наберите в консоли man 8 glob и внимательно прочтите секцию QUOTING. Если вы впоследствии планируете сдать экзамен LPI, то оставляю это вам в качестве домашнего задания при подготовке к нему. :)

Заключение и список литературы

Заключение

Ну что же. Мои поздравления! Вы только что дощли до конца курса по самым основам Linux. Очень надеюсь, что данный текст поможет вам пополнить знания о базовых основах работы в ОС Linux. Темы которые вы прошли, включают в себя основы командной оболочки bash, основные команды в Linux, ссылки и шаблоны. В следующем руководстве мы будем изучать такие вещи как регулярные выражения, настройку владельцев, настройку разрешений для доступа к файлам, управления учётными записями пользователей и многое-многое другое.

Пройдя до конца всю серию данных руководств, вы сможете быть готовым к сдаче сертификата LPIC первого уровня. Если это вас заинтересовало, то рекомендую посмотреть список материалов который указан в самом конце.

LPI 101 (Часть вторая)

Содержание

- 1 Перед тем как начать
- 2 Регулярные выражения
- 3 Стандарт FHS и поиск файлов
- 4 Контроль за процессами
- 5 Обработка текста
- 6 Модули ядра
- 7 Заключение и ресурсы

Перед тем как начать

О данном руководстве

Добро пожаловать в "Основы администрирования", вторую часть из серии руководств разработанных специально для подготовки к сдаче экзамена Linux Professional Institute 101.

В данном руководстве будет рассказано как использовать регулярные выражения для поиска файлов по определённым шаблонам. Также мы познакомимся с Filesystem Hierarchy Standard (FHS), описывающим где находятся те или иные файлы в системе. Затем мы научимся контролировать процессы в вашей системе, которые были запущены в фоновом режиме, выводить списки процессов, отключать процессы от терминала и многое другое. Кроме того, мы изучим вводный курс по таким вещам как конвейеры, перенаправление вывода, команды для обработки текста. И, наконец, мы познакомимся с модулями ядра Linux.

Это руководство (часть вторая) подходит для тех, кто уже имеет хорошие базовые знания о работе в bash и хочет получить более целостное восприятие того, что необходимо знать о задачах системного администрирования. Если же вы совсем новичок в Linux, то для начала рекомендуем изучить первую часть руководства, перед тем как продолжить изучение данного материала. Тем, кому по большей части предназначается эта работа, узнают много нового для себя, но даже те, кто уже имеют знания в этой области могут освежить свои знания в этой области и подтянуть навыки системного администрирования в Linux.

Тем кто изучал первую часть данного руководства по причинам не связанным с подготовкой к экзамену LPI, то вряд ли вам это понадобится в дальнейшем. Если вы всё же решили готовиться к экзамену, то изучение данных материалов крайне рекомендуется.

Регулярные выражения

Что такое регулярные выражения?

Регулярные выражения (также называемые как regex или regexp) - это специальный синтаксис используемый для описания различных текстовых шаблонов. В Linux-системах регулярные выражения чаще всего используются чтобы найти некий кусок текста, над которым необходимо произвести операцию замены.

Сравнение с подстановкой

Изучая регулярные выражения, вы, наверное, заметите, что они по своему назначению довольно схожи с шаблонами подстановки, которые мы рассматривали в первой части данной работы. Однако, не стоит обольщаться. Несмотря на внешнюю похожесть, оба механизма имеют коренные различия. Регулярные выражения и шаблоны подстановки - это абсолютно разные вещи.

Простые подстрочные выражения

Итак, получив необходимые наставления, давайте приступим к изучению наиболее общего и простого типа регулярных выражений - простых подстрочных выражений. Чтобы это сделать, нам необходимо будет использовать команду grep, которая сканирует содержимое текста согласно заданным регулярным выражениям. Команда grep выводит любую строку текста подходящую под заданный шаблон и пропускает то, что под него не подходит:

$ grep bash /etc/passwd operator:x:11:0:operator:/root:/bin/bash root:x:0:0::/root:/bin/bash ftp:x:40:1::/home/ftp:/bin/bash

Собственно, в данном примере первое слово после grep является регулярным выражением. второе слово - это имя файла. Grep читает каждую строку в файле /etc/passwd и выдаёт простой подстрочник содержащий слово bash, если он его находит. В случае нахождения очередного совпадения, он выводит очередной найденный вариант строкой ниже. Если же он ничего не находит, то он игнорирует строку и идёт дальше.

Разбираемся с работой простых подстрочников

В общем случае, если вы ищете подстроку, то вам достаточно указать в условиях поиска точное слово без использования как-либо спецсимволов. но если вам надо отыскать что-то специфическое, где в тексте имеются такие символы как +, ., *, [,], или \, то в этом случае необходимо закавычить данный текст и поместить всё это внутрь символов косой обратной черты. Давайте для большей ясности продемонстрируем несколько примеров:

- /tmp (ищет строку содержащую слово /tmp)

- "\[box\]" (ищет строку содержащую слово [box])
- "*funny*" (ищет строку содержащую слово *funny*)
- "ld\.so" (ищет строку содержащую слово ld.so)

Метасимволы

С помощью регулярных выражений вы можете проводить более сложный поиск, чем в тех примерах, которые мы рассматривали до этого, не используя так называемые метасимволы (их ещё называют спецсимволами (прим. перев.)). Одним из таких метасимволов является знак . (называемый как "период"), который означает любой символ в единственном числе:

$ grep dev.hda /etc/fstab /dev/hda3 / reiserfs noatime,ro 1 1 /dev/hda1 /boot reiserfs noauto,noatime,notail 1 2 /dev/hda2 swap swap sw 0 0 #/dev/hda4 /mnt/extra reiserfs noatime,rw 1 1

В данном примере текста dev.hda не существует ни в одной строке в файле /etc/fstab. Тем не менее, grep и не пытается искать текст, который бы точно назывался dev.hda, но он ищет шаблон текста dev.hda. Запомните ещё раз - метасимвол . заменяет собой один любой символ. В этом отношении его функциональность схожа с символом ? при используемая в шаблонах подстановки.

Использование символа квадратных скобок []

Если мы хотим найти какой-либо символ, который не подходит под использование спецсмвола ., то можно попробовать использовать [] (квадратные скобки), чтобы указать набор символов, которые мы хотели бы найти:

$ grep dev.hda[12] /etc/fstab /dev/hda1 /boot reiserfs noauto,noatime,notail 1 2 /dev/hda2 swap swap sw 0 0

Тут можно обратить внимание на то, что данная особенность работает совершенно аналогично квадратным скобкам используемых в обычных поисковых шаблонах. И снова напомню, что одна из особенностей регулярных выражений в том, что синтаксис их похож, но не идентичен синтаксису уже рассмотренных нами шаблонов подстановки. Иногда подобные вещи сбивают с толку при изучении регулярных выражений.

Использование символа [^]

Вы можете изменить значение квадратных скобок, разместив там символ ^ сразу же прямо за левой квадратной скобкой. Данное действие будет означать, что будет производиться поиск по тем символам, которые *не указаны* в квадратных скобках. И снова мы замечаем схожесть работы с символом !, который мы использовали при работе с обычными шаблонами:

$ grep dev.hda[^12] /etc/fstab /dev/hda3 / reiserfs noatime,ro 1 1 #/dev/hda4 /mnt/extra reiserfs noatime,rw 1 1

Различный синтаксис

Следует сделать крайне важное замечание. Дело в том, что синтаксис внутри квадратных скобок полностью отличается от всего остального используемого в регулярных выражениях. Например, если поместь символ . в квадратные скобки, то он станет обрабатываться как символ обычной точки, как цифры 1 и 2 в примерах выше. Но если вынести символ точки за пределы квадратных скобок, то он будет обрабатываться как метасимвол до тех пор, пока мы не поставим символ косой обратной черты. Просим учитывать этот факт, когда попытаетесь вывести на экран список всех строк в файле /etc/fstab, которые содержат в себе текст dev.hda:

$ grep dev[.]hda /etc/fstab

И другой способ:

$ grep "dev\.hda" /etc/fstab

Ни одно из этих регулярных выражений не покажет совпадений в /etc/fstab.

Метасимвол "*"
Некоторые метасимволы ничего не делают будучи используемыми сами по себе, но они меняют значение тех символов, которые стоят прямо за ними. Одним из таких символов является * (астериск), использующийся для нахождения одной или нескольких встречающихся последовательностей. Это также означает совершенно иной смысл в сравнении с шаблоном подстановки. Для лучшего понимания приведём примеры использования данного метасимвола и в скобках укажем его отличие от подстановочного шаблона:

- ab*c находит abbbbc, но не abqc (в случае использования символа подстановки он бы показал оба варианта. Кстати, вы можете объяснить почему?)
- ab*c находит abc, но не abbqbbc (в случае использования символа подстановки были бы показаны оба варианта)
- ab*c находит ac, но не cba (в случае использования символа подстановки ничего не было найдено)

- b[cq]*e находит bqe и be (в случае использования символа подстановки, был бы найден bqe, но не be)
- b[cq]*e находит bccqqe, но не bccc (в случае использования символа подстановки, был бы найден первый вариант, но не второй)
- b[cq]*e нашёл бы bqqcce но не cqe (с символом подстановки был бы найден только первый вариант)
- b[cq]*e может найти bbbeee (но только не в случае символа подстановки)
- .* совпадает с любой строкой. (в случае использования как символа подстановки, был бы показан вариант, когда имя начинается с точки)
- foo.* показывает совпадение с вариантом, где строка начинается на foo (при использовании символа подстановки будет показан вариант, в котором будет совпадение с любой строкой в которых имеется четыре буквы начинающихся на foo..)

А теперь самое время для небольшой задачки, которая напряжёт ваш мозг: строка ac совпадает с регулярным выражением ab*c, поскольку символ * также позволяет предшествующему выражению (b) быть задействованным *ноль раз*. Таким образом, ещё раз напоминаем, что механизм работы символа астериск при использовании его как спецсимвола полностью отличается от его использования в шаблонах поиска.

Начало и конец строки

Последние виды специальных символов, которые мы бы хотели тут рассмотреть, это $ и ^ используемые для обозначения конца и начала строки. Используя спецсимвол ^ в самом начале регулярного выражения, вы можете обозначить в шаблоне начало строки. В следующем примере мы используем ^# для того, чтобы найти любую строку в файле /etc/fstab начинающуюся со знака решётки:

$ grep ^# /etc/fstab # /etc/fstab: static file system information. #

Полнотекстовые регулярные выражения

Символы ^ и $ вполне могут быть скомбинированы в одной строке. Для наглядности, мы поместим символ # в начале строки и символ . в самом конце с любыми буквами, цифрами и прочими знаками посередине:

$ grep '^#.*\.$' /etc/fstab # /etc/fstab: static file system information.

Собственно, в примере выше мы поместили указанное регулярное выражение внутрь одинарных кавычек, чтобы избежать обработки символа $ оболочкой. Без этих скобок данное выражение будет проинтерпретировано совершенно неверно и приведёт к тому, что его обработка дойдёт до символа $ где и закончится.

Стандарт FHS и поиск файлов

Введение в стандарт FHS (Filesystem Hierarchy Standard)
Стандарт Иерархии Файловой Системы - это документ описывающий расположение каталогов с данными в файловой системе операционной системы Linux. FHS был придуман для того, чтобы определить размётку и упростить разработку программных продуктов независимыми программистам. Данный стандарт указывает на то, что должно быть единым в любом дистрибутиве Linux. Согласно спецификации FHS, расположение каталогов (директорий) должно быть следующим:

- / (корневой раздел)
- /boot (файлы необходимые для загрузчика)
- /dev (файлы устройств)
- /etc (настройки кнофигурации специфичные для данной машины)
- /lib (необходимые разделяемые библиотеки и модули ядра)
- /mnt (точка монтирования для различных устройств)
- /opt (место для расположения дополнительных пакетов)
- /sbin (необходимые для операционной системы библиотеки)
- /tmp (временные файлы)
- /usr (дополнительная иерархия)
- /var (прочие данные)

Две независимых категории FHS

Стандарт FHS основан на той идее, что имеются две различные категории файлов: разделяемые и не разделяемые, а также статические и переменные. Разделяемые данные могут быть разнесены между различными машинами, а неразделяемые описывают конфигурацию отдельно взятого хоста (например конфигурационные файлы на вашем компьютере). Переменные данные могут быть так или иначе перезаписаны, а статические остаются относительно неизменными (за исключением случаев, когда вы производите обновление или обслуживание системы).

Следующая таблица наглядно покажет вам четыре различных комбинации вместе с примерами каталогов, которые под эти критерии подпадают. Сама таблица взята из спецификаций FHS:

+--------------+-----------------+-------------+ | | разделяемые |неразделяемые| +--------------+-----------------+-------------+ |статические | /usr | /etc | | | /opt | /boot | +--------------+-----------------+-------------+ |переменные | /var/mail | /var/run | | | /var/spool/news | /var/lock | +--------------+-----------------+-------------+

Дополнительная иерархия в /usr

В каталоге /usr можно найти ещё одну иерархическую систему каталогов, которая выглядит весьма похожей на основную систему, которую вы видите у себя в корневом разделе. Нет ничего страшного в том, что /usr уже есть в наличии в момент включения компьютера. И это несмотря на то, что данный каталог вполне может быть сетевым (то есть разделяемым), или же напротив, он может быть смонтирован с CD-диска (статическим). Собственно, большинство инсталляций Linux не требуют разделения /usr, но тем не менее, понимание того, чем отличается основная иерархическая система в корневой директории от той, которая находится в /usr крайне полезно.

Вот и всё, что хотелось бы сказать для начала про стандарт HFS. Сам документ описывающий всё это читается довольно легко, посему любопытствующие могут просмотреть его сами. Думаю, устройство файловой системы в Linux станет более понятным, если вы его прочитаете. Сам документ находится по адресу http://www.pathname.com/fhs/.

Поиск файлов

В системах с установленной ОС Linux имеются сотни, а то и тысячи файлов. Вполне можем допустить, что вы обладаете хорошей памятью и никогда не забываете что и где лежит. Тем не менее, время от времени всё же возникает необходимость что-нибудь найти. В Linux имеется несколько инструментов позволяющих выполнить данную задачу. Данный материал поможет вам подобрать верный инструмент для этой работы.

Переменная PATH

Когда вы запускаете команду из командной строки, то bash просматривает некий список каталогов, чтобы найти искомую программу. К примеру, если вы вдруг решите ввести команду ls, то bash изначально не может знать, что данная программа находится в /usr/bin. Однако оболочка знает, что существует переменная PATH, в которой внесёт список необходимых каталогов разделённых двоеточиями. Давайте-ка поглядим на эту переменную:

$ echo $PATH /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin

Получив вывод переменной PATH (лично у вас он может несколько отличаться), bash сначала проверяет /usr/local/bin, а затем /usr/bin на наличие программы 'ls'. Скорее всего, ls окажется в /usr/bin, так что как только она будет найдена поиск прекратится.

Редактирование переменной PATH

Вы можете добавить новое значение в переменную PATH сделав следующее:

$ PATH=$PATH:~/bin $ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin:/home/agriffis/bin

В случае возникшей необходимости необходимое значение можно также удалить, но сделать это будет несколько сложнее, если не знать содержимого переменной заранее. Самых хорошим способом сделать необходимое в данной ситуации будет следующее:

$ PATH=/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:~/bin $ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/agriffis/bin

Чтобы созхранить полученный результат для дальнейшей работы, нужно ввести:

$ export PATH

Всё о команде which

Вы можете определить наличие необходимой программы в списке каталогов указанных в переменной PATH с помощью инструмента which. Ради интереса попробуем поискать с его помощью программу с названием sense:

$ which sense which: no sense in (/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin)

Или небезуспешно попробуем найти ls:

$ which ls /usr/bin/ls

Команда which -a

И, наконец, мы поглядим на использование программы which с опцией -a, которая покажет все места расположения вашей программы указанной внутри переменной PATH:

$ which -a ls /usr/bin/ls /bin/ls

Команда whereis

Если вам необходимо чуть больше информации о наличии той или иной программы, то вам точно стоит попробовать команду whereis:

$ whereis ls ls: /bin/ls /usr/bin/ls /usr/share/man/man1/ls.1.gz

Из информации приведённой выше видно, что утилита ls располагается в двух местах, в /bin и /usr/bin. Помимо этого, оттуда так же следует, что программа имеет справку располагающуюся в /usr/share/man. Данную справку можно просмотрев введя в консоли man ls.

Программа whereis также имеет возможности для поиска исходных текстов к программам, указывать дополнительные пути для поиска и искать отличные от стандартных записи. Для более подробной информации просмотрите справку по этой программе.

Команда find

Команда find это ещё один полезный инструмент в вашем наборе. Используя find вы не ограничены поиском только одних программ. Вы также можете искать любые файлы по вашему желанию, испоьзуя самые разнообразные критерии для поиска. В примере ниже мы покажем, как можно найти файл с именем README в /usr/share/doc:

$ find /usr/share/doc -name README /usr/share/doc/ion-20010523/README /usr/share/doc/bind-9.1.3-r6/dhcp-dynamic-dns-examples/README /usr/share/doc/sane-1.0.5/README

Команда find и использование шаблонов

Вы можете использовать команду find в сочетании с универсальными шаблонами добавив к команде аргумент -name допускающий оформление нужного текста кавычками или поставить символ косой обратной черты. Дляпримера мы попробуем найти тот файл README, но уже с указанием расширения файла:

$ find /usr/share/doc -name README* /usr/share/doc/iproute2-2.4.7/README.gz /usr/share/doc/iproute2-2.4.7/README.iproute2+tc.gz /usr/share/doc/iproute2-2.4.7/README.decnet.gz /usr/share/doc/iproute2-2.4.7/examples/diffserv/README.gz /usr/share/doc/pilot-link-0.9.6-r2/README.gz /usr/share/doc/gnome-pilot-conduits-

0.8/README.gz /usr/share/doc/gimp-1.2.2/README.i18n.gz /usr/share/doc/gimp-1.2.2/README.win32.gz /usr/share/doc/gimp-1.2.2/README.gz /usr/share/doc/gimp-1.2.2/README.perl.gz [578 additional lines snipped]

Пропуск символов при поиске

Пользуясь командой find, мы так же можем задать какие символы мы собираемся игнорировать при поиске:

$ find /usr/share/doc -name '[Rr][Ee][Aa][Dd][Mm][Ee]*'

Использование find с регулярными выражениями

Если вы знакомы с использованием регулярных выражений, то можно работать с утитилтой find задав опцию -regex, позволяющей ограничить вывод результатов поиска согласно заданному образцу. Существует ещё опция -iname, которая соответствует опции -iregex, игнорирующая символы в шаблоне:

$ find /etc -iregex '.*xt.*' /etc/X11/xkb/types/extra /etc/X11/xkb/semantics/xtest /etc/X11/xkb/compat/xtest /etc/X11/app-defaults/XTerm /etc/X11/app-defaults/XTerm-color

Советую обратить внимание на то, что в отличие от других программ, find требует прописывания шаблона для выборки полностью, а не частично. По этой причине необходимо добавлять .* в начале и конце запроса. В противном случае, слово xt в примере выше просто не сработало бы как регулярное выражение.

Find и типы

Опция -type позволяет искать файлы конкретного типа. Возможные для этой опции аргументы следующие: -b (блочные устройства), -c (символьные устройства), -d (каталоги), -p (именованный канал), -f (обычный файл), -l (символическая ссылка), -s (сокет). Поищем ради интереса символические ссылки в которых есть указание на слово vim:

$ find /usr/bin -name '*vim*' -type l /usr/bin/rvim /usr/bin/vimdiff /usr/bin/gvimdiff

Find и mtime
Опция -mtime позволяет искать файлы по времени их модификации. Аргументом для опции -mtime является период в 24 часа. Наиболее полезный результат можно получить, если указать после него символ "+" (что означает "после") или символ "-" (что означает "до"). Чтобы лучше понять как это работает, сделаем следующее:

$ ls -l ? -rw------- 1 root root 0 Jan 7 18:00 a -rw------- 1 root root 0 Jan 6 18:00 b -rw------- 1 root root 0 Jan 5 18:00 c -rw------- 1 root root 0 Jan 4 18:00 d $ date Mon May 7 18:14:52 EST 2003

Теперь поищем файлы которые были созданы за прошедшие 24 часа:

$ find . -name \? -mtime -1 ./a

Или поищем файлы, которые были созданы за текущие сутки:

$ find . -name \? -mtime +0 ./b ./c ./d

Опция -daystart

Если вы дополнительно укажете опцию -daystart, то можно посмотреть файлы у которых период "сегодня" не ограничивается 24-ю последними часами. Давайте для примера посмотрим, есть ли у нас файлы которые были созданы вчера и позавчера:

$ find . -name \? -daystart -mtime +0 -mtime -3 ./b ./c $ ls -l b c -rw------- 1 root root 0 May 6 18:00 b -rw------- 1 root root 0 May 5 18:00 c

Опция -size

Опция -size позволяет искать файлы по их размеру. По умолчанию, опция -size задаёт размер блока от 512 байт, но добавление нужного суффикса легко решает эту проблему. Доступные суффиксы: 'b' (512-байтные блоки), 'c' (байты), 'k' (килобайты), и 'w' (2-байтные слова). В дополнение к этому можно указать дополнительные символы "плюс" (означающий "больше чем") и "минус" (означающий "меньше чем").
Примера ради попробуем поискать файлы в каталоге /usr/bin размером менее 50 байт:

$ find /usr/bin -type f -size -50c /usr/bin/krdb /usr/bin/run-nautilus /usr/bin/sgmlwhich /usr/bin/muttbug

Обработка найденных файлов

А теперь, давайте представим, что можно сделать с тем, что мы нашли! Ну, к примеру, команда find имеет возможность обрабатывать найденные файлы с помощью опции -exec. Данная опция позволяет принимать командную строку для её выполнения в качестве аргумента заканчивающегося символом ";" и заменяющий любые встречающиеся совпадения именем файла.
Чтобы немного понять как это работает, продемонстрируем следующий пример:

$ find /usr/bin -type f -size -50c -exec ls -l '{}' ';' -rwxr-xr-x 1 root root 27 Oct 28 07:13 /usr/bin/krdb -rwxr-xr-x 1 root root 35 Nov 28 18:26 /usr/bin/run-nautilus -rwxr-xr-x 1 root root 25 Oct 21 17:51 /usr/bin/sgmlwhich -rwxr-xr-x 1 root root 26 Sep 26 08:00 /usr/bin/muttbug

Как видно, find очень мощная команда. Она выросла за многие годы благодаря развитию UNIX и Linux. У этой команды есть ещё множество полезных опций, но чтобы их узнать рекомендуем обратиться к man-странице по этой программе.

Команда locate

Мы уже изучили такие команды как "which" , "whereis" , и "find". Мы также знаем, что команда find может потребовать изрядное количество времени на выполнение, особенно, когда нужно просмотреть одну директорию за другой во время поиска. Тут мы открываем для себя команду locate, которая ускоряет поиск за счёт использования внешней базы данных, которая генерируется с помощью updatedb (её рассмотрим чуть позже).
Команда locate ищет совпадения по любой части имени пути, но не по файлам. Делает это примерно так:

$ locate bin/ls /var/ftp/bin/ls /bin/ls /sbin/lsmod /sbin/lspci /usr/bin/lsattr /usr/bin/lspgpot /usr/sbin/lsof

Использование updatedb

Подавляющее большинство Linux-систем имеют задание в планировщике cron для периодического обновления базы данных. Если locate возращает ошибку наподобие той, что в примере, то вам потребуется запустить locatedb из под суперпользователя root для генерации новой базы:

$ locate bin/ls locate: /var/spool/locate/locatedb: No such file or directory $ su - Password: # updatedb

Данная команда может потребовать достаточно долгого времени на выполнение операции. При наличии шумного жёсткого диска вы, ко всему прочему, получите ещё больше шума во время индексации файловой системы :).

Команда slocate

В большинстве дистрибутивов Linux команда locate заменена командой slocate. Обычно это символическая ссылка на команду locate, так что вам не потребуется запоминать что вы там используете. Slocate означает "secure locate". Она сохраняет разрешения для доступа к файлам в своей базе данных, поэтому непривилегированные пользователи не смогут прочесть те каталоги куда у них нет доступа. Принцип использования slocate абсолютно тот же самый как и у locate, хоть и вывод у этих программ немного отличается.

Контроль за процессами

Запускаем xeyes

Чтобы побольше узнать о контроле за процессом, нам его надо сначала создать. Убедитесь, что вы запустили среду X и выполните такую команду:

$ xeyes -center red

Вы увидите всплывшее окно приложения xeyes и красные глаза следящие за вашим курсором мыши бегающим по экрану. Можете также обратить внимание на то, что после запуска программы у вас не появилось нового приглашения в командной строке.

Останов процесса

Чтобы вернуть строку приглашения снова, вам необходимо нажать сочетание клавиш Control-C (часто записывается как Ctrl-C или ^C).
Вы получите новое приглашение среды bash, но программа xeyes исчезнет. Это означает, что данное приложение было *убито*. Вместо убийства приложения сочетанием клавиш Control-C можно программу просто остановить сочетанием Control-Z:

$ xeyes -center red Control-Z [1]+ Stopped xeyes -center red $

Мы снова попали в строку приглашения bash и программа xeyes остановилась. Можно с этим немножко поиграться, но прошу заметить, что программа "зависла" в одном месте. Если окно xeyes будет скрыто другим окном, а потом снова раскрыто, то обнаружили бы, что глаза не перерисовываются. Процесс ничего не делает. Он - стоит.

Команды fg и bg

Чтобы "раз-стопить" процесс и запустить его снова, нам необходимо его восстановить и вытащить на передний план с помощью команды fg:

$ fg (test it out, then stop the process again) Control-Z [1]+ Stopped xeyes -center red $

Теперь уберём его в фоновый режим с помощью bg:

$ bg [1]+ xeyes -center red & $

Круто! Процесс снова заработал в фоновом режиме, а мы получили новое приглашение командной строки.

Используем символ &

Если мы желаем запустить xeyes в фоновом режиме сразу (не используя Control-Z и bg), нам всего лишь надо добавить символ "&" (амперсанд) в конце команды:

$ xeyes -center blue & [2] 16224

Множественные фоновые процессы

Теперь у нас в фоне запущено несколько процессов xeyes с красными и синими глазами. Просмотреть эти процессы мы можем с помощью встроенной в bash команды jobs.

$ jobs -l [1]- 16217 Running xeyes -center red & [2]+ 16224 Running xeyes -center blue &

Цифра в левой колонке - это номер задачи назначенной оболочкой bash. Задача номер два имеет символ "+" после номера задачи, что означает, что задача является "текущей", что в свою очередь означает, при вводе команды fg вывод этой задачи на передний план. Также вы можете вывести на передний план любую другую задачу указав её номер. Например, fg 1 вытащит окно xeyes с красным цветом глаз на передний план. Следующая колонка - идентификатор процесса или pid, включающий в себя опцию -l описывающий порядок расположения задач. И, наконец, обе задачи обозначены статусом "Running". Справа находится сама выполненная команда.

Введение в сигналы

Для убийства, останова или продолжения работы процесса в Linux введён специальный тип сообщений именуемый как "сигналы". Посылая конкретный сигнал процессу, вы можете прервать, остановить или сделать что-то ещё с процессом. То, что вы делали с помощью Control-C, Control-Z, команд fg и bg - использовали bash для посылки определённых сигналов процессу. Эти же сигналы можно посылать с помощью команды kill с указанием идентификатора процесса в командной строке:

$ kill -s SIGSTOP 16224 $ jobs -l [1]- 16217 Running xeyes -center red & [2]+ 16224 Stopped (signal) xeyes -center blue

Как вы здесь все видите, команда kill совершенно необязательно "убивает" процесс. Используя опцию -s kill может посылать любой сигнал процессу. Linux убивает, останавливает или восстанавливает процессы посылая сигналы SIGINT, SIGSTOP, или SIGCONT. Имеются и другие сигналы, которые вы можете послать процессу. Некоторые из них интерпретируются в зависимости от приложения. Узнать о сигналах побольше можно с помощью соответствующей страницы документации в разделе SIGNALS.

SIGTERM и SIGINT

Если вы хотите убить процесс, то для этого существует несколько возможностей. по умолчанию, kill посылает SIGTERM, который не равен SIGINT, который получается при нажатии Control-C, хотя результат аналогичен:

$ kill 16217 $ jobs -l [1]- 16217 Terminated xeyes -center red [2]+ 16224 Stopped (signal) xeyes -center blue

Большое убийство

Процессы способны игнорировать SIGTERM и SIGINT по выбору, или потому, что они остановились, или по причине зависания. В таких случаях нужна кувалда в виде сигнала с именем SIGKILL. Ни один процесс не способен проигнорировать SIGKILL:

$ kill 16224 $ jobs -l [2]+ 16224 Stopped (signal) xeyes -center blue $ kill -s SIGKILL $ jobs -l [2]+ 16224 Interrupt xeyes -center blue

nohup

Терминал в котором вы посылали команду запустить тот или иной процесс называется контрольным терминалом. Некоторые оболочки (не только bash) будут отправлять сигнал SIGHUP в фоновом режиме при попытке выйти из сеанса. Чтобы этого не случилось, необходимо защитить процесс командой nohup:

$ nohup make & [1] 15632 $ exit

Использование ps для просмотра списка процессов

Команда jobs позволяет просматривать задачи запущенные только с момента запуска вашей сессии bash. Чтобы увидеть все процессы в системе, необходимо использовать команду ps с опциями "a" и 'x":

$ ps ax PID TTY STAT TIME COMMAND 1 ? S 0:04 init [3] 2 ? SW 0:11 [keventd] 3 ? SWN 0:13 [ksoftirqd_CPU0] 4 ? SW 2:33 [kswapd] 5 ? SW 0:00 [bdflush]

Здесь показаны лишь самые первые строчки весьма длинного списка. Это даёт вам картину того, что творится на вашей машине, но такое обилие информации тоже нехорошо. Если вы уберёте опцию "ax", то вам будут показаны опции владельцем которых являетесь только вы и контрольный терминал. Если вы используете "ps a", то увидите список всех процессов подключеных к терминалам.

Гуляем в лесу, глядим на деревья

Также вы можете получить различную информацию об отдельно взятом процессе. Опция --forest позволяет легко получить иерархию процессов в системе, что даст вам информацию о том, как процессы связаны друг с другом. Когда процесс начинает новый процесс, то получившийся процесс называется "дочерним". При использовании опции --forest мы увидим, что родители находятся левее, а дети - правее:

$ ps x --forest PID TTY STAT TIME COMMAND 927 pts/1 S 0:00 bash 6690 pts/1 S 0:00 _ bash 26909 pts/1 R 0:00 _ ps x --forest 19930 pts/4 S 0:01 bash 25740 pts/4 S 0:04 _ vi processes.txt

Опции "u" и "l" у команды ps

Опции "u" и "l" могут быть использованы совместно с любой комбинацией, в том числе, с уже упомянутыми "ax". Это даёт ещё кое-какую дополнительную информацию о процессах:

$ ps au USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND agriffis 403 0.0 0.0 2484 72 tty1 S 2001 0:00 -bash chouser 404 0.0 0.0 2508 92 tty2 S 2001 0:00 -bash root

408 0.0 0.0 1308 248 tty6 S 2001 0:00 /sbin/agetty 3 agriffis 434 0.0 0.0 1008 4 tty1 S 2001 0:00 /bin/sh /usr/X chouser 927 0.0 0.0 2540 96 pts/1 S 2001 0:00 bash $ ps al F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND 100 1001 403 1 9 0 2484 72 wait4 S tty1 0:00 -bash 100 1000 404 1 9 0 2508 92 wait4 S tty2 0:00 -bash 000 0 408 1 9 0 1308 248 read_c S tty6 0:00 /sbin/ag 000 1001 434 403 9 0 1008 4 wait4 S tty1 0:00 /bin/sh 000 1000 927 652 9 0 2540 96 wait4 S pts/1 0:00 bash

Использование утилиты top

Если вы хотя бы пару раз запускали команду ps несколько раз подряд, то, наверное, вы наверняка пытались найти там какие-либо отличия в выхлопе. Вообще-то, для таких вещей используется top. Команда "top" отображает постоянно обновляющиеся данные с полезной общей информацией:

$ top 10:02pm up 19 days, 6:24, 8 users, load average: 0.04, 0.05, 0.00 75 processes: 74 sleeping, 1 running, 0 zombie, 0 stopped CPU states: 1.3% user, 2.5% system, 0.0% nice, 96.0% idle Mem: 256020K av, 226580K used, 29440K free, 0K shrd, 3804K buff Swap: 136544K av, 80256K used, 56288K free 101760K cached PID USER PRI NI SIZE RSS SHARE STAT LIB %CPU %MEM TIME COMMAND 628 root 16 0 213M 31M 2304 S 0 1.9 12.5 91:43 X 26934 chouser 17 0 1272 1272 1076 R 0 1.1 0.4 0:00 top 652 chouser 11 0 12016 8840 1604 S 0 0.5 3.4 3:52 gnome-termin 641 chouser 9 0 2936 2808 1416 S 0 0.1 1.0 2:13 sawfish

Использование nice

Каждый процесс в Linux имеет свой приоритет использования разделяемого процессорного времени. Вы можете задать приоритет процессу запустив его с помощью команды nice:

$ nice -n 10 oggenc /tmp/song.wav

Момент выставления приоритета процессу называется "nice", где легко запомнить, что чем выше число nice, тем больше приоритет у этого процесса. По умолчанию, процессы запускаются со значением nice равным нулю. Таким образом, если я присвою процессу oggenc значение 10, то приоритет использования процессора у него будет выше, чем у прочих приложений. Уровень niceness приложений можно посмотреть в столбце NI в листинге чуть выше.

Использование renice

Команда nice способна менять приоритет программы только при запуске программы. Если вы хотите изменить приоритет уже запущенной программы, то тут потребуется команда renice:

$ ps l 641 F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND 000 1000 641 1 9 0 5876 2808 do_sel S ? 2:14 sawfish $ renice 10 641 641: old priority 0, new priority 10 $ ps l 641 F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND 000 1000 641 1 9 10 5876 2808 do_sel S ?

Обработка текста

Возвращаясь к перенаправлению

В одной из ранних частей данного руководства мы рассматривали использование оператора ">" для перенаправления вывода потока в файл:

$ echo "firstfile" > copyme

В дополнение к перенаправлению файлов мы можем использовать такую мощную вещь как каналы (пайпы). Используя каналы, мы можем, например пропускать вывод одной команды на вход другой. Для примера попробуем сделать следующее:

$ echo "hi there" | wc 1 2 9

Здесь, символ "|" (прямой черты) используется для связи с команды расположенной слева с командой расположенной справа. В этом примере команда "echo" выводит на экран строку "hi there". Данный вывод попадает на обычный терминал, но перенаправляется при этом в канал, где команда wc подсчитывает строк, слов и общее количество букв в выводе.

Пример канала

Покажем вам другой простой пример:

$ ls -s | sort -n

В вышеприведённом случае команда "ls -s" выводит обычный листинг содержимого каталога на экран терминала, но поскольку мы перенаправили вывод в канал, который в свою очередь вызывает "sort -n", то он сортирует вывод по номерам. Воистину удобно для сортировки файлов в вашем домашнем каталоге!

Конвейер для распаковки

Для того, чтобы распаковать файл с архивом, в обычном случае нам приходится проделывать такую операцию:

$ bzip2 -d linux-2.4.16.tar.bz2 $ tar xvf linux-2.4.16.tar

Метод ниже покажем вам, как можно получить распакованный архив без создания промежуточного файла на диске. Программа tar умеет читать данные на входе (вместо скармливания ей файла), поэтом мы получим тот же результат с помощью конвейера:

$ bzip2 -dc linux-2.4.16.tar.bz2 | tar xvf -

О, да! Мы распаковали архив с тарболлом без создания промежуточного файла.

Длинные конвейеры

Давайте рассмотрим пример такого конвейера:

$ cat myfile.txt | sort | uniq | wc -l

Мы использовали команду cat, чтобы скормить вывод содержимого файла с именем myfile.txt команде sort, Затем, когда sort получила данные на вход, она сортирует все полученные данные в алфавитном порядке, после чего пересылает эти данные команде uniq. Uniq, в свою очередь, удаляет дубликаты строк (и заранее требует рассортированных входных данных, кстати), отсылает обработанный вывод далее, команде "wc -l". Ранее мы уже видели работу команды wc, но без каких-либо заданных опций. В этом случае, мы добавляем опцию -l, которая печатает общую сумму количества строк в выводе не включая туда количество слов и букв. Вы можете увидеть, что данный конвейер выводит количество уникальных (не повторяющихся) строк в текстовый файл.
Попроуйте интереса ради создать пару проверочных файлов в вашем любимом текстовом редакторе и используйте конвейеры, чтобы увидеть получившийся результат.

Давайте пробежимся по обработке текста

Теперь бегло пройдёмся по обзору стандартных утилит Linux, предназначенных для обработки текстовых команд. Поскольку мы и так рассмотрели весьма немалый объём информации в этом руководстве, то приводить примеры мы не будем, поскольку не хватит никакого места, чтобы давать примеры для каждой команды. Вместо этого мы рекомендуем почитать справку по каждой отдельно взятой команде (например, напечатав в терминале man echo), изучить команды и их опции поигравшись с ними. Но как правило, большинство этих команды выводят на экран терминала результат обработки некоего текста, вместо изменения файлов. После прохождения нашего беглого тура по стандартным командам Linux для обработки текстов, мы взглянем немного поближе на перенаправление ввода/вывода. И да, мы наконец-то увидим свет в конце туннеля. :)

Команда 'echo' печатает свои аргументы в терминале. Используйте опцию -e, если хотите встроить в конструкцию последовательность в которой есть символ обратной косой черты. Например, вот так: echo -e "foo\nfoo" выведет на экран слово foo, затем, новую строчку и слово foo снова. Используйте опцию '-n' дабы сказать команде echo игнорировать перевод на новую строку, который добавляется в вывод по умолчанию.

Команда 'cat' выведет содержимое файлов в качестве аргумента на терминал. Весьма удобна в использовании как первая команда в конвейере, например вот так: 'cat foo.txt | blah'.

Команда 'sort' выведет содержимое указанного файла в командной строке в алфавитном порядке. Разумеется, команда sort также может принимать данные из потока на вход. Введите man sort чтобы ознакомиться с различными возможностями управления сортировкой.

Команда 'uniq' обрабатывает только уже отсортированные ранее потоки с данными (через конвейер) и удаляет дубликаты строк.

Команда 'wc' выводит на экран количество слов, строк, и байт в указанном вами файле или с входного потока (используя конвейер). Введите man wc, чтобы узнать как настроить вывод подсчёта более удобным образом.

Команда 'head' печатает первые десять строк из файлав или потока. Используйте опцию -n, чтобы указать сколько строк необходимо вывести.

Команда 'tail' печатает последние десять строк из файла или потока. Используйте опцию -n, для указания количества строк, которые необходимо вывести на экран.

Команда 'tac' очень похожа на 'cat' по результату, только печатает все строки в обратном порядке. То бишь, последняя строчка из файла будет напечатана первой и т.д.

Команда 'expand' конвертирует табуляции в пробелы. Для использования, укажите опцию -t, чтобы указать точку для курсора.

Команда 'unexpand' конвертирует пробелы в табуляции. Для использования, укажите опцию -t, чтобы указать точку для курсора.

Команда 'cut' используется для получения символов-разделителей полей из каждой строки входного файла или потока.

Команда 'nl' добавляет номер перед каждой строчкой. Полезно при печати листингов программ, кстати.

'pr' используется для разделения одного файла на несколько страниц при выводе. Используется при выводе текста на принтер.

'tr' утилита для подмены символов. Необходима для поиска какого-то символа и подмены его другим в выходном потоке.

'sed' мощный потоковый текстовый редактор. Если хотите узнать о sed больше, то рекомендую изучить на сайте проекта Funtoo следующие статьи:

- Sed в примерах, часть 1 (англ.)
- Sed в примерах, часть 2 (англ.)
- Sed в примерах, часть 3 (англ.)

Если вы планируете сдавать экзамен LPI, то прочтение первых двух статей выше необходимо в обязательном порядке.

'awk' - это полезный язык предназначенный для обработки строк в тексте. Чтобу узнать об awk побольше, рекомендуем к прочтению следующие статьи на сайте проекта Funtoo:

* Awk в примерах, часть 1 (англ.) * Awk в примерах, часть 2 (англ.) * Awk в примерах, часть 3 (англ.)

Утилита 'od' разработана для конвертации входного потока в шестнадцатеричный формат, более известный как HEX.

'split' - это команда предназначенная для разделения больших файлов на кусочки поменьше.

'fmt' форматирует параграфы, размещая их по величине страницы. На сегодняшний день довольно бесполезная штука, учитывая, что это умеют все текстовые редакторы, но хотя бы будете знать.

'paste' берёт два или более файла на входе, объединяет линиями каждую строку и выдаёт результат. Полезна для создания таблиц или колонок в тексте.

'join' довольно похожа на команду paste, но она использует поле (обычно, первое по умолчанию) из каждой полученной строки

'tee' берёт данные со входа и отправляет их в файл и на экран одновременно. Очень удобно использовать, когда вы создаёте какой-то журнал, но при этом вам надо одновременно видеть происходящий процесс на экране.

Пробежка окончена! Изучаем перенаправление.

По аналогии с символом ">" мы можем использовать в командной оболочке bash символ "<", который необходим для перенаправления данных из файла в команду. Для подавляющего большинства команд достаточно просто указать имя файла в командной строке. однако, некоторые программы могут работать только со стандартным входом.

Bash, а также некоторые другие оболочки поддерживает концепцию известную как "herefile". Это позволяет вам вставлять команду прямо в строку идущую за вызовом команды, обрывая её неким(?) значением. Проще всего будет показать это вот на таком примере:

$ sort <

В примере выше мы напечатали слова как apple, cranberry и banana, со словом END в самом конце, обозначающее конец вывода. Программа вернула нам значение, но уже отсортированное в алфавитном порядке.

Используем >>

Будучи уже знакомым с символом ">>", вы наверняка думаете, что "<<" будет работать примерно аналогичным образом, но это не так. Это означает, что данная команда добавляет некий вывод в файл не перезаписывая его, как ">". Пример для понимания сказанного:

$ echo Hi > myfile $ echo there. > myfile $ cat myfile there.

Уй! Мы где-то потеряли слово "Hi!". Тогда мы поступим вот так:

$ echo Hi > myfile $ echo there. >> myfile $ cat myfile Hi there.

Уже намного лучше!

Модули ядра

Знакомьтесь с uname

Uname - это команда предоставляющая нам разнообразнейшую интересную информацию о вашей системе. Сейчас я вам покажу информацию с одной из моих рабочих станций используемых для разработки, которую я получаю когда ввожу команду uname -a. Она выведет мне всю доступную информацию в одну строку:

$ uname -a Linux inventor 2.4.20-gaming-r1 #1 Fri Apr 11 18:33:35 MDT 2003 i686 A

Больше безумств с uname

Давайте поглядим, какую же информацию может дать uname:

info. option arg example kernel name -s "Linux" hostname -n "inventor" kernel release -r "2.4.20-gaming-r1" kernel version -v "#1 Fri Apr 11 18:33:35 MDT 2003" machine -m "i686" processor -p "AMD Athlon(tm) XP 2100+" hardware platform -i "AuthenticAMD" operating system -o "GNU/Linux"

Захватывающе! А что команда uname -a выдаст у вас?

Номер выпуска ядра

А теперь - фокус-чпокус. Для начала, введите uname -r, дабы получить вывод версии текущего запущенного ядра.
Далее, найдите каталог /lib/modules и... вуаля! Вы находите такой каталог. Ну ладно, не фокус это, но тем не менее, настало время поговорить о значении каталога /lib/modules и объяснить, что же за такие модуля ядра там.

Ядро

Ядро Linux - это сердце всего того, что принято вообще называть Linux. Это некий кусок программного кода, получающий доступ к вашему аппаратному обеспечению напрямую, и предоставляющий некий слой абстракции, чтобы ваши старые добрые программы могли работать. Благодаря ядру, вашему текстовому редактору не нужно беспокоиться о том, что он пишет на SCSI или IDE-диск, или даже в оперативную память. Он пишет на некую файловую систему, а ядро всё остальное берёт на себя.

Введение в модули ядра

Так что же такое модули ядра? В общих чертах, это часть ядра, которая сохраняется в определённом формате на диске. С помощью ваших команд они могут быть загружены в работающее ядро и обеспечить дополнительный функционал.

Так как модули ядра загружаются только по требованию, то вы можете получить в ядре массу дополнительной функциональности, которую обычным путём включить не получается. И однажды эти модули могут оказаться крайне полезными, и подключаются они автоматически, чаще всего для того, чтобы добавить поддержку какой-нибудь странной файловой системы или устройства, которое мало где используется.

Модули ядра в примерах

Итак, модули ядра позволяют работающему ядру включить возможности по запросу. Без модулей вам придётся пересобирать полностью новое ядро и перезагружаться, чтобы получить поддержку чего-то нового.

lsmod

Чтобы посмотреть, какие же модули загружены в ядро нужно следующее:

lsmod Module Size Used by Tainted: PF vmnet 20520 5 vmmon 22484 11 nvidia 1547648 10 mousedev 3860 2 hid 16772 0 (unused) usbmouse 1848 0 (unused) input 3136 0 [mousedev hid usbmouse] usb-ohci 15976 0 (unused) ehci-hcd 13288 0 (unused) emu10k1 64264 2 ac97_codec 9000 0 [emu10k1] sound 51508 0 [emu10k1] usbcore 55168 1 [hid usbmouse usb-ohci ehci-hcd]

Список модулейМодули ядра

Знакомьтесь с uname

Uname - это команда предоставляющая нам разнообразнейшую интересную информацию о вашей системе. Сейчас я вам покажу информацию с одной из моих рабочих станций используемых для разработки, которую я получаю когда ввожу команду uname -a. Она выведет мне всю доступную информацию в одну строку:

$ uname -a Linux inventor 2.4.20-gaming-r1 #1 Fri Apr 11 18:33:35 MDT 2003 i686 A

Больше безумств с uname

Давайте поглядим, какую же информацию может дать uname:

info. option arg example kernel name -s "Linux" hostname -n "inventor" kernel release -r "2.4.20-gaming-r1" kernel version -v "#1 Fri Apr 11 18:33:35 MDT 2003" machine -m "i686" processor -p "AMD Athlon(tm) XP 2100+" hardware platform -i "AuthenticAMD" operating system -o "GNU/Linux"

Захватывающе! А что команда uname -a выдаст у вас?

Номер выпуска ядра

А теперь - фокус-чпокус. Для начала, введите uname -r, дабы получить вывод версии текущего запущенного ядра.
Далее, найдите каталог /lib/modules и... вуаля! Вы находите такой каталог. Ну ладно, не фокус это, но тем не менее, настало время поговорить о значении каталога /lib/modules и объяснить, что же за такие модуля ядра там.

Ядро

Ядро Linux - это сердце всего того, что принято вообще называть Linux. Это некий кусок программного кода, получающий доступ к вашему аппаратному обеспечению напрямую, и предоставляющий некий слой абстракции, чтобы ваши старые добрые программы могли работать. Благодаря ядру, вашему текстовому редактору не нужно беспокоиться о том, что он пишет на SCSI или IDE-диск, или даже в оперативную память. Он пишет на некую файловую систему, а ядро всё остальное берёт на себя.

Введение в модули ядра

Так что же такое модули ядра? В общих чертах, это часть ядра, которая сохраняется в определённом формате на диске. С помощью ваших команд они могут быть загружены в работающее ядро и обеспечить дополнительный функционал.

Так как модули ядра загружаются только по требованию, то вы можете получить в ядре массу дополнительной функциональности, которую обычным путём включить не получается. И однажды эти модули могут оказаться крайне полезными, и подключаются они автоматически, чаще всего для того, чтобы добавить поддержку какой-нибудь странной файловой системы или устройства, которое мало где используется.

Модули ядра в примерах

Итак, модули ядра позволяют работающему ядру включить возможности по запросу. Без модулей вам придётся пересобирать полностью новое ядро и перезагружаться, чтобы получить поддержку чего-то нового.

lsmod
Чтобы посмотреть, какие же модули загружены в ядро нужно следующее:

lsmod Module Size Used by Tainted: PF vmnet 20520 5 vmmon 22484 11 nvidia 1547648 10 mousedev 3860 2 hid 16772 0 (unused) usbmouse 1848 0 (unused) input 3136 0 [mousedev hid usbmouse] usb-ohci 15976 0 (unused) ehci-hcd 13288 0 (unused) emu10k1 64264 2 ac97_codec 9000 0 [emu10k1] sound 51508 0 [emu10k1] usbcore 55168 1 [hid usbmouse usb-ohci ehci-hcd]

Список модулей
Как видите, в моей системе запущено довольно много модулей. Модули vmnet и vmmon необходим для работы VMWare Workstation, позволяющие запускать виртуальные компьютеры на моём компьютере. Модуль nvidia принадлежит компании NVIDIA, позволяющий мне использовать возможности моей видеокарты под Linux, получая удовольствие от множества интересных функций.

Также у меня есть модули отвечающие за работу моих USB-устройств, имеющих название "mousedev," "hid," "usbmouse," "input," "usb-ohci," "ehci-hcd" и "usbcore." Очень часто есть смыл конфигурировать ядро с поддержкой модулей для USB-устройств. Почему? Просто устройства USB относятся к классу "Plug and Play", поэтому имея поддержку таких устройств в виде модулей, вы можете убрать старое устройство и взять новое, но модули не загрузятся до тех пор, пока вы его не установите. Такие вещи бывают очень нужны.

Сторонние модули
Заканчивают мой список модули с именами "emu10k1", "ac97_codec", и "sound", позволяющие работать моей звуковой карте Sound Blaster Audigy.

Следует заметить, что некоторые модули ядра идут вместе с исходными текстами оного. Например, все модули относящиеся к поддержке USB-устройств, скомпилированы из стандартных текстов входящих в поставку. Зато, например, модули nvidia, emu10k1 and VMWare взяты из других источников. Главная особенность модулей ядра Linux - возможность третьим сторонам предоставлять ту функциональность, которая необходима и "включить" эту функциональность в ядро Linux. Без перезагрузки, причём.

depmod и его друзья
В моём каталоге /lib/modules/2.4.20-gaming-r1/ лежат некоторые файлы, которые имеют в своём названии слово начинающееся на "modules":

$ ls /lib/modules/2.4.20-gaming-r1/modules.* /lib/modules/2.4.20-gaming-r1/modules.dep /lib/modules/2.4.20-gaming-r1/modules.generic_string /lib/modules/2.4.20-gaming-r1/modules.ieee1394map /lib/modules/2.4.20-gaming-r1/modules.isapnpmap /lib/modules/2.4.20-gaming-r1/modules.parportmap /lib/modules/2.4.20-gaming-r1/modules.pcimap /lib/modules/2.4.20-gaming-r1/modules.pnpbiosmap /lib/modules/2.4.20-gamingr1/modules.usbmap

Эти файлы содержат некоторое количество информации о зависимостях. В одном из них, есть запись "зависимости", которая содержит информацию для модулей - одни модули могут требовать другого запущенного модуля, чтобы запуститься. Именно подобного рода информация записана в этих файлов.

Как установить модули?

Некоторые модули ядра разработаны для того, чтобы работать с определённым оборудованием, наподобие emu10k1, предназначенного для моей звуковой карты SoundBlaster Audigy. Модули такого типа имеют записи которые хранят PCI ID и тому подобные идентифицирующие признаки того оборудование, которые они поддерживают.

Такая информация, например, используется в сценариях для горячего подключения (рассмотрим их несколько позже, в одном из руководств), чтобы автоматически определять оборудование и подгружать тот модуль, который необходим.

Используем depmod

Если вы вдруг установите новый модуль, то информация о зависимостях может устареть. чтобы избежать этого, обновите эту информацию снова, набрав depmod -a. Команда depmod просканирует весь список модулей в каталоге /lib/modules, затем обновить информацию о зависимостях. Делает он это сканируя содержимое файлов и находя специальные "символы" внутри модулей.

Местонахождение модулей

Так как всё же выглядят эти модули? Для ядер серии 2.4 они, обычно, лежат в /lib/modules и имеют расширение ".o". Для просмотра вообще всех имеющихся модулей, наберите следующее:

find /lib/modules -name '*.o' /lib/modules/2.4.20-gaming-r1/misc/vmmon.o /lib/modules/2.4.20-gaming-r1/misc/vmnet.o /lib/modules/2.4.20-gaming-r1/video/nvidia.o /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o /lib/modules/2.4.20-gaming-r1/kernel/fs/vfat/vfat.o /lib/modules/2.4.20-gaming-r1/kernel/fs/minix/minix.o [далее листинг обрезан для краткости]

insmod против modprobe

Так каким же образом можно загрузить модуль для ядра? Одним из способов является использование команды insmod и указание полного пути до модуля, который вам необходимо загрузить:

insmod /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o # lsmod | grep fat fat 29272 0 (unused)

Однако, в стандартном случае, модули запускаются с помощью команды modprobe. Одна из самых приятных возможностей этой программы является то, что она запускает модуль вместе с его зависимостями. ещё одним плюсом является то, что не требуется прописывать полный путь до модуля и указывать его расширение.

Команды rmmod and modprobe в деле

Давайте теперь выгрузим модуль "fat.o", а затем снова загрузим его используя modprobe:

rmmod fat # lsmod | grep fat # modprobe fat # lsmod | grep fat fat 29272 0 (unused)

Как можно увидеть, rmmod работает аналогично modprobe, только с обратным эффектом - выгружая модуль.

Познакомимся с друзьями: modinfo и modules.conf

Для того, чтобы узнать информацию об интересующих модулях, надо воспользоваться командой modinfo:

modinfo fat filename: /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o description:

Следует особо отметить наличие файла /etc/modules.conf. Этот файл соджержит конфигурационные настройки для программы modprobe. Здесь вы всегда можете тонко настроить modprobe "под себя", указав, какие модули надо загружать или выгружать, добавить запуск сценариев в определённых ситуациях и так далее.

Особенности файла modules.conf

Синтаксис и функционал файла modules.conf довольно сложен для понимания, посему останавливаться мы на нём не станем (если интересно, изучите справочную страницу man modules.conf), но есть вещи которые знать следует *в обязательно порядке*.

Во-первых, во многих дистрибутивах этот файл генерируется автоматически, из списка файлов другой директории, например, /etc/modules.d/. Например, Gentoo Linux имеет каталог /etc/modules.d/ и при запуске команды update-modules, она проверяет все файлы в /etc/modules.d/, генерируя при этом новый /etc/modules.conf. Так что если вы сделали какие-либо изменения в /etc/modules.d/, то запустите update-modules в Gentoo. В случае использования дистрибутива Debian всё будет практически аналогично, за тем исключением, что каталог называется /etc/modutils/.

Как видите, в моей системе запущено довольно много модулей. Модули vmnet и vmmon необходим для работы VMWare Workstation, позволяющие запускать виртуальные компьютеры на моём компьютере. Модуль nvidia принадлежит компании NVIDIA, позволяющий мне использовать возможности моей видеокарты под Linux, получая удовольствие от множества интересных функций.

Также у меня есть модули отвечающие за работу моих USB-устройств, имеющих название "mousedev," "hid," "usbmouse," "input," "usb-ohci," "ehci-hcd" и "usbcore." Очень часто есть смыл конфигурировать ядро с поддержкой модулей для USB-устройств. Почему? Просто устройства USB относятся к классу "Plug and Play", поэтому имея поддержку таких устройств в виде модулей, вы можете убрать старое устройство и взять новое, но модули не загрузятся до тех пор, пока вы его не установите. Такие вещи бывают очень нужны.

Сторонние модули

Заканчивают мой список модули с именами "emu10k1", "ac97_codec", и "sound", позволяющие работать моей звуковой карте Sound Blaster Audigy.

Следует заметить, что некоторые модули ядра идут вместе с исходными текстами оного. Например, все модули относящиеся к поддержке USB-устройств, скомпилированы из стандартных текстов входящих в поставку. Зато, например, модули nvidia, emu10k1 and VMWare взяты из других источников. Главная особенность модулей ядра Linux - возможность третьим сторонам предоставлять ту функциональность, которая необходима и "включить" эту функциональность в ядро Linux. Без перезагрузки, причём.

depmod и его друзья

В моём каталоге /lib/modules/2.4.20-gaming-r1/ лежат некоторые файлы, которые имеют в своём названии слово начинающееся на "modules":

$ ls /lib/modules/2.4.20-gaming-r1/modules.* /lib/modules/2.4.20-gaming-r1/modules.dep /lib/modules/2.4.20-gaming-r1/modules.generic_string /lib/modules/2.4.20-gaming-r1/modules.ieee1394map /lib/modules/2.4.20-gaming-r1/modules.isapnpmap /lib/modules/2.4.20-gaming-r1/modules.parportmap /lib/modules/2.4.20-gaming-r1/modules.pcimap /lib/modules/2.4.20-gaming-r1/modules.pnpbiosmap /lib/modules/2.4.20-gaming-r1/modules.usbmap

Эти файлы содержат некоторое количество информации о зависимостях. В одном из них, есть запись "зависимости", которая содержит информацию для модулей - одни модули могут требовать другого запущенного модуля, чтобы запуститься. Именно подобного рода информация записана в этих файлов.

Как установить модули?

Некоторые модули ядра разработаны для того, чтобы работать с определённым оборудованием, наподобие emu10k1, предназначенного для моей звуковой карты SoundBlaster Audigy. Модули такого типа имеют записи которые хранят PCI ID и тому подобные идентифицирующие признаки того оборудование, которые они поддерживают.

Такая информация, например, используется в сценариях для горячего подключения (рассмотрим их несколько позже, в одном из руководств), чтобы автоматически определять оборудование и подгружать тот модуль, который необходим.

Используем depmod

Если вы вдруг установите новый модуль, то информация о зависимостях может устареть. чтобы избежать этого, обновите эту информацию снова, набрав depmod -a. Команда depmod просканирует весь список модулей в каталоге /lib/modules, затем обновить информацию о зависимостях. Делает он это сканируя содержимое файлов и находя специальные "символы" внутри модулей.

Местонахождение модулей

Так как всё же выглядят эти модули? Для ядер серии 2.4 они, обычно, лежат в /lib/modules и имеют расширение ".o". Для просмотра вообще всех имеющихся модулей, наберите следующее:

find /lib/modules -name '*.o' /lib/modules/2.4.20-gaming-r1/misc/vmmon.o /lib/modules/2.4.20-gaming-r1/misc/vmnet.o /lib/modules/2.4.20-gaming-r1/video/nvidia.o /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o /lib/modules/2.4.20-gaming-r1/kernel/fs/vfat/vfat.o /lib/modules/2.4.20-gaming-r1/kernel/fs/minix/minix.o [далее листинг обрезан для краткости]

insmod против modprobe

Так каким же образом можно загрузить модуль для ядра? Одним из способов является использование команды insmod и указание полного пути до модуля, который вам необходимо загрузить:

insmod /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o # lsmod | grep fat fat 29272 0 (unused)

Однако, в стандартном случае, модули запускаются с помощью команды modprobe. Одна из самых приятных возможностей этой программы является то, что она запускает модуль вместе с его зависимостями. ещё одним плюсом является то, что не требуется прописывать полный путь до модуля и указывать его расширение.

Команды rmmod and modprobe в деле

Давайте теперь выгрузим модуль "fat.o", а затем снова загрузим его используя modprobe:

rmmod fat # lsmod | grep fat # modprobe fat # lsmod | grep fat fat 29272 0 (unused)

Как можно увидеть, rmmod работает аналогично modprobe, только с обратным эффектом - выгружая модуль.

Познакомимся с друзьями: modinfo и modules.conf

Для того, чтобы узнать информацию об интересующих модулях, надо воспользоваться командой modinfo:

modinfo fat filename: /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o description:

Следует особо отметить наличие файла /etc/modules.conf. Этот файл соджержит конфигурационные настройки для программы modprobe. Здесь вы всегда можете тонко настроить modprobe "под себя", указав, какие модули надо загружать или выгружать, добавить запуск сценариев в определённых ситуациях и так далее.

Особенности файла modules.conf

Синтаксис и функционал файла modules.conf довольно сложен для понимания, посему останавливаться мы на нём не станем (если интересно, изучите справочную страницу man modules.conf), но есть вещи которые знать следует *в обязательно порядке*.

Во-первых, во многих дистрибутивах этот файл генерируется автоматически, из списка файлов другой директории, например, /etc/modules.d/. Например, Gentoo Linux имеет каталог /etc/modules.d/ и при запуске команды update-modules, она проверяет все файлы в /etc/modules.d/, генерируя при этом новый /etc/modules.conf. Так что если вы сделали какие-либо изменения в /etc/modules.d/, то запустите update-modules в Gentoo. В случае использования дистрибутива Debian всё будет практически аналогично, за тем исключением, что каталог называется /etc/modutils/.

Заключение и ресурсы

Заключение

Мои поздравления! Вы только что закончили ещё одно руководство по основам администрирования операционной системы Linux. Надеемся, что это помогло вам подтянуть знания по основам Linux. Пожалуйста, изучите наше следующее руководство рассчитанное на средний уровень, которое будет рассматривать такие вещи, как модель разрешений и владельцев в Linux, создание файловых систем и их монтирование и многое другое. И помните, продолжая изучение этих руководств, вы готовите себя к сдаче экзмена на сертификат LPIC первого уровня от Linux Professional Institute.

LPI 101 (Часть третья)

Содержание

- 1 Перед тем как начать
- 2 Системная и сетевая документация
- 3 Модель разрешений в Linux
- 4 Управление учётными записями в Linux
- 5 "Тонкая" настройка пользовательского окружения
- 6 Заключение и ресурсы

Перед тем как начать

О данном руководстве

Добро пожаловать в "Непосредственное администрирование" третье из четырёх руководств, разработанное для подготовки вас к экзамену Linux Professional Institute 101 (второе издание). Данное руководство (третья его часть) прекрасно подходит тем, кто хочет улучшить основные навыки системного администрирования в Linux. Мы рассмотрим здесь различные темы, касающиеся системной и сетевой документации, модели разрешений в Linux, управление учётными записями пользователей и более тонкой настройке пользовательского окружения.

Если вы совсем не разбираетесь в Linux, то рекомендуем вам для начала ознакомиться с первой и второй частью данного руководства. Для большинства людей этот материал будет новым, но даже тем, кто уже является достаточно опытным пользователем Linux он может пригодиться для того, чтобы освежить свои знания в области администрирования системы.

По окончанию изучения этой серии материалов (всего их восемь, охватывающих экзамены LPI 101 и 102) у вас появятся знания позволяющие вам стать системным администратором Linux, и быть готовым к сдаче экзамена для получения сертификата LPIC 1-го уровня в Linux Professional Institute, если вы того пожелаете.

Системная и сетевая документация

Виды документации в системе

Имеется три основных вида документации в Linux-системе: справочные страницы, информационные страницы и идущая в поставке с приложением документация, которая располагается в /usr/share/doc. В данном разделе мы подробно рассмотрим все три вида источника информации.

Справочные страницы

Справочные страницы (больше известные как "manual pages" или маны) - это общепринятый вид документации в UNIX и Linux среде. В идеале, вы всегда можете посмотреть справочную страницу по любой команде, конфигурационному файлу, или базовой библиотеке. Хотя в реальности Linux это свободная система, поэтому пишут их далеко не всегда, или они являются устаревшими. Но так или иначе, справочные страницы это первое место, куда нужно обращаться за помощью.

Чтобы получить доступ к справочным страницам, достаточно ввести команду "man" по той теме, которая вам требуется, после чего откроется необходимый текст. Если вам потребуется выйти, то достаточно нажать клавишу "q". В качестве примера приведём ситуацию, когда вам требуется прочесть документацию по команде ls:

$ man ls

Для более быстрого получения необходимой вам информации, рекомендуем изучить структуру справочной страницы. В большинстве случаев, в man-странице имеются следующие разделы:

- NAME Название и краткое описание команды
- SYNOPSIS Как использовать команду
- DESCRIPTION Подробное описание функционала команды
- EXAMPLES Примеры использования команд
- SEE ALSO Смежные темы (обычно, тоже man-страницы)

Небольшое отступление. При включённой русской локали, данная документация может быть доступна полностью на русском языке.

Структура справочных страниц

Файлы включающие в себя справочные страницы содержатся в каталоге /usr/share/man (или как в некоторых устаревших системах /usr/man). Внутри данного каталога можно обнаружить следующую структуру по которой организованы данные страницы:

- man1 Пользовательские команды
- man2 Системные вызовы
- man3 Функции библиотек
- man4 Специальные файлы
- man5 Форматы файлов
- man6 Игры
- man7 Прочее

Справочные страницы со сложной структурой

Некоторые темы могут описываться в нескольких разделах. Дабы продемонстрировать вам подобное, давайте используем команду whatis, которая показывает все доступные справочные страницы по выбранной нами теме:

$ whatis printf printf (1) - format and print data printf (3) - formatted output conversion

В данном случае, команда man printf по умолчанию показывает раздел описывающий пользовательскую команду в разделе 1. Однако, если б мы писали программу на языке программирования Си, то нас бы явно заинтересовала стринца находящаясяв третьем разделе ("Библиотечные функции"). Вызвать необходимую страницу можно указав конкретный раздел при вводе в командной строке. Таким образом, чтобы получить информацию о printf(3), нам необходимо сделать следующее:

$ man 3 printf

Как найти правильную справочную страницу

Иногда бывает довольно проблематично сразу найти нужную документацию по выбранной теме. В этом случае можно попробовать воспользоваться командой man -k, которая смотрит заголовки справочных страниц в поле NAME. Обратите внимание, что данная команда работает с построчным поиском, так что если вы вдруг запустите что-нибудь вроде man -k ls, то у вас получится очень большой выхлоп!

Давайте поглядим пример запроса:

$ man -k whatis apropos (1) - search the whatis database for strings makewhatis (8) - Create the whatis database whatis (1) - search the whatis database for complete wo

Узнаём всё о команде apropos

Предыдущий рассмотренный пример показал нам кое-что новое. Во-первых, это apropos. Команда, которая по действию аналогична команде man -k (Я вам по секрету, кстати, даже больше скажу. Когда вы запускаете man -k, в фоне работает именно apropos). Во-вторых, следующим результатом поиска нам была показана команда makewhatis, которая сканирует вашу систему на наличие всех man-страниц и создаёт базу данных, которую потом используют команды whatis и apropos. В обычной ситуации, её надо периодически запускать от лица суперпользователя root, чтобы поддерживать её в актуальном состоянии:

makewhatis

Для бОльших подробностей о команде man и других полезных вещах, вам нужно ввести в консоли следующее:

$ man man

Переменная MANPATH

По умолчанию команда man ищет документацию в /usr/share/man, /usr/local/man, /usr/X11R6/man и в некоторых случаях /opt/man. В определённых случаях вам бывает необходимо добавить дополнительный путь для поиска документации. В этом случае, вам достаточно просто отредактировать файл /etc/man.conf в текстовом редакторе и добавить строку вроде такой:

MANPATH /opt/man

С этого момента будут находиться любые справочные страницы которые лежат в каталоге /opt/man/man*. И помните, что вам будет необходимо запустить makewhatis, чтобы добавить новые man-страницы в базу данных whatis.

GNU info

Одним из недостатков man-страниц является то, что они не поддерживают гиперссылки, поэтому вы не можете просто так переходить от одного раздела к другому. Ребята из проекта

GNU учли этот недостаток, и посему они разработали другой формат документации - "информационные" страницы. Многие из GNU-шных программ идут с обширной документацией в виде этих информационных страниц. Для того, чтобы их прочесть, достаточно набрать команду info:

$ info

Вызов команды info запускает отображение индекса всех info-страниц в системе. Вы можете перемещаться туда-сюда используя клавиши со стрелками, ходить по ссылкам (помеченные знаком звёздочки) используя клавишу Enter и выходить по клавише q. Это основные клавиши редактора Emacs. Так что можете легко ориентароваться во всём этом, если знакомы с данным текстовым редактором. Для введения Emacs можно посмотреть сайт IBM DeveloperWorks, где есть статья-руководство "Living in Emacs". (англ.)

Можно также посмотреть конкретную info-страницу с помощью команд:

$ info diff

Для более подробной информации по использованию команды info, попробуйте прочесть информационную страницу по этой команде. Для навигации по ней достаточно использовать те клавиши, которые я уже указывал:

$ info info

Каталог /usr/share/doc

Это последнее место, где можно отыскать справку в вашей системе. Многие программы идут с дополнительной документацией в разных форматах: текстовом, pdf, PostScript, HTML и других. Взгляните в каталог /usr/share/doc (или в /usr/doc/ на старых системах). Там вы найдёте обширный список директорий, каждая из которых относится к определённой программе установленной в вашей системе. Поиск среди этой документации частенько помогает найти такое, что не удаётся откопать с помощью справочных или информационных страниц. Например, подробные руководства или дополнительную техническую информацию. Беглый осмотр показывает просто прорву материала для чтения:

$ cd /usr/share/doc $ find . -type f | wc -l 7582

Хе-хе-хе! Даю вам в качестве домашнего задания задачу - прочитать хотя бы половину (3791) из этого материала. Начинать можете с завтрашнего дня. :-)

Проект Linux Documentation

В дополнение к документации идущей с системой, имеется определённое количество прекрасных ресурсов посвящённых ОС Linux в сети Интернет. Проект Linux Documentation - это группа добровольцев, которые вместе занимаются созданием полной и свободной документации по операционной системе Linux. Проект работает объединяя разрозненные куски документации по Linux в одном месте и организовывая простой и удобный поиск по ним.

Обзор LDP

Проект LDP состоит из нескольких разделов:

- Учебники и руководства - подробные, длинные книги вроде "Руководства программиста под Linux" (англ.)
- HOWTO - сборник рецептов по каким-то конкретным темам. Например, DSL HOWTO
- FAQ - сборник ответов на самые часто задаваемые вопросы.
- Man-страницы - справка по конкретным командам (то же самое, что можете увидеть введя команду man у себя в системе).

Если не знаете в каком разделе смотреть, то можете воспользоваться поиском, который покажет вам необходимые результаты по темам.

LDP кроме того поддерживает список ссылок и ресурсов, например, на LinuxGazette (см. ссылки в разделе "Ресурсы" в конце этой статьи) и Linux Weekly News, а также ссылки на почтовые рассылки и архивы новостей.

Почтовые рассылки

Почтовые рассылки решают, пожалуй, наиболее важную часть взаимодействия между разработчиками Linux. Очень часто проекты разрабатываются контрибуторами, которые живут в разных местах, нередко даже в разных частях земного шара. Почтовые рассылки позволяют разработчикам связываться с другими и поддерживать дискуссии используя электронную почту. Одна из самых знаменитых рассылок - Linux Kernel Mailing List aka LKML.

Подробнее о рассылках

Помимо разработки, почтовые рассылки могут являться способом для того, чтобы задавать вопросы и получать ответы на интересующие вопросы от именитых разработчиков или других пользователей. К примеру, определённые дистрибутивы всегда имеют рассылку для новичков. Мы можете посмотреть информацию во Всемирной паутине и найти нужную рассылку по вашему дистрибутиву.

Если вы потратите немного времени на чтение FAQ в уже упомянутой нами рассылке LKML, то обратите внимание, что там предупреждают новых подписчиков о том, что существуют вопросы на которые вы не сможете получить быстрый ответ. Ещё более разумным будет использование поиска по архиву рассылки, перед написанием вопроса. Это может помочь вам и сэкономит время, в том числе и ваше.

Группы новостей

Группы новостей Интернете очень похожи на почтовые рассылки, с той разницей, что они используют протокол NNTP (Network News Transfer Protocol) вместо электронной почты. Чтобы подписаться, вам необходимо использовать NNTP-клиент вроде 'slrn' или 'pan'. Основное её преимущество в том, что вы можете выбрать только нужную часть дискуссии, вместо того, чтобы забивать ваш почтовый ящик ей целиком. :)

Изучение групп новостей следует начать в первую очередь с comp.os.linux. Более полный список можно посмотреть на сайте LDP.

Сторонние сайты производителей

Веб-сайты различных дистрибутивов Linux часто имеют обновляемый набор документации, инструкций по установке, списков совместимости оборудования и других видов поддержки, вроде поиска по базе знаний. Например, можете посмотреть сайты:

- RedHat Linux
- Debian Linux
- Gentoo Linux
- SuSE Linux
- Turbolinux

Производители программного и аппаратного обеспечения

Многие производители программного и аппаратного обеспечения добавили поддержку Linux в свои продукты за последние несколько лет. На этих сайтах вы можете найти информацию о том, какое аппаратное обеспечение поддерживает Linux, программы и инструменты для разработки, исходники, скачать драйверы для операционной системы Linux для специфического оборудования и многое другое. Примеры:

- IBM и Linux
- HP и Linux
- Sun и Linux
- Oracle и Linux

Модель разрешений в Linux

Один пользователь, одна группа

В этом разделе мы рассмотрим модель разрешений и владельцев в операционной системе Linux. Мы уже знаем, что каждый файл используется определённым пользователем и группой. Это самая основа модели разрешений в Linux. Посмотреть имя пользователя и группу у конкретного файла можно с помощью команды ls -l:

$ ls -l /bin/bash -rwxr-xr-x 1 root wheel 430540 Dec 23 18:27 /bin/bash

В данном примере исполняемый файл /bin/bash принадлежит пользователю root и группе wheel. Модель разрешений в Linux работает поддерживая три независимых уровня для каждого объекта в файловой системе: владелец, группа, остальные пользователи.

Разбираемся в команде ls -l

Давайте посмотрим на наш вывод команды ls -l и изучим первую колонку в листинге:

$ ls -l /bin/bash -rwxr-xr-x 1 root wheel 430540 Dec 23 18:27 /bin/bash

Первое поле *-rwxr-xr-x* содержит символьную расшифровку разрешений обычного файла. Первый символ "-" - поле описывающее, что это самый обычный файл. В других случаях возможны варианты:

'd' директория 'l' символическая ссылка 'c' символьное устройство 'b' блочное устройство 'p' очередь FIFO 's' сокет

Триплеты

$ ls -l /bin/bash -rwxr-xr-x 1 root wheel 430540 Dec 23 18:27 /bin/bash

Следующим в рассматриваемом нами поле являются три символа именуемые "триплетами". Первый триплет обозначает разрешения для владельца файла, второй - для группы. Третий триплет рассматривает разрешения для всех остальных пользователей не входящих в указанную группы:

"rwx" "r-x" "r-x"

Итак, символ r выше обозначает разрешённое чтение (просмотр содержимого файла), w - его запись (и, как правило, удаление), а x - выполнение (то есть, запуск программы). Теперь используя всю полученную информацию, мы можем понять, что все могут читать и запускать данный файл, но правом записи в него обладает только суперпользователь root. Вот почему обычные пользователи могут его копировать, но только root имеет право его удалять и записывать в него.

Кто я?

Перед тем как мы изучим смену пользователя и группы у файла, давайте поглядим, как можно узнать ваш текущий идентификатор и группу в которую вы входите. Если вы не используете такую команду как su часто, то вы и так узнаете свой идентификатор при входе в систему. Если же вы используете su слишком часто, то в какой-то момент можете забыть из под какого же пользователя вы сидите. Чтобы узнать это, наберите команду whoami:

whoami root # su drobbins $ whoami drobbins

В какой же я группе?

Чтобы узнать к акой группе вы принадлежите, наберите команду groups:

$ groups drobbins wheel audio

В примере выше видно, что я нахожусь в группе drobbins, wheel и audio. Если же вы хотите узнать в каких группах находятся другие пользователи, то надо всего лишь указать в аргументах команды их имена:

$ groups root daemon root: root bin daemon sys adm disk wheel floppy dialout tape video daemon: daemon bin adm

Смена владельца, группы и пользователя

Чтобы изменить владельца или группу у какого-либо файла или иного объекта в файловой системе, необходимо использовать команды chown и chgrp. После каждой команды можно указать имя одного или нескольких файлов.

chown root /etc/passwd # chgrp wheel /etc/passwd

Впрочем, никто не мешает сменить одновременно владельца и группу одновременно с помощью одной команды chown:

chown root:wheel /etc/passwd

Вы не сможете использовать chown не являясь при этом суперпользователем, но команда chgrp работает из под любого запущенного пользователя и меняет владельца файла или группу на ту, к которой принадлежит пользователь запустивший её.

Рекурсивная смена владельца

Обе команды chown и chgrp имеют опцию -R, которую можно использовать для рекурсивной смены владельца по всему дереву каталогов. Например, так:

chown -R drobbins /home/drobbins

Введение в команду chmod:

Команды chown и chgrp нужны для смены владельцев и группы любого объекта в файловой системе. но существует другая программа - chmod, которая меняет разрешения "rwx", которые вы видите в выхлопе команды ls -l. chmod допускает два и более аргумента: "режим" описывающий какие разрешения будут изменены для одного или нескольких файлов:

$ chmod +x scriptfile.sh

В примере выше наш "режим" - +x. Как вы уже наверное догадались, режим +x сообщает команде chmod, что надо указанный файл требуется сделать исполняемым для всех. Если вы хотите убрать все разрешения на запуск, достаточно сделать следующее:

$ chmod -x scriptfile.sh

Разбиение пользователей/групп/прочих

Итак, в примерах использования команды chmod выше, мы применили разрешения для всех триплетов сразу - для пользователей, группы и всех остальных. но очень часто бывает необходимо изменить права для одного или только двух триплетов сразу. Чтобы сделать это, достаточно указать буквенное обозначение на необходимый триплет, который хотите изменить, до указания символа "+" или "-". Используйте символ "u" для указания триплета пользователя, символ "g" - для указания триплета группы и символ "o" для указания триплета "все остальные":

$ chmod go-w scriptfile.sh

Мы только что удалили разрешение на запись для группы и остальных пользователей, но оставили права владельца нетронутыми.

Сброс разрешений

В дополнение к смене бита разрешений, мы также можем их все сбросить. Используя оператор =, мы говорим команде chmod сбросить только разрешения и ничего более:

$ chmod =rx scriptfile.sh

В примере выше мы указали, что требуется установить биты "запись" и "выполнение", но сбросить при этом все биты "запись". Если вы хотите сбросить какой-то определённый триплет, то необходимо указать символьное имя для него перед символом знака "равно":

$ chmod u=rx scriptfile.sh

Цифровые режимы

До сего момента мы использовали символьные режимы для указания смены определённого разрешения с помощью chmod. Однако существует другой способ указания требуемых разрешений: используя 4-х разрядный шестнадцатеричный код. Используя синтаксис, известный как "цифровые разрешения", где каждый разряд указывает на соответствующий триплет. Например, числа 1777, 777 устанавливают флаги доступа к файлам в тот вид, который мы обсуждали в примерах выше. Цифра 1 здесь устанавливает специальный разрешающий бит, который мы будем рассматривать немного погодя (см. раздел "Незаметный первый разряд" в конце главы). Таблица ниже покажет, как надо расшифровывать последние три из четырёх разрядов:

Режим Разряд rwx 7 rw- 6 r-x 5 r-- 4 -wx 3 -w- 2 --x 1 --- 0

Синтаксис для цифровых разрешений

Использование синтаксиса для указания разрешений в цифровом виде бывает крайне полезен в ситуации, когда необходимо прописать все разрешения для определённого файла, как в примере ниже:

$ chmod 0755 scriptfile.sh $ ls -l scriptfile.sh -rwxr-xr-x 1 drobbins drobbins 0 Jan 9 17:44 scriptfile.sh

в этом примере мы использовали редим 0755, который полностью соответствует устанавливаемому разрешению вида -rwxr-xr-x.

Значение umask

Когда некий процесс создаёт файл, то он указывает те права доступа, которые ему нужны. Чаще всего задаются права доступа 0666, которые разрешают запись и стение для всех, что является не столь безопасным, как нам бы того хотелось. К счастью, Linux поддерживает такую штуку под названием umask, используемую при создании нового файла. Система использует значением umask для понижения изначально указанных привилегий до уровня более безопасного и подходящего нам. Вы можете просмотреть ваше текущее значение umask введя следующую команду:

$ umask 0022

Нормальное значение umask на системах с установленной ОС Linux равно 0022, которое позволяет читать вам свежесозданные файлы (по возможности, конечно), но не модифицировать их. Дабы сделать новые файлы более защищёнными по умолчанию, можно сменить значение umask:

$ umask 0077

Данное значение umask гарантированно обеспечит вам то, что у группы и других пользователей не будет никакой возможности доступа к новым файлам. Так как же работает umask? Работая отличным от задания "обычных" прав доступа к файлам, umask указывает какие разрешения должны быть отключены. Давайте изучим табличку где описывается соответствие режима разряду, чтобы было понять значение числа 0077:

Режим Разряд rwx 7 rw- 6 r-x 5 r-- 4 -wx 3 -w- 2 --x 1 --- 0

Глядя на эту таблицу становится ясно, что 0077 обозначает ---rwxrwx. Только помните, что umask - это команда которая говорит системе какие права доступа *надо убрать*. Сложив два последних триплета, мы заметим, что разрешения для группы "другие" и членов той же группы были убраны, но права доступа для нашего пользователя остались нетронутыми.

Знакомимся с suid и sgid

Когда вы входите в систему, создаётся новый процесс командной оболочки. Как вы может уже знаете, а может и нет, но новый процесс командной оболочки (обычно, bash) запускается с вашим пользовательским идентификатором. Таким образом, bash может заполучить доступ ко всем вашим файлам и каталогам. Фактически, как пользователи мы полностью зависим от других программ, чтобы провести операции которые нам необходимы. Так как программы которые вы запускаете наследуют ваш идентификатор пользователя, то они не смогут получить доступ к любым объектам файловой системы для которых у них нет прав доступа.

Например, файл passwd не может быть изменён обычным, непривилигированным пользователем, поскольку флаг "запись" для этого файла сброшен для всех, кроме суперпользователя root:

$ ls -l /etc/passwd -rw-r--r-- 1 root wheel 1355 Nov 1 21:16 /etc/passwd

Однако обычные пользователи иногда нуждаются в необходимости модификации файла /etc/passwd (хотя бы не напрямую) для того, чтобы сменить собственный пароль. Но ведь если пользователь пользователь не может этого сделать, то как же это сработает?

suid

Благодаря модели разрешений в Linux существует два специальных бита которые называются sgid и suid. Когда на исполняемый файл устанавливается suid-бит, то он запускается с правами владельца файла, которые могут отличаться от прав доступа этот файл запускающего.

Так вот вернёся теперь к проблеме с файлом /etc/passwd. Если вы взглянете на исполняемый файл passwd, то обратите внимание, что владельцем файла является root:

$ ls -l /usr/bin/passwd -rwsr-xr-x 1 root wheel 17588 Sep 24 00:53 /usr/bin/passwd

Вы также заметите, что в том месте триплета где прописывается символ "x" стоит символ "s". Это указывает нам на то, что для конкретной программы выставлены биты исполняемости и suid. Поэтому когда вы запускаете команду passwd он будет выполняться от лица суперпользователя root (у которого имеются все полномочия), что отличается от прав пользователя который запустил её. Ну а поскольку мы запустили команду passwd от пользователя root, то мы можем модифицировать файл /etc/passwd без каких-либо проблем.

Необходимые пояснения по поводу suid/sgid

Как мы видим, suid и sgid работают одинаковым образом. Это позволяет программам наследовать права доступа отлчиные от прав запускающего их пользователя.

Важно знать!

Следует знать некоторую очень необходимую информацию касающуюся suid и sgid. Во-первых, sud и sgid находятя в том же месте, что и бит "x" видный в листинге команды ls -l. Если бит "x" установлен одновременно с вышеуказанными, то старший бит будет указан как s (в нижнем регистре). Если же бит выполнения не установлен, то он будет отображаться как S (в верхнем регистре).

Важно знать!

Второе важное примечание: suid и sgid может пригодится в очень многих случах, но неправильное использование указанных битов может привести к тому, что безопасность системы будет сильно снижена. Чем меньше у вас будет suidных программ, тем лучше. Хотя команда passwd одна из немногих, где выставленный бит suid жизненно необходим.

Изменение suid и sgid

Установка и удаление битов sgid и suid весьма несложные. Вот так, например, мы выставляем suid-бит:

chmod u+s /usr/bin/myapp

А вот так мы снимаем sgid с целого каталога. Мы также увидим, что данные разрешения будут применены в пару приёмов:

chmod g-s /home/drobbins

Разрешения и директории Итак, мы уже рассмотрели установку разрешений на примере обычных файлов. когда дело доходит до директориЙ, то ситуация немного отличается. Сами флаги для каталогов одинаковые, но вот интерпретируются они несколько по-другому.

Например, если для каталога выставлен флаг "чтение", то вы можете просмотреть содержимое каталога. Флаг "запись" означает, что вы можете создавать файлы в каталоге и "выполнение" означает, что вы можете войти в директорию и получить доступ к директориям находямися внутри неё. Без флага "выполнение" объекты файловой системы внутри каталога будут недоступны. Без флага "чтение" вы не сможете войти в каталог, но вы можете получить доступ к его содержимому указав точный абсолютный путь до него.

Директории и sgid

Если каталог создан с флагом sgid, то любые объекты файловой системы внутри него будут наследовать права группы для этого каталога. Эта особенность бывает необходима, например, длятого, чтобы дать права доступа к дереву каталогов для всех пользователей находящихся в одной группе:

mkdir /home/groupspace # chgrp mygroup /home/groupspace # chmod g+s /home/groupspace

Теперь, все пользователи входящие в группу mygroup могут создавать файлы и каталоги внутри /home/groupspace, и им будут автоматически назначаться права для группы mygroup. В зависимости от настроек umask все новые объекты файловой системы могут быть читабельны, или напротив, не читабельны, модифицируемы, или исполняемы другими членами группы mygroup.

Каталоги и их удаление Изначально настройки каталогов в Linux не являются пригодными для любой ситуации. Обычно, любой пользователь может переименовать или удалить файл внутри каталога, сколько угодно времени имея необходимые права. Для каталогов, которые принадлежат конкретным пользователям, такая ситуация вполне приемлема.

Однако, для тех каталогов, которые используются множеством пользователей вроде /tmp и /var/tmp, всё будет очень печально. Так все все могут записывать в эти каталоги, все могут удалять или переименовывать там какие угодно файлы - даже не являясь при этом владельцами этих файлов! В общем, очевидно, что использовать /tmp в качестве для чего-то ценного весьма проблематично, так как любой может выполнить команду rm -rf /tmp/* и уничтожить все ваши файлы в любой момент.

Но в Linux есть нечто, что называется "липкий бит" (sticky bit). Например, если установить "липкий бит" на каталог /tmp (с помощью команды chmod +t), то удалять или переименовывать файлы могут только владельцы файлов или каталога /tmp (обычно, им является root) или суперпользователь root. Как правило, во всех дистрибутивах Linux этот бит для каталога /tmp уже установлен, но вы можете найти данную возможность весьма полезной и в других ситуациях.

Незаметный первый разряд

И в завершение данного раздела мы, наконец-то, рассмтрим незаметный первый разряд цифрового режима. Как видите, он используется для "липкого" бита, а также для sgid и suid:

suid sgid sticky mode digit on on on 7 on on off 6 on off on 5 on off off 4 off on on 3 off on off 2 off off on 1 off off off 0

Приведу пример использования 4-х разрядного цифрового режима для выставления разрешений каталога, котрый будет использоваться рабочей группой:

chmod 1775 /home/groupfiles

В качестве домашнего домашнего задания рекомендую выяснить, что же означает данное число в правах доступа.

Управление учётными записями в Linux

Введение в файл /etc/passwd

В этом разделе мы изучим механизм управления учётными записями в Linux, начиная с файла /etc/passwd, который описывает нам, какие пользователи существуют в системе. Содержимое своего файла /etc/passwd вы можете узнать введя в консоли команду less /etc/passwd.

Каждая строка в файле /etc/passwd соответствует учётной записи в системе. В качестве примера приведу свой /etc/passwd:

drobbins:x:1000:1000:Daniel Robbins:/home/drobbins:/bin/bash

Как видите, информации содержащейся в одной строке более чем достаточно. Кстати, каждая строка в файле /etc/passwd содержит несколько полей, разделённых символом двоеточия.

Первое поле описывает имя пользователя, а второе поле содержит символ "x". На старых системах с ОС Linux это поле содержало зашифрованный пароль необходимый для аутентификации, но в реальности, все современные Linux-системы держат информацию об этом пароле в другом файле.

Третье поле (1000) содержит числовой идентификатор пользователя ассоциируемый с конкретным пользователем, четвёртое поле (1000) соотносится с конктерной группой, куда входит пользователь. В следующих разделах мы рассмотрим, как задаётся группа.

Пятое поле содержит текстовое описание данной учётной записи. В нашем случае это имя пользователя. Шестое поле описывает расположение домашней директории пользователя, а седьмое - его командную оболочку по умолчанию, которая запускается сразу же после его входа в систему.

Советы и фокусы связанные с файлом /etc/passwd

Как вы наверное заметили, в файле /etc/passwd перечислено значительно больше пользователей, чем тех, кто реально использует систему. Так происходит потому, что различные компоненты Linux используют учётные записи для улучшения безопасности. Обычно, в таких случаях, учётные записи системных пользователей имеют идентификационный номер меньше ста и большинство из них имеют оболочку для входа указанную как /bin/false. Программа /bin/false ничего не делает, кроме выдавания кода ошибки, что хорошо защищает попыток входа в систему используя системные учётные записи. Всё это необходимо только для внутреннего использования операционной системой.

Файл /etc/shadow

Итак, пользователи перечисляются в файле /etc/passwd. Системы с установленной там Linux имеют ещё один файл, который прилагается к /etc/passwd, называемый /etc/shadow. Этот файл, в отличие от /etc/passwd может просматривать только пользователь root и содержит зашифрованную информацию о паролях пользователей системы. Давайте поглядим примерное содержимое файла /etc/shadow:

drobbins:$1$1234567890123456789012345678901:11664:0:-1:-1:-1:-1:0

Каждая строка содержит информацию о конкретной учётной записи, а поля так же как и в прошлом примере разделяются символом двоеточия. Первое поле в строке указывает на учётную запись, которая связывается с так называемой теневой записью. Второе поле содержит зашифрованный пароль. Оставшиеся поля расшифровываются следующей таблицей:

поле 3 # количество дней отсчитывая от 1/1/1970, когда пароль был изменён

поле 4 # количество дней до того, после которого пароль можно будет изменить (0 разрешает изменение пароля в любое время)

поле 5 # количество дней, перед принудительным запросом смены пароля (-1 означает "никогда")

поле 6 # количество дней до истечения пароля когда пользователь будет предупреждён об этом (-1 означает "не предупреждать")

поле 7 # количество дней после истечения срока действия пароля, когда учётная запись будет автоматом заблокирована (-1 - "никогда не блокировать")

поле 8 # количество дней во время блокировки (-1 значит "учётная запись включена")

поле 9 #Зарезервировано на будущее

Файл /etc/group

Следующее на что мы посмотрим, это файл /etc/group, который определяет все группы в Linux. В качестве примера рассмотрим такую строку:

drobbins:x:1000:

Поля в файле /etc/group имеют уже рассмотренный нами формат. Первое поля указывает на название группы, второе поле содержит несуществующий пароль, заменённое символом "икс". Третье поле, как и в примерах выше указывает на идентификатор группы к которой он принадлежит. Четвёртое поле (в нашем примере оно пустое) содержит список пользователей, которые входят в эту группу.

Как вы помните, в примере с файлом /etc/passwd была строка указывающая на идентификатор группы равный 1000. Это является эффектом того, что пользователь drobbins находится в группе drobbins, несмотря на то, что имени drobbins нету в четвёртом поле файла /etc/groups.

Примечания к группам

Сделаю кое-какие примечания связанных с привязкой пользователей к группам: на некоторых системах можно обратить внимание на то, что каждая новая учётная запись пользователя связана с одинаково названным (обычно, ещё и одинаково пронумерованным) группам. На других системах все новые учётные записи помещаются в отдельную выделенную группу для пользователей. Подход используемый для администрирования системы зависит только от вас. создание отдельной группы для каждого пользователя на практике упрощает контроль за файлами путём помещения тех людей, которым вы доверяете в свои персональные группы.

Добавление пользователей и групп вручную

А теперь настало время показать, как можно создавать собственные учётные записи и группы. Самый лучший способ научиться этому, это добавить пользователя вручную. Перед там как начать, необходимо удостовериться, что ваша переменная EDITOR указывает на ваш любимый текстовый редактор:

echo $EDITOR vim

Если что-то не так, то переменную EDITOR можно изменить примерно таким образом:

export EDITOR=/usr/bin/emacs # vipw

А теперь откройте файл /etc/passwd в вашем любимом текстовом редакторе и убедитесь, что он появился у вас на экране. При модификации системных файлов passwd и group крайне важно использовать команды vipw и vigr. Это обеспечит дополнительные меры предосторожности, надёжно заблокировав файлы passwd и group так, чтобы их никто не повредил.

Редактирование файла /etc/passwd

Теперь, поскольку вы уже открыли файл /etc/passwd, пора добавить в него следующую строку:

testuser:x:3000:3000:LPI tutorial test user:/home/testuser:/bin/false

В этом примере мы только что добавили пользователя с именем "testuser" с UID (идентификатором пользователя) 3000. Мы также добавили его в группу имеющую GID (идентификатор группы) равным 3000, которую мы никогда не создавали до этого. Как вариант, мы может назначить GID от другой группы, если мы пожелаем. К новому пользователю добавлен комментарий который переводится как "Учебный тестовый пользователь для LPI". Домашняя директория для этого пользователя находится в /home/testuser, а командная оболочка указана как /bin/false в целях безопасности. Если вам надо создать не проблную учётную запись, то достаточно заменить это место на /bin/bash. Отлично, теперь сохраняемся и выходим.

Редактируем /etc/shadow

Давайте теперь добавим запись в файл /etc/shadow для нужного нам пользователя. Набираем для этого команду vipw -s . Должен открыться ваш любимый редактор, с содержимым файла /etc/shadow. Теперь давайте скопируем одну строку от уже существующей учётной записи пользователя (такие учётные записи имеют пароль и гораздо длиннее, чем стандартные системные учётные записи)

drobbins:$1$1234567890123456789012345678901:11664:0:-1:-1:-1:-1:0

Теперь заменим там имя пользователя и удостоверимся, что все остальные поля имеют те значения, которые нам нужны:

testuser:$1$1234567890123456789012345678901:11664:0:-1:-1:-1:-1:0

Теперь сохраняемся и выходим.

Задаём пароль

Возвращаемся в командную строку. Настало время задать пароль для свежесозданного пользователя:

passwd testuser Enter new UNIX password: (enter a password for testuser) Retype new UNIX password: (enter testuser's new password again)

Правим файл /etc/group

Теперь, когда мы отредактировали файлы /etc/passwd и /etc/shadow, настало время правильно настроить /etc/group. Чтобы выполнить это, наберите:

vigr

Перед вами должен появиться файл /etc/group готовый к правке. Если вы хотите назначить стандартную группу для нового пользователя, то добавлять новые группы в /etc/group не требуется. Однако, если же мы хотим создать новую группу для этого пользователя, то необходимо проделать следующее:

testuser:x:3000:

Сохраняемся. Выходим.

Создание домашнего каталога

Мы почти закончили. Введите следующие команды необходимые для создания домашенго каталога:

cd /home # mkdir testuser # chown testuser.testuser testuser # chmod o-rwx testuser

Теперь наша директория находится там где надо и можно приступить к работе. Ну, не совсем. Если вы планируете использовать эту учётную запись в своей работе, то вам потребуется команда vipw, чтобы изменить используемую по умолчанию командную оболочку на /bin/bash.

Полезные программы для управления учётными записями

Теперь, когда мы научились создавать учётные записи вручную, настало время познакомиться с некоторыми полезными и экономящими время программами доступными в Linux. Дабы сэкономить уйму места, мы не будем описывать действие этих программ в подробностях. Просто помните, что вы всегда сможете получить необходимую информацию по этим программам воспользовавшись справкой. Если планируете сдавать экзамен LPIC 101, то я крайне рекомендую потратить время на более близкое знакомство с каждой из этих команд.

Команда newgrp

По умолчанию, любые файлы создаваемые пользователем назначаются в ту группу, которая которая указана в файле /etc/passwd и куда входит нужный нам пользователь. Если пользователь состоит в других группах, то он или она могут ввести команду newgrp thisgroup, чтобы сменить текущее членство в группе по умолчанию. В дальнейшем, все новые файлы будут наследовать атрибуты другой указанной группы.

chage Команда chage необходима для просмотра и изменений возраста паролей сохранённых в файле /etc/shadow.

gpasswd Основная программа для управления группами

groupadd/groupdel/groupmod Используется для добавления/удаления/редактирования групп в файле /etc/group

useradd/userdel/usermod Используется для добвления/удаления/редактирования пользователей в файле /etc/passwd. Эти программы имеют и другие возможности, о которых можно узнать из справки.

pwconv/grpconv Используется для конвертирования файлов passwd и group в теневые пароли "нового формата". Все новые Linux-системы по умолчанию имеют такие пароли, так что вам не надо пользоваться ими.

"Тонкая" настройка пользовательского окружения

Знакомимся с программой fortune

Ваша командная оболочка имеет множество полезных возможностей, которые можно использовать для создания персональных настроек. Однако, мы ещё ни разу не затрагивали способов задания этих настроек автоматическим способом, при каждом входе в систему, кроме повторного набора команды. Данный раздел расскажет вам о том, как настраивать пользовательское окружение с помощью файлов автозагрузки.

Давайте для примера добавим приветственное сообщение появляющееся при каждом первом входе в систему. Дабы посмотреть на пример такого сообщения, достаточно набрать команду

fortune.$ fortune No amount of careful planning will ever replace dumb luck.

Файл .bash_profile

Теперь настроим программу forune на автозапуск при каждом входе в систему. Используйте ваш текстовый редактор для редактирования файла .bash_profile в вашем домашнем каталоге. Если такого файла нет, то его надо создать. После чего в файле пишите туда следующее:

fortune

Далее, попробуйте вйти из сеанса и зайти снова. Если только вы не используете менеджеры для графического входа в систему, вроде xdm, gdm, или kdm, то увидите вот такую оптимистичную надпись:

mycroft.flatmonk.org login: chouser Password: Freedom from incrustations of grime is contiguous to rectitude. $

Оболочка для входа в систему

Когда запускается bash, то он идёт к файлу .bash_profile расположенному в вашем домашнем каталоге, и запускает оттуда каждую указанную там строчку. Это называется поиск по файлу.

Bash действует в зависимости от того, каким способом он был запущен. Если он запускается как штатная оболочка при входе в систему, то он будет отрабатывать как в примере выше - сначала прочитает общесистемный /etc/profile, а затем ~/.bash_profile в домашнем каталоге.

Существует два способа указать командному интерпретатору bash, чтобы он запускался как оболочка испорльзуемая при входе в систему. Первый способ используется при первом входе в систему. Когда вы входите в свою учётную запись, то bash запускается с именем процесса "-bash", Это можно заметив взглянув на листинг процессов в системе:

$ ps u USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND chouser 404 0.0 0.0 2508 156 tty2 S 2001 0:00 -bash

Вероятнее всего, ваш листинг будет длиннее, но как минимум, вы увидите хоть один процесс в поле COMMAND с чертой перед именем процесса вашей оболочки вроде "-bash" как в примере выше. Данная черта используется для того, чтобы определить, что командный интерпретатор запущен в качестве оболочки для входа в систему.

Изучаем опцию --login

Второй способ указать командному интерпретатору, что нужно запуститься в режиме оболочки - это использовать опцию --login. Этот способ используется некоторыми эмуляторами терминала, например, xterm, чтобы каждая сессия bash работала как отдельная попытка входа в систему.

После входа в систему запускается ещё несколько копий оболочки. Несмотря на то, что эти сессии запускаются с параметром --login, и имеют черту перед своим именем, они не являются оболочками для входа. Если они выдают вам приглашение командной строки, то такие оболочки называются интерактивными. В интерактивном режиме bash полностью игнорирует файлы /etc/profile и ~/.bash_profile, но зато просматривает файл ~/.bashrc.

interactive login profile rc yes yes source ignore yes no ignore source no yes source ignore no no ignore ignore

Проверка интерактивности

В некоторых случаях, bash просматривает содержимое ~/.bashrc даже если он не находится в интерактивном режиме. Например, во время работы команд rsh и scp. Важно помнить об этом, так как он может вывести текст, вроде того, что мы использовали в примере с fortune, что в свою очередь, может помешать работе неинтерактивной сессии bash. Поэтому хорошим тоном было бы использование переменной PS1, чтобы определять, работает ли запущенная сессия в интерактивном режиме или нет, перед тем как вывести текст на экран.

if [-n "$PS1"]; then fortune fi

Файлы /etc/profile и /etc/skel

Как системный администратор, вы отвечаете за файл /etc/profile. Поскольку данный файл просматривается всеми пользователями, то крайне необходимо, чтобы данный файл был корректным. Это, кроме всего прочего, ещё и серьёзный инструмент позволяющий обеспечить правильную работу профилей новых пользователей, когда они совершают свой первый вход в систему.

Существуют также некоторые настройки обеспечивающие установку определённых умолчаний для всех новых пользователей, но при этом также легко меняющие их. Они находятся в каталоге /etc/skel. Когда вы выполняете команду useradd, то все файлы копируются в домашний каталог нового пользователя. Что означает возможность вставки чего-нибудь полезного в файлы .bash_profile и .bashrc находящихся в каталоге /etc/skel для начала.

Команда export

Переменные в командном интерпретаторе bash могут быть установлены единым образом для всех новых оболочек в момент запуска. Это называется экспортирование. Вы можете получить список всех экспортированных переменных для вашей сессии bash.

$ export declare -x EDITOR="vim" declare -x HOME="/home/chouser" declare -x MAIL="/var/spool/mail/chouser" declare -x PAGER="/usr/bin/less" declare -x PATH="/bin:/usr/bin:/usr/local/bin:/home/chouser/bin" declare -x PWD="/home/chouser" declare -x TERM="xterm" declare -x USER="chouser"

Экспортируем переменные

Если вы не установите переменную для экспорта, то все запущенные оболочки не получат заданный набор переменных. Хотя вы можете указать нужную переменную на экспорт путём передачи команде export следующей конструкции:

$ FOO=foo $ BAR=bar $ export BAR $ echo $FOO $BAR foo bar $ bash $ echo $FOO $BAR bar

В этом примере были заданы две переменные с именами FOO и BAR. Но только BAR была помечена на экспорт. Когда запустилась новая сессия bash,то он значение выставленное для переменной FOO было утеряно. Если же вы выйдете из этой новой сессии, то увидите изначальные значения переменных для FOO и BAR:

$ exit $ echo $FOO $BAR foo bar

Команды export и set -x

Как и в предыдущем случае, переменные могут быть заданы в файлах ~/.bash_profile или /etc/profile, помечены на экспорт и более никогда не трогаться. Но дело в том, что некоторые опции не могут быть экспортированы, поэтому необходимо в нужной последовательности вложить их сначала в ~/.bashrc а затем в ваш профиль.Подобную настройку можно произвести с помощью встроенной команды set:

$ set -x

Опция -x говорит bash, что требуется печатать каждую команду перед её запуском.

$ echo $FOO $ echo foo foo

Это позволяет также объяснить непредвиденное появление кавычек, или другие странности. Чтобы отключить опцию -x необходимо дать команду set +x. Для более подробной информации рекомендуем просмотреть справочную странцу bash по параметрам встроенной команды set.

Настройка переменных с помощью команды set

Встроенная команда set тоже может устанавливать значения переменных, правда, это возможность для неё побочная. Команда bash set FOO=foo означает то же самое, что FOO=foo. Чтобы снять значение переменной, достаточно воспользоваться командой unset:

$ FOO=bar $ echo $FOO bar $ unset FOO $ echo $FOO

Использование Unset вместо FOO=

Вообще-то это два совершенно разных метода сброса переменных, но тем не менее, местами чертовски трудно объяснить различие между ними. Единственный способ объяснить это - попробовать использовать команду set без параметров, чтобы просмотреть список всех доступных переменных:

$ FOO=bar $ set | grep ^FOO FOO=bar $ FOO= $ set | grep ^FOO FOO= $ unset FOO $ set | grep ^FOO

Использование команды set без указания паметров весьма похоже на действие команды export за тем исключением,что set показывает все перменные, а не только те, которые были экспортированы.

Экспортирование меняющее поведение команд

Крайне часто поведение команд можно изменить заданием переменных окружения. Такие программы могут увидеть экспортированные переменные, как только будут запущены новые сессии командного интепретатора bash. Например, команда man проверяет значение переменной PAGER, чтобы знать какую программу использовать для показа текстовой страницы:

$ PAGER=less $ export PAGER $ man man

Если установить значение переменной PAGER как less, то можно листать страницы нажимая клавишу "пробел". Если же указать в переменнной PAGES команду cat, то текст будет прокручен целиком без остановки.

$ PAGER=cat $ man man

Использование команды env

Если вы вдруг забыли установить значение переменной PAGER обратно на less, то команда man будет показывать все тексты одним куском, без возможности прокрутки. Если вам нужно установить cat в качестве значения переменной всего на один раз, то нужно использовать команду env:

$ PAGER=less $ env PAGER=cat man man $ echo $PAGER less

В этом случае, команда man будет использовать получит значение переменной PAGER cat, но само значение переменной внутри сессии bash останется неизменным.

Заключение и ресурсы

Заключение

Наши поздравления касающиеся завершения третьей части данного руководства! Начиная с этого момента, вы теперь знаете как находить информацию в системе и сети Интернет, а также получили достаточно хорошее представление о модели разрешений используемой в операционной системе Linux, управлении учётными записями и настройке пользовательского окружения.

LPI 101 (Часть четвёртая)

Содержание

- 1 Перед тем как начать
- 2 Файловые системы, блочные устройства, разделы
- 3 Загрузка операционной системы
- 4 Уровни запуска
- 5 Квоты файловой системы
- 6 Журналирование системы
- 7 Итоги и ресурсы

Перед тем как начать

Добро пожаловать в "Углубленное администрирование", последнюю из четырёх частей данной работы, разработанной для подготовки к сдаче экзамена LPI 101 (второй выпуск). В даном руководстве (четвёртой его части) мы расширим ваши знания и навыки по касающиеся углубленного администрированя Linux-систем, благодаря рассмотрению таких тем, как файловые системы в Linux, процесс загрузки системы, уровни запуска, квоты файловой системы и системные журналы.

Данное руководство особенно необходимо тем, кто станет заниматься системным администрированием впервые, посему мы рассмотрим множество низкоуровневых вещей, которые системные администраторы должны знать. Если же вы совсем новичок в Linux, то лучше всего начать с первой части и пройтись по всей серии статей вплоть до текущего момента. Для большинства аудитории данный матеирал будет внове, но, впрочем, даже опытные пользователи могут посчитать данное руководство хорошим способом для того, чтобы освежить основы навыков системного админстрирования и подготовиться к сдаче следующего экзамена LPI.

В конце данной серии статей (их планируется восемь, рассматривающие экзамены LPI 101 и 102) у вас появятся необходимые знания для того, чтобы стать системным администратором Linux-систем, и быть готовым к получению сертификата первого уровня LPIC от Linux Professional Institute, если того пожелаете.

Логотип LPI - торговая марка Linux Professional Institute.

Файловые системы, блочные устройства, разделы

Введение в блочные устройства

В данном разделе мы хорошенько рассмотрим различные аспекты связанные с дисками в Linux, включая файловые системы, разделы и блочные устройства. После знакомства с плюсами и минусами дисков и файловых систем, мы пройдёмся по процессу создания создания разделов и файловых систем в Linux.

Чтобы начать, я познакомлю вас с понятием "блочные устройства". Самый известный тип блочного устройства в Linux, представляет, наверное, жёсткий IDE-диск:

/dev/hda

Если же у вас имеется SCSI-диск, то тогда он будет обозначаться так:

/dev/sda

Уровни абстракции

Блочные устройства в Linux представляют собой абстракцию к интерфейсу диска. Пользовательские программы взаимодействуют с вашим диском не беспокоясь о драйверах для IDE, SCSI или к чему-либо ещё. Программы просто адресуют место на дисковом массиве используя набор смежных блоков с произвольным доступом размером в 512 килобайт.

Разделы

В Linux мы создаём файловые системы используя специальную команду mkfs (а также mke2fs, mkreiserfs и прочие) определяющая конкретное блочное устройство с помощью аргумента в командной строке.

Вообще, в теории мы можем использовать целое блочное устройство (представляясь единым диском) вроде /dev/hda или /dev/sda для размещения одной-единственной файловой системы, хотя на практике это не делается. Вместо этого, целое блочное устройство разделяется на более мелкие, куда более управляемые куски, именуемые разделами. Разделы создаются с помощью инструмента под названием fdisk, который используется для создания и редактирования таблицы разделов сохранённой на каждом диске. Таблица разделов определяет, как будет разделён весь диск целиком.

Знакомимся с fdisk

Теперь взглянем на таблицу разделов используя запущенный fdisk, указав нужное блочное устройство определяющее диск целиком в виде аргумента.

Примечание:

Существуют альтернативные интерфейсы для отображения таблицы разделов, такие как cfdisk, parted и partimage. Мы бы рекомендовали вам избегать использования cfdisk (несмотря на то, что пишет об этом справочная страница), так как в некоторых случаях она некорректно определяет геометрию дисков.

fdisk /dev/hda # fdisk /dev/sda

Важно знать!

Вам не следует сохранять или производить какие-либо изменения в таблица разделов диска, если любой из имеющихся разделов содержит хоть какие-то важные данные. В большинстве случаев такая операция приводит к потере данных.

Fdisk в подробностях

После запуска fdisk вы будете встречены строкой наподобие этой:

Command (m for help):

Нажмите клавишу "p", чтобы просмотреть текущую конфигурацию ваших дисков:

Command (m for help): p Disk /dev/hda: 240 heads, 63 sectors, 2184 cylinders Units = cylinders of 15120 * 512 bytes Device Boot Start End Blocks Id System /dev/hda1 1 14 105808+ 83 Linux /dev/hda2 15 49 264600 82 Linux swap /dev/hda3 50 70 158760 83 Linux /dev/hda4 71 2184 15981840 5 Extended /dev/hda5 71 209 1050808+ 83 Linux /dev/hda6 210 348 1050808+ 83 Linux /dev/hda7 349 626 2101648+ 83 Linux /dev/hda8 627 904 2101648+ 83 Linux /dev/hda9 905 2184 9676768+ 83 Linux Command (m for help):

Указанный выше диск сконфигурирован для размещения семи файловых систем (каждая из которых обозначена как Linux) и двух разделов с файлами подкачки (обозначенных как "Linux swap").

Обзор блочных устройств и партиционирования

Просим обратить ваше внимание на название блочных устройств указанных в левой колонке, начинающихся на /dev/hda1 и заканчивающихся на /dev/hda9. На заре персональных компьютеров, программы для создания разделов позволяли создавать не более четырёх разделов (именуемых "основными"). Это было чересчур неудобно, поэтому существует обходной манёвр, называемый созданием расширенных разделов. Расширенные разделы весьма похожи по своему стройству на основные разделы, за тем исключением, что не имеют ограничения на четыре раздела. Кроме того, расширенные разделы могут содержать в себе любое количество логических разделов, эффективно обходя проблему с ограничением на четыре раздела.

Продолжая обзор партиционирования

Все разделы начинающиеся с hda5 и выше, являются логическими разделами. Цифры с 1 по 4 зарезервированыы для основных и расширенных разделов.

В нашем примере диски hda1 по hda3 являются основными разделами, а hda4 расширенным, который в свою очередь содержит логические разделы начиная с hda5 по hda9. Раздел под названием /dev/hda4 использовать для хранения данных напрямую нельзя. Это всего лишь контейнер, в который вложены разделы с hda5 по hda9.

Типы разделов

Кстати, как вы уже успели заметить, каждый раздел содержит идентификатор (обозначенный как Id), который называется "тип раздела". Когда вы создаёте новый раздел, то нужно обязательно быть уверенным в том, что тип раздела указан правильным образом. Номер 83 - это правильный тип раздела, указываемый для размещения файловых систем Linux, а 82 - верный номер типа для для обозначения разделов предназначенных для файлов подкачки. Вы можете установить тип раздела прямо в fdisk используя клавишу "t". Ядро Linux использует тип раздела, чтобы автоматически опознавать разделы с данными и файлами подкачки во время загрузки.

Использование fdisk для создания разделов

После знакомства со способами разделения дисков в Linux, мы можем теперь приступить к процессу создания разделов и файловых систем для новой установки Linux. В процессе этого, мы сконфигурируем диск для новых разделов и создадим на нём файловую систему.

Все эти шаги позволяет нам получить полностью чистый диск без данных, в дальнейшем использованный для установки новой операционной системы.

Важно знать!

Перед тем как проделать все необходимые действия, вам нужно позаботиться о наличии жёсткого диска на котором нет важных данных, так как все производимые действия сотрут все данные на вашем диске. Если вы незнакомы со всем этим, то вы можете просто почитать об этом, либо использовать загрузочный диск с Linux на тестовой системе без риска потерять данные.

Как должен выглядеть разбитый диск

После проведения процесса разбиения диска, ваша таблица разделов должна смотреться примерно так:

Disk /dev/hda: 30.0 GB, 30005821440 bytes 240 heads, 63 sectors/track, 3876 cylinders Units = cylinders of 15120 * 512 = 7741440 bytes Device Boot Start End Blocks Id System /dev/hda1 * 1 14 105808+ 83 Linux /dev/hda2 15 81 506520 82 Linux swap /dev/hda3 82 3876 28690200 83 Linux Command (m for help):

Комментарии к примеру разбиения

В предложенной нами "конфигурации для новичков" имеется три раздела. Самый первый (/dev/hda1) начинающийся с самого начала диска и имеющего малый размер, называется загрузочным разделом. Загрузочный раздел предназначен для сохранения всех критичных данных необходимой для загрузки - информация загрузчика GRUB (если, конечно, вы будете использовать GRUB) и ваше ядро(а) Linux. Загрузочный раздел даст нам безопасное место для хранения всего необходимого связанного с загрузкой Linux. При ежедневном использовании ваш загрузочный раздел должен быть размонтированным в целях безопасности. Если вы настраиваете систему с SCSI-диском, то скорее всего, загрузочный раздел будет находиться на /dev/sda1.

Раньше рекомендовалось размещать загрузочные разделы (содержащие необходимую информацию для работы загрузчика) в начале диска. Сейчас этого делать необязательно, но эта полезная традиция пошла с тех времён, когда использовался загрузчик LILO, который на тот момент ещё не умел загружать ядро с расширенных разделов находящихся за границей 1024 цилиндра.

Второй раздел (/dev/sda2) используется для размещения подкачки. Ядро использует подкачку как виртуальную память, когда не хватает размера оперативной памяти. Этот раздел, обычно, не очень большой. Обычно, он занимает около 512 мегабайт. При настройке системы на SCSI диске, этот раздел будет именоваться /dev/sda2.

Третий раздел (/dev/sda3) самый большой и занимает всю оставшуюся часть диска. Данный раздел будет называться корневым разделом и будет использоваться для хранения файловой системы и всех данных на ней. На SCSI дисках, как вы уже догадались, он будет обозначен как /dev/sda3.

Приступаем к выполнению

Ну вот и хорошо. Теперь можно приступить к созданию разделов согласно примеру, который мы рассмотрели выше. во-первых, необходимо войти в fdisk используя команду fdisk /dev/hda или fdisk /dev/sda в зависимости от типа используемого диска. Затем нажмите клавишу "p", дабы просмотреть текущую конфигурацию разделов. На диске есть что-нибудь, что следовало бы сохранить? Если да, то лучше остановиться. Если вы продолжите действовать, то все данные будут потеряны.

Важно!

Следование инструкциям ниже может привести к тому, что все имеющиеся данные на диске будут стёрты! Если у вас есть на диске хоть что-то, то удостоверьтесь, что это не критичные данные, которые лучше не терять. Также убедитесь, что вы выбрали правильный диск, иначе можете по ошибке удалить данные не с того диска.

Проверка уже существующих разделов

Настало время удалить уже существующие разделы с данными. чтобы сделать это, нажмите клавишу "d" и клавишу "ввод". Будет запрошен номер раздела для удаления. Дабы удалить уже имеющийся раздел /dev/hda1, наберите следующее:

Command (m for help): d Partition number (1-4): 1

Раздел подготовлен для процесса удаления. Теперь, если вы снова нажмёте клавишу "p", то вы не увидите его, но удаление не произойдёт пока вы не сохраните изменения. Если вы допустили ошибку или не хотите ничего удалять, то нажмите клавишу "q" и ничего потеряно не будет.

Теперь удостоверимся, что мы удалили все имеющиеся разделы с нашей системы. Для чего надо повторно нажать клавишу "p" для получения списка таблицы разделоа, а затем нажать клавишу "d", чтобы удалить то, что осталось. В конце-концов, должно получиться следующая картина:

Disk /dev/hda: 30.0 GB, 30005821440 bytes 240 heads, 63 sectors/track, 3876 cylinders Units = cylinders of 15120 * 512 = 7741440 bytes Device Boot Start End Blocks Id System Command (m for help):

Создание загрузочного раздела

Как мы помним, таблица разделов на диске девственно чиста, и теперь мы полностью готовы к созданию загрузочного раздела. Чтобы выполнить это, нажмите клавишу "n", а затем клавишу "p", чтобы указать fdisk, что мы хотим создать основной раздел. Затем нажмите на клавишу 1, чтобы создать первый основной раздел. когда он запросит номер первого цилиндра, то жмите клавишу "ввод". Когда он запросит номер последнего цилидра, то напечатайте следующее: ***+100M***, для создания раздела размером в 100 мегабайт. Вот вывод проделанных операций:

Command (m for help): n Command action e extended p primary partition (1-4) p Partition number (1-4): 1 First cylinder (1-3876, default 1): Using default value 1 Last cylinder or +size or +sizeM or +sizeK (1-3876, default 3876): +100M

Теперь, когда вы нажмёте клавишу "p", то должны увидеть вот такую распечатку:

Command (m for help): p Disk /dev/hda: 30.0 GB, 30005821440 bytes 240 heads, 63 sectors/track, 3876 cylinders Units = cylinders of 15120 * 512 = 7741440 bytes Device Boot Start End Blocks Id System /dev/hda1 1 14 105808+ 83 Linux

Создание раздела подкачки

Следующим шагом будет создание раздела для подкачки. Чтобы выполнить это, нажмите клавишу "n", а затем клавишу "p", чтобы указать fdisk, что мы хотим создать второй основной раздел, им будет /dev/hda2 в нашем случае. После запроса цилиндра нажмите клавишу "ввод". После запроса последнего цилиндра, наберите ***+512M***, чтобы создать раздел размером 512 мегабайт. После того, как вы закончите, нажмите клавишу "t" и введите номер 82, чтобы обозначить тип раздела как "Linux Swap". После завершения всех шагов, нажмите клавишу "p", чтобы получить таблицу аналогичную этой:

Command (m for help): p Disk /dev/hda: 30.0 GB, 30005821440 bytes 240 heads, 63 sectors/track, 3876 cylinders Units = cylinders of 15120 * 512 = 7741440 bytes Device Boot Start End Blocks Id System /dev/hda1 1 14 105808+ 83 Linux /dev/hda2 15 81 506520 82 Linux swap

Делаем раздел загрузочным

И, наконец, нам необходимо установить флаг "загрузочный" на наш загрузочный раздел и сохранить все сделанные изменения. Чтобы пометить /dev/hda1 как "загрузочный" раздел, требуется нажать клавишу "a" в меню, затем нажать клавишу "1" для указания номера раздела. Если после этого нажмёте клавишу "p", то увидите, что /dev/hda1 помечен символом "астериск", более известный вам как "*" в колонке Boot. чтобы сохранить все сделанные изменения, нажмите клавишу "w", а затем "ввод". Теперь ваши диски правильно сконфигугрированы для установки Linux.

Примечание: Если fdisk вас попросит об этом, то перезагрузитесь и проверьте, чтобы система правильно отображала новую конфигурацию разделов.

Расширенные и логические разделы

В примере выше мы создали единый корневой раздел, который будет в дальнейшем содержать файловую систему и все наши сохранённые данные. Это означает, что после установки Linux, она станет основной файловой системой и будет смонтирована в "/" с содержимым дерева каталогов и нашими файлами.

Хоть это наиболее распространённый вариант настройки системы, но существует и другой способ с которым вы, кстати, вполне можете быть знакомы. Этот способ использует несколько разделов для хранения файловых систем, и после чего связывается в единое дерево файловой системы. Например, вы можете разместить каталоги /home и /var на отдельных разделах.

Мы также можем сделать hda2 расширенным, а не основным разделом. Затем, мы можем создать логические разделы hda5, hda6, и hda7 (которые технически будут размещены внутри /dev/hda2) используемые для размещения каталогов /, /home, и /var в зависимости от вашего желания.

Узнать больше о подобных конфигурациях с различными файловыми системами изучив раздел со списком ресурсов на следующей странице.

Создание файловых систем

Поскольку мы уже создали разделы, настало время создать файловую систему для корневого и загрузочного разделов, чтобы их можно было смонтировать и сохранять туда данные. Мы также сконфигурируем раздел для подкачки для обслуживания хранилища под соответствующие файлы.

Linux поддерживает разнообразные файловые системы, каждая из которых имеет свои сильные и слабые стороны и обладает собственным набором характеристик отвечающих за их производительность. Мы рассмотрим в данной работе создание таких файловых систем, как ext2, ext3, XFS, JFS, и ReiserFS. Перед тем как создать файловую систему, произведём обзорный экскурс по файловым системам в Linux. Более подробно мы рассмотрим их в дальнейшем на страницах данного руководства.

Файловая система ext2

Ext2 - наиболее надёжная и испытанная файловая система, но в ней отсутствует поддержка журналирования метаданных, что означает весьма долгую по времени стандартную проверку во время запуска системы. Сейчас существует куда более широкий спектр файловых систем нового поколения, которые могут быть проверены на целостность очень быстро и они куда более предпочтительны для применения в обычной повседневной работе, в отличие от нежурналируемых аналогов. Журналируемые системы позволяют избежать длительной задержки при загрузке операционной системы, когда ваша файловая система находится в нерабочем состоянии.

Файловая система ext3

Ext3 - это журналируемая версия файловой системы ext2, обеспечивающая журналирование метаданных для быстрого восстановления в дополние к другим широким возможностям журналирования, таких как ведение упорядоченного журнала с данными для всего содержимого на диске. Ext3 очень хорошая и надёжная файловая система. Позволяет получить приемлемую производительность на разумных условиях. Так как она мало использует так называемые "деревья" внутри своей архитектуры, то она не очень хорошо масштабируется. В свою очередь это означает, что это не самый хороший выбор для больших файловых систем или ситуаций, когда требуется хранить большие файлы или множество больших файлов в одном каталоге. Но если не использовать её в ситуациях не предусмотренных архитектурой, то ext3 - отличная файловая система.

Одной их приятных особенностей ext3 является то, что все существующие системы с файловой системой ext2 могут быть легко обновлены до уровня ext3. Это позволяет обеспечить прозрачное обновление систем, где уже используется файловая система ext2.

Файловая система ReiserFS

ReiserFS - файловая система использующая B-деревья в своей архитектуре, что очень хорошо сказывается на скорости и надёжности в сравнении с ext2 и ext3, когда дело касается множества мелких файлов. Начиная с версии ядра 2.4.18 и выше, ReiserFS является крайне надёжной и крайне рекомендуемой системой как для случаев повседневного использования, так и для ситуаций, где требуется наличие файловых систем очень большого размера, где используется множество мелких файлов, файлов очень большого размера и каталогов, в которых имеются десятки тысяч файлов. Мы крайне рекомендуем использовать ReiserFS для не загрузочных разделов.

Файловая система XFS

XFS - это тоже файловая система с журналированием метаданных. Поставляется с солидным набором возможностей и оптимизирована для масштабируемости. Но мы рекомендуем её использовать только в случае наличия высокопроизводительных SCSI-дисков или хранилищ доступных по Fibre Channel и источников бесперебойного питания. Поскольку XFS крайне агрессивно кэширует передавемые данные в оперативной памяти, то неправильно спроектированные программы (относится к тем, кто не предпринимает мер во время записи на диск (хоть их и весьма мало) могут привести к потере изрядного куска данных во время внезапной остановки системы.

Файловая система JFS

JFS - высокопроизводительная журналируемая файловая система от IBM. Относительно недавно была подготовлена для промышленной эксплуатации, поэтому нет каких-либо данных относительно стабильности данной файловой системы.

Рекомендации по выбору файловых системам

Если вы ищете наиболее надёжную файловую систему, то используйте ext3. Если нужна высокопроизводительная файловая система для повседневных задач с поддержкой журналирования, то выбирайте ReiserFS. Впрочем, обе файловые системы являются зрелыми, совершенными и пригодными для типовых задач.

Основываясь на примерах выше, мы используем следующие команды, чтобы задействовать все файловые системы:

mke2fs -j /dev/hda1 # mkswap /dev/hda2 # mkreiserfs /dev/hda3

Мы выбрали ext3 для загрузочного раздела /dev/hda1, так как это надёжная файловая система поддерживаемая всеми основными загрузчиками. Использование mkswap для задействования раздела для подкачки на /dev/hda2 - и так довольно очевидно. Для нашей основной файловой системе располагающейся на /dev/hda3 мы выбрали ReiserFS, так ак это надёжная журналируемая файловая система предлагающая превосходную производительность. А теперь пойдём и инициализируем файловые ситемы.

Создание раздела подкачки

Команда mkswap инициализирует раздел подкачки:

mkswap /dev/hda2

В отличие от обычных файловых систем, разделы с подкачкой не монтируются. Вместо этого они включаются командой swapon:

swapon /dev/hdc6

Сценарии запуска вашей системы сами позаботятся о том, чтобы автоматически включить раздел подкачки. Таким образом, команда swapon обычно нужна только для того, чтобы сразу же включить недавно добавленный раздел с подкачкой в вашу систему. Просмотреть полный список включённых разделов подкачки можно с помощью команды cat /proc/swaps.

Создание файловых системам с ext2, ext3 и ReiserFS

Для создания файловой системы с ext2 надо ввести команду mke2fs:

mke2fs /dev/hda1

Если вам необходимо создать файловую систему ext3 вместе с журналом, то надо выполнить mke2fs -j:

mke2fs /dev/hda3

Файловая система ReiserFS создаётся командой mkreiserfs:

mkreiserfs /dev/hda3

Создание файловых систем XFS и JFS

Для создания XFS введите команду mkfs.xfs:

mkfs.xfs /dev/hda3

Примечание. Возможно, вы захотите добавить парочку дополнительных флагов при выполнении команды mkfs.xfs: -d agcount=3 -l size=32m. Команда -d agcount=3 уменьшает количество размещённых групп. XFS будет требовать, как минимум, одну группу на 4 гигабайта пространства, поэтому, например, для раздела размеров в 20Гб вам потребуется значение agcount не менее 5. Команда l size=32m увеличивает размер журнала до 32 мегабайт, что хорошо сказывается на производительности.

JFS создаётся командой mkfs.jfs:

mkfs.jfs /dev/hda3

Монтирование файловых систем

После того, как файловая система была создана её можно смонтировать используя команду mount:

mount /dev/hda3 /mnt

Для монтирования файловой системы необходимо указать раздел с блочным устройством и "точку монтирования" в качестве второго аргумента. Новая файловая система можно сказать "вырастет" в заданной точке монтирования. Существует так же эффект "скрытия" файлов, которые в этот момент находились в корневом каталоге директории /mnt на родительской файловой системе. позднее, когда файловая система будет отмонтирована, файлы появятся снова. После выполнения команды монтирования и копирования всех новых файлы внутри каталога /mnt с файловой системой ReiserFS будут сохранены должным образом.

Давайте заодно смонтируем наш загрузочный раздел в /mnt. Проделывается следующим образом:

mkdir /mnt/boot # mount /dev/hda1 /mnt/boot

Теперь наш загрузочный раздел доступен в катлоге /mnt/boot. Если мы создадим файлы в этом каталоге, то они будут сохранены на разделе с ext3, который физически располагается на диске /dev/hda1. Если же мы создадим файл внутри каталога /mnt, но не /mnt/boot то они сохранятся на разделе с ReiserFS, который располагается на /dev/hda3. Файлы созданные вне каталога /mnt не будут сохранены ни на одном из указанных разделов, кроме раздела на основной системе или загрузочном диске.

Чтобы узнать какие файловые системы у нас смонтированы необходимо выполнить команду mount. Для примера посмотрим вывод команды на нашей текущей системе, в которой разделе сконфигурированы указанным выше образом:

/dev/root on / type reiserfs (rw,noatime) none on /dev type devfs (rw) proc on /proc type proc (rw) tmpfs on /dev/shm type tmpfs (rw) usbdevfs on /proc/bus/usb type usbdevfs (rw) /dev/hde1 on /boot type ext3 (rw,noatime)

Нечто подобное вы уже, наверное ,видели выполняя команду cat /proc/mounts. Корневая файловая система /dev/hda3 смонтирована ядром автоматически в процессе загрузки и ему дано символическое имя /dev/hda3. В нашей системе оба раздела /dev/hda и /dev/root находятся на одном и том же блочном устройстве.

ls -l /dev/root lr-xr-xr-x 1 root root 33 Mar 26 20:39 /dev/root -> ide/host0/bus0/target0/lun0/part3 # ls -l /dev/hda3 lr-xr-xr-x 1 root root 33 Mar 26 20:39 /dev/hde3 -> ide/host0/bus0/target0/lun0/part3

Чуть больше интересной информации про монтирование

Так что же это за файл такой /dev/ide/host0....? Системы вроде моей, использующие специальную файловую систему devfs для управления устройствами располагающимися в /dev, имеют гораздо более длинные официальные имена, чем более старые Linux-системы. К примеру, /dev/ide/host0/bus1/target0/lun0/part7 это официальное имя для /dev/hdc7, а /dev/hdc7 в свою очередь является символической ссылкой на блочное устройство. Вы и сами можете определить использование devfs вашей системой проверив наличие файла /dev/.devfsd. Если он на месте, то devfs активна.

При использовании команды монтирования, операционная система сама пытается определить тип файловой системы. но иногда это не срабатывает и приходится указывать определённый тип файловой системы вручную, используя опцию -t как здесь:

mount /dev/hda1 /mnt/boot -t ext3

или

mount /dev/hda3 /mnt -t reiserfs

Опции монтирования

Существует возможность для задания различных атрибутов для монтируемых файловых систем путём задания различных опций монтирования. Для примера можно смонтировать ФС в режиме "только для чтения" используя опцию ro:

mount /dev/hdc6 /mnt -o ro

Смонтировав /dev/hdc6 в режиме "только для чтения" вы не сможете изменить какие-либо файлы в /mnt - только смотреть. Если ваша файловая система находится в режиме "чтение/запись", то её необходимо "перемонтировать" и переключить в режим "только для чтения". можно использовать опцию "remount", чтобы не проводить операцию по монтирования/размонтированию два раза:

mount /mnt -o remount,ro

Обратите внимание на то, что мы не указывали блочное устройство для смонтированного раздела, поскольку оно уже было смонтировано и программа mount знала, что /mnt используется с /dev/hdc6. чтоюы снова иметь возможность записи, проделываем следующее:

mount /mnt -o remount,rw

Ещё помните о том, что команда перемонтирования не сработает, если хоть один процесс в этот момент обращается к любому файлу или каталогу в /mnt. Дабы познакомиться с опциями монтирования в Linux более подробно, просмотрите man mount.

Знакомимся с файлом fstab

Итак, мы рассмотрели разбиение на разделы и монтирование файловых систем вручную используя загрузочный диск. Но однажды установив Linux, как мы сможем настроить систему, чтобы она монтировала правильную файловую систему в нужное время?

Например, мы установим Gentoo Linux на указанную выше конфигурацию. Откуда система узнает, что корневой раздел надо искать на /dev/hda3? А остальные файловые системы вроде раздела подкачки, необходимые во время загрузки системы как узнают, которая из них потребуется?

Допустим, что ядро Linux скажет какая корневая файловая система используется загрузчиком, и мы ещё рассмотрим тему загрузчиков в данном руководстве. А для всего остального в Linux есть такой файл, называемый /etc/fstab, говорящий о том, какие файловые системы доступны для монтирования. Настало время взглянуть на него.

Пример содержимого fstab

Давайте взглянем на примерное содержимое /etc/fstab:

Каждая незакомментированная строка в примере выше - это указание на нужный раздел блочного устройства, точку монтирования, тип файловой системы и опции необходимые для монтирования, а также два числовых поля. Первое числовое поле необходимо для команды dump, необходимой для резервного копирования. Если вы не планируете использовать dump, то можно спокойно игнорировать это поле. Последнее поле используется программой для проверки целостности диска fsck, указывающее порядок проверки разделов при загрузке системы. Мы также затронем fsck через какое-то время.

Взгляните на строку, где указана /dev/hda1. Вы увидите, что /dev/hda1 это раздел с ext3, который монтируется в точку монтирования /boot. Давайте рассмотрим опции монтирования /boot в колонке opts. Опция *noauto* сообщает нам, что не требуется автоматическое монтирование во время процесса загрузки. Без данной опции /boot будет монтироваться автоматически во время запуска системы.

Обратите внимание на опцию *noatime*, которая отключает режим записи информации о последнем времени доступа к диску. Данная информация обычно не требуется, посему отключение этой опции положительно сказывается на производительности дисковой подсистемы.

Теперь внимательно изучите строку с разделом /proc и обратите внимание на опции используемые по умолчанию. Используёте умолчания если вы хотите смонтировать файловую систему в стандартной точке монтирования. Поскольку /etc/fstab имеет несколько полей, то нельзя просто так взять и оставить поле с опциями пустым.

А теперь прошу обратить внимание на поле с /dev/hda2. Данная строка определяет /dev/hda2 как раздел для подкачки. Так как этот раздел не монтируется как обычная файловая система, поэтому в точке монтирования указано значение *none*. Благодаря файлу fstab, раздел /dev/hda2 будет подключен автоматически во время загрузки.

В файле fstab содержится запись /dev/cdrom наподобие указанной, необходимая для более простого монтирования компакт-дисков. Вместо ввода вот такой конструкции:

mount -t iso9660 /dev/cdrom /mnt/cdrom -o ro

Вам нужно будет вводить только это:

mount /dev/cdrom

Таким образом, использование файла /etc/fstab позволяет нам по желанию использовать дополнительную опцию *user*. Данная опция монтирования говорит системе о том, что надо разрешить монтирование указанной файловой системы от лица любого пользователя. Это бывает полезно для устойств вроде приводов компакт-дисков. Без данной опции монтирования в файле fstab, только суперпользователь root имеет право подключать компакт-диски к системе.

Размонтирование файловых систем

В обычной ситуации, все файловые системы сами октлючаются когда система прекращает свою работу или перезагружается. Когда файловая система отмонтируется, то все записанные в кэш-память данные сбрасываются на диск.

Тем не менее, есть возможность отключить файловые системы вручную. Перед тем как файловая система будет отмонтирована, убедитесь, что никакие запущенные процессы не обращаются к файлам на диске. Затем, используйте команду unmount, указав имя устройства или точку монтирования в качестве аргумента:

umount /mnt

или так:

umount /dev/hda3

После отмонтирования, файлы в /mnt "спрятанные" на файловой системе смонтированной до этого, снова появятся.

Знакомимся с программой fsck.

Если ваша система дала сбой или заблокирована по каким-то причинам, то она не сможет правильно отмонтировать файловые системы имеющиеся в ней. Когда это случается, файловая система находится в нецелостном (непредсказуемом) состоянии. При перезагрузке, программа fsck обнаружит такие неверно отмонтированные файловые системы и захочет провести проверку целостности в соответствии с записями об файловых системах указанных в /etc/fstab.

Крайне важно!

Файловые системы проверяемые fsck должны иметь ненулевое значение поля "проход" (последний столбец). Обычно, для корневой файловой системы устанавливается значение прохода равным 1, говорящее о том, что данная файловая система будет проверена первой. Для всех остальных файловых систем, которые должны проверяться при загрузке должно быть установлено значение равное 2 и выше. Для некоторых журналируемых файловых систем вроде ReiserFS лучше указывать значение прохода равным 0, так как журнал (а не внешней утилиты fsck) позаботится о целостности файловой системы.

В некоторых случаях после перезагрузки будет обнаружено, что fsck не может отремонтировать полностью или частично повреждённую файловую систему. В этом случае, вам необходимо перевести систему в однопользовательский режим и запустить fsck вручную, указав необходимое блочное устройство в качестве аргумента. Программа fsck проведёт ремонт и может запросить об исправлении некоторых обнаруженных дефектов. Как правило, вам следует отвечать "y" (то есть yes) на все задаваемые fsck вопросы.

Проблемы использования fsck

Одной из проблем fsck является то, что сканирование разделов может занять весьма длительное время, необходимое для проверки записей метаданных файловой системы (внутренней структуры данных), для того, чтобы удостовериться в их целостности. Для очень больших файловых систем такая исчерпывающая проверка591 можеть занять у fsck более часа.

Для решения этой проблемы был спроектирован новый тип файловых систем, называемый журналируемой файловой системой. Журналируемые файловые системы пишут на диск специальный журнал обо всех последних сделанных изменениях в структуре метаданных. В случае сбоя драйвер файловой системы смотрит в первую очередь именно его. Поскольку журнал содержит точное количество последних сделанных изменений на диске, то только эти части нуждаются в проверке на ошибки. Благодаря этому отличию, проверка ждурналируемых файловых систем отнимает секунды вне зависимости от размера файловой системы. Именно по этой причине такие файловые системы весьма популярны в сообществе пользователей Linux. Для большей информации по ним рекомендуем почитать "Обзор файловых систем в Funtoo, часть первая. Журналирование и ReiserFS"

Ну и двайте пробежимся по основным используемым файловым системам в Linux вместе со связанными с ними командами и опциями.

Файловая система ext2

Файловая система ext2 долгие годы была стандартом в Linux. Она предлагает хорошую производительность для большинства приложений, но не имеет никаких возможностей для журналирования. Что делает её непригодной для очень больших по размеру файловых систем, так как проверка в fsck будет отнимать много времени. Вдовесок к этому, ext2 имеет ряд внутренних ограничений, вроде ограниченного количества одновременно удерживаемых инодов. Тем не менее, ext2 самая испытанная и надёжная из всех не журналируемых файловых систем.

- Ядро: версии 2.0 и выше
- Журналирование: Нет
- Команда mkfs: mke2fs
- Пример команды mkfs: mke2fs /dev/hdc7
- Связанные с ней команды: debugfs, tune2fs, chattr
- Опция связаная с производительностью: noatime

Файловая система ext3

Файловая система ext3 имеет тот же формат данных, что и ext2, но с добавлением возможностей журналирования. Все системы использующие ext3 имеют широкие возможности связанные с журналированием, включая поддержку не только журналирования метаданных, но и упорядоченное журналирование (режим по умолчанию), режим полного журналирования данных+метаданных. Эти особые режимы помогают быть уверенным в целостности данных, не только благодаря коротким проверкам файловых систем и другим возможностям журналов. Именно по этой причине ext3 является лучшей файловой системой в случае, когда обеспечение целостности данных - основная задача. Однако, эти возможности для целостности данных сильно влияют на производительность. К тому же ext3 использует тот же формат, что и ext2, поэтому испытывает те же проблемы с масштабируемостью, как и её не журналируемая сестра. Так что если вы ищете хорошую файловую систему для повседневных задач, то ext3 - лучший выбор, поскольку весьма надёжна.

- Ядро: версия 2.4.16 и выше
- Журналирование: метаданные, упорядоченная запись данных, полное журналирование данных+метаданных

- Команда mkfs: mke2fs -j
- Пример команды mkfs: mke2fs -j /dev/hdc7
- Связанные с ней команды: debugfs, tune2fs, chattr
- Опция связаная с производительностью: noatime
- Другие поции монтирования:
  - data=writeback (отключить журнал)
  - data=ordered (используется по умолчанию, журнал метаданных и обычных данных записываются на диск вместе с метаданными)
  - data=journal (полное журналирование всех данных для обеспечение целостности данных на диске и метаданных. Сильно снижает производительность операций записи.)
- Ресурсы по ext3:
  - Обзор файловых систем в Funtoo, часть 4. Введение в ext3.
  - Обзор файловых систем в Funtoo, часть 5. Ext3 в действии.

Файловая система ReiserFS

Относительно новая файловая система разработнная с целью обеспечения высокой производительности на операциях с маленькими файлами, имеющую хорошую общую производительность и высокую масштабируемость. В общих чертах, ReiserFS имеет хорошую скорость для большинства операций. Именно поэтому ReiserFS предпочитается весьма многими.

- Ядро: версия 2.4.0 и выше (крайне рекомендуем использовать версию 2.4.18 и выше)
- Журналирование: metadata
- Команда mkfs: mkreiserfs
- Пример команды mkfs: mkreiserfs /dev/hdc7

- Опции связанные с производительностью: noatime, notail
- Ресурсы о ReiserFS:
    - Обзор файловых систем в Funtoo, часть 1. Журналирование и ReiserFS.
    - Обзор файловых систем в Funtoo, часть 2. Использование ReiserFS в Linux.

Файловая система XFS

XFS - файловая система промышленного уровня спортированная в Linux компанией SGI. XFS - богатая возможностями, масштабируемая, журналируемая файловая система, являющаяся хорошим выбором для очень высоконадёжного оборудования (в связи с большим кэшированием данных в оперативной памяти) и плохой выбор для дешёвых систем.

- Ядро: версия 2.5.34 и выше. Для серий ядер 2.4 требуется патч.
- Журналирование: метаданные
- Команда mkfs: mkfs.xfs
- Пример команды mkfs: mkfs.xfs /dev/hdc7
- Опция связаная с производительностью: noatime
- Ресурсы про XFS:
    - Домашняя страница XFS

Файловая система JFS

JFS - высокопроизводительная файловая система спортированная в Linux компанией IBM. JFS используется для серверов корпоративного уровня и разработана для высокопроизводтельных приложений. Узнать больше о JFS можно на сайте проекта JFS.

- Ядро: версия 2.4.20 и выше
- Журналирование: метаданные
- Команда mkfs: mkfs.jfs

- Пример команды mkfs: mkfs.jfs /dev/hdc7
- Опция связаная с производительностью: noatime
- Ресурсы по JFS:
  - Веб-сайт проекта JFS на сайте компании IBM.

Файловая система VFAT

Файловая система VFAT - это, на самом деле не файловая система для хранения данных в Linux. По сути, это DOS-совместимый драйвер файловой системы предназначенный для монтирования и обмена данными с DOS и Windows-системами. Драйвер для него имеется в штатной поставке с ядра Linux.

Загрузка операционной системы

О данном разделе

Данный раздел познакомит вас с процедурой загрузки Linux. Мы также рассмотрим концепцию загрузчика, настройку ядра для загрузки, и как проверять журнал загрузок на ошибки.

MBR aka Master Boot Record

Процесс загрузки аналогичен для большинства машин, вне зависимости от установленного дистрибутива. Расмотрим следующий пример жесткого диска:

+----------------+ | MBR | +----------------+ | Раздел 1: | |Корень Linux (/)| | содержит | | ядро Linux | | и систему. | +----------------+ | Раздел 2: | | подкачка | +----------------+ | Раздел 3: | | Windows 3.0 | | (последний раз | |работала в 1992)| +----------------+

Во-первых, BIOS компьютера читает первые несколько секторов с вашего жёсткого диска. Данные секторы содержат очень маленькую программу именуемую главной загрузочной записью (Master Boot Record), или MBR как её ещё называют. MBR хранит месторасположение ядра Linux на жёстком диске (первый раздел в нашем примере), затем загружает ядро в память и запускает его.

Процесс загрузки ядра

Следующее, что вы увидите (хотя оно может промелькнуть весьма быстро) это строка похожая на нечто такое:

Linux version 2.4.16 (root@time.flatmonk.org) (gcc version 2.95.3 20010315 (release)) #1 Sat Jan 12 19:23:04 EST 2002

Данная строка выводится ядром в момент старта. Первая часть строки - номер версии ядра, затем идёт идентификатор пользователя собравшего ядро (обычно, это root), версия компилятора которым это всё было собрано, и отметка времени окончания сборки.

Затем последует уйма строк вывода ядра, которая покажет какое оборудование у вас в системе стоит: процессор, шина PCI, дисковый контроллер, диски, последовательные порты, флоппи-приводы, устройства использующие шину USB, сетевые адаптеры, звуковые карты, и многое другое сообщит вам о своём статусе.

/sbin/init

Когда ядро закончит загружаться, запустится программа под названием init. Эта программа работает до тех пор, пока система не выключится. Ей всегда назначается идентификатор процесса равный 1. Что, собственно, и видно:

$ ps --pid 1 PID TTY TIME CMD 1 ? 00:00:04 init.system

Программа init запускается в большинстве дистрибутивов серией специальных сценариев (скриптов). Обычно, эти скрипты живут в /etc/rc.d/init.d или /etc/init.d, и они обеспечивают запуск таких служб как имя системы, проверка файловых систем на ошибки, монтирование дополнительных файловых систем, включение сети, запуск служб печати и тому подобное. Когда все скрипты выполнятся, программа init вызовет другуб программу, именуемую getty, которая отобразит вам приглашение для входа в систему и вы можете приступать к работе!

Копнём поглубже: LILO

Теперь, поскольку мы прошлись по процессу загрузки Linux, настало время посмотреть поближе на первую часть - MBR и загрузку ядра. Обслуживанием MBR занимается такая вещь как "загрузчик". Два наиболее популярных загрузчика для систем с архитектурой x86 это LILO (LInux LOader) и GRUB (GRand Universal Bootloader).

Из этих двух, LILO наиболее старый и простой загрузчик. О существовании LILO говорится при загрузке, показом короткой строчки приглашения "LILO boot:". Учтите, что вам может потребоваться зажать клавишу левый shift во время загрузки, чтобы увидеть приглашение. Связано это с тем, что чаще всего система настроена таким образом, что пропускает это всё без остановки.

Приглашение LILO весьма простое и без украшательств, но если вы нажмёте клавишу Tab, то увидите список доступных вам ядер (или операционных систем) для загрузки. Чаще всего пункт в списке всего один. Вы можете загрузиться набрав его название и нажав клавишу "ввод". Или просто нажмите клавишу "ввод", тогда по умолчанию загрузится первый пункт меню.

Использование LILO

Может случиться такая ситуация, когда вы хаотите выполунить какую-нибудь опцию необходимую при загрузке системы. Наиболее частые опции: root= - указывающая месторасположение другого корневого раздела, init= - указывающая на расположение другой программы инициализации (init=/bin/sh, например, для восстановления неправильно настроенной системы), и mem= для указания количества памяти в системе (например, опция mem=512M необходима в случае, если Linux видит только 128Мб ОЗУ). Вы можете ввести все эти параметры в строке приглашения LILO:

LILO boot: linux root=/dev/hdb2 init=/bin/sh mem=512M

Если вам потребуется указать опции командной строки для постоянного использования, вы можете рассмотреть их добавление в файл /etc/lilo.conf. Описание формата файла имеется в man-странице lilo.conf(5).

Важное замечание по LILO

Перед тем как перейти к загрузчику GRUB, хотелось бы сделать важное отступление касающееся LILO. Перед тем, как сделать изменения в файле /etc/lilo.conf или установить новое ядро, вы обязательно должны запустить программу lilo. Эта программа перепишет MBR на основе сделанных изменений, вплоть до абсолютного пути к ядру системы. В примере ниже мы специально использовали опцию -v для наглядности:

lilo -v LILO version 21.4-4, Copyright (C) 1992-1998 Werner Almesberger 'lba32' extensions Copyright (C) 1999,2000 John Coffman Reading boot sector from /dev/hda Merging with /boot/boot.b Mapping message file /boot/message Boot image: /boot/vmlinuz-2.2.16-22 Added linux * /boot/boot.0300 exists - no backup copy made. Writing boot sector.

Копая глубже: GRUB-legacy

GRUB-legacy - это другой популярный загрузчик в Linux. GRUB-legacy поддерживает куда больше операционных систем и предоставляет больше возможностей, вроде парольной защиты загрузочного меню и более простого управления.

Обычно, GRUB-legacy устанавливается в с помощью команды grub-install или grub-legacyinstall. Уже установленное, меню GRUB-legacy легко управляется с помощью редактирования файла /boot/grub/grub.conf. Рассмотрение обеих задачи выходят далеко за рамки этого документа, поэтому вам необходимо прочитать info-странцу GRUB-legacy перед тем, как попробовать установить или управлять им.

Использование GRUB-legacy

Для передачи паметров ядру, достаточно нажать клавишу "e" на клавиатуре в меню загрузки. Это даст вам возможность редактирования (очередным нажатием клавиши "e") имени ядраи параметров загрузки.После окончания редактирования нажмите "ввод", а затем "b" чтобы продолжить загрузку с вашими изменениями.

Смотрим dmesg

Сообщения возникающие при загрузке Linux и работе скриптов инициализации прокручивается очень быстро. Вы можете заметить сообщение об ошибке, но вряд ли успеете подробно разобрать его. В таком случае нам потребуется заглянуть в пару мест после загрузки системы и выяснить, что не так (и, надеюсь, решить как это исправить).

Если ошибка возникает во время загрузки ядра и опроса устройств, вам необходимо запросить копию загрузки журнала используя команду dmesg:

#dmesg Linux version 2.4.16 (root@time.flatmonk.org) (gcc version 2.95.3 20010315 (release)) #1 Sat Jan 12 19:23:04 EST 2002

Э-э-эй! А мы ведь знаем, что это за строчка. Это ведь самая первая строка возникающая при загрузке Linux. Более того, если вы поместите вывод dmesg в просмотрщик, то сможете построчно посмотреть то, что ядро выводит на экран во время загрузки, плюс все сообщения ядра выводимые на тот момент в консоль.

/var/log/messages

Второе место куда необходимо идти за информацией, это файл /var/log/messages. Этот файл пишется системным демоном syslog, который принимает ввод из библиотек, демонов и ядра. Каждое сообщение в этом файле имеет пометку о времени. Этот файл - хорошее место для начала разбора ошибок произошедших во время работы скриптов инициализации при загрузке. Вот, например, вывод нескольких сообщений с сервера имён:

grep named /var/log/messages | tail -3 Jan 12 20:17:41 time /usr/sbin/named[350]: listening on IPv4 interface lo, 127.0.0.1#53 Jan 12 20:17:41 time /usr/sbin/named[350]: listening on IPv4 interface eth0, 10.0.0.1#53 Jan 12 20:17:41 time /usr/sbin/named[350]: running

Дополнительная информация

Дополнительная информация относящаяся к данному разделу может быть найдена здесь:

- Учебник: "Как познать GRUB"
- LILO Mini-HOWTO
- Домашняя страница проекта GRUB
- Опции командной строки ядра в файле /usr/src/linux/Documentation/kernel-parameters.txt.

Уровни запуска

Однопользовательский режим

Возвращаясь к содержимому раздела рассказывающеи о загрузчиках, мы вспоминаем о возможности передачи параметров ядру. Одним из таких наиболее часто используемых параметров является параметр "s", приводящий к тому, что система загружается однопользовательском режиме. Этот режим позволяет смонтировать только корневую файловую систему с минимальным набором сценариев инициализации, а также запустить оболочку отличную от необходимой для входа в систему. Кроме всего прочего, в этом режиме не включается сеть, что исключает влияние внешних факторов на работу системы.

Разбираемся с однопользовательским режимом

Так что же за работа должна выполняться в таком состоянии системы? Чтобы ответить на этот вопрос, необходимо учесть огроменную разницу между Windows и Linux. Windows изначально была разработана для работы в консоли не более одного человека в данный момент времени. И она эффективно использует этот "однопользовательский" режим. С другой стороны, Linux чаще всего использовался для обслуживания сетевых приложений, предоставлял оболочку или X-сессию удалённым сетевым пользователям. Все эти дополнительные переменные весьма нежелательны, когда вы захотите провести сеанс обслуживания системы, вроде восстановления данных из резервной копии, модификации файловой системы, обновления системы с компакт-диска или чего-то подобного. Вот для этих случаев и необходимо использовать однопользовательский режим.

Уровни запуска

Вообще-то совсем нет необходимости в перезагрзуке машины для того, чтобы попасть в однопользовательский режим. Существует программа init, занимающаяся тем, что обеспечивает работу текущего "уровня запуска" системы. стандартные уровни запуска в Linux перечислены следующим образом:

- 0: Остановить компьютер
- 1: или s: Однопользовательский режим
- 2: Многопользовательский режим, без сети
- 3: Многопользовательский режим, текстовая консоль
- 4: Многопользовательский режим, графическая консоль
- 5: аналогично 4
- 6: Перезагрузка компьютера

Данные уровни запуска сильно варьируются от дистрибутива к дистрибутиву, так что посмотрите этот момент в документации к вашему дистрибутиву.

Команда telinit

Чтобы перейти в однопользовательский режим, необходимо вызвать команду telinit. Вот как надо инструктировать telinit для смены уровня запуска:

telinit 1

Как видно из примера выше, таким же образом мы можем потушить или перезагрузить компьютер. Команда telinit 0 выключит компьютер, а telinit 6 его перезагрузит. Когда вы используете telinit, то она запускает сценарий для запуска или остановки соответствующих служб.

Этикет при использовании уровней запуска

Однако, прошу заметить, что это весьма бесцеремонно так обходиться с пользователями, которые в этот момент могут работать в системе (и которые будут потом весьма злы на вас). Команда shutdown позволит вам обеспечить более разумный способ смены уровня запуска предупреждая при этом пользователей. Аналогичная возможность есть у команды kill, позволяющая отсылать различные сигналы процессу. Команда shutdown может быть использована для полного останова системы, перезагрузки, или перехода в однопользовательский режим. К примеру, вот как можно перейти в однопользовательский режим в течение пяти минут:

shutdown 5 Broadcast message from root (pts/2) (Tue Jan 15 19:40:02 2002): The system is going DOWN to maintenance mode in 5 minutes!

Если вы нажмёте сочетание клавиш Control-C, то вы остановите переход в однопользовательский режим. Сообщение выше появится на всех терминалах в системе, так что у пользователей будет разумное количество времени, чтобы сохранить свою работу и выйти из системы (Хотя тут ещё можно поспорить, является или не является 5 минут достаточным для этого).

Аргументы "now" и halt.

Если вы единственный пользователь этой системы, то вместо указания времени можно использовать аргумент now, как это сделано в примере ниже:

shutdown -r now

Возможностей для нажатия Control-C не будет в этом случае. Система пойдёт выключаться сразу же. И, наконец, опция для полного останова системы:

shutdown -h 1 Broadcast message from root (pts/2) (Tue Jan 15 19:50:58 2002): The system is going DOWN for system halt in 1 minute!

Уровень запуска по умолчанию

Как вы уже смогли разобраться, программа init весьма важна для работы системы. Сконфигурировать программу init можно с помощью файла /etc/inittab, который детально описывается в man-странице inittab(5). Но всё же приведём одну важную строчку оттуда:

grep ^id: /etc/inittab id:3:initdefault:

На моей системе уровень запуска 3 выставлен по умолчанию. Будет весьма полезным изменить это значение, если вы хотите загружаться сразу же в графическом режиме (обычно, это уровни 4 и 5). Чтобы изменить его, просто отредактируйте файл сменив значение в этой строке. Но будьте осторожными! Если вы измените что-то не то, то вам придётся проворачивать трюк с init=/bin/sh описываемый ранее.

Дополнительная информация

Дополнительная информация касающаяся этого раздела может быть найдена здесь:

- Документация по Sysvinit от RedHat

- [Раздел руководства для системных админстраторов описывающих init]

Квоты файловой системы

Введение в квотирование

Квотирование - это возможность предназначенная для отслеживания использования места пользователем или группой. Это весьма полезно для предотвращения ситуаций, когда один пользователь или группа захватывают чрезмерное пространство на файловой системе или полного занятия места. Квоты могут устанавливаться и управляться только суперпользователем root. В этом разделе я опишу как настроить ковотирование и эффективно всем этим управлять.

Поддержка ядром Linux

Квоты - возможность файловой системы. Поэтому нам необходимо чтобы она была включена в ядре. Дабы удостовериться, что квотирование поддерживается вашим ядром, необходимо сделать следующее с помощью команды grep:

cd /usr/src/linux # grep -i quota .config CONFIG_QUOTA=y CONFIG_XFS_QUOTA=y

Если же данная команда вернёт вам что-то менее подходящее (вроде значения того, что CONFIG_QUOTA не установлена), то вам будет необходимо пересобрать ядро с поддержкой квотирования. Процесс несложный, но выходит за рамки рассмотрения в данной статье. Если вы не знакомы с шагами необходимыми для сборки и установки нового ядра, то вам, наверное следует почитать данное руководство.

Поддержка файловой системой

Перед тем как погрузиться в управление квотами, пожалуйста, примите к сведению, что квотирование не поддерживается ядрами Linux серии 2.4 в полном объёме. Данные проблемы касаются поддержки квотирования в файловых системах ext2 и ext3, а ReiserFS их вообще не поддерживает. Поэтому за основу мы возьмём файловую систему XFS, где квоты поддерживаются должным образом.

Конфигурирование квот

Для того, чтобы начать конфигурировать квоты в системе, необходимо отредактировать файл /etc/fstab, дабы квоты сработали на указанных файловых системах:

grep quota /etc/fstab /usr/users /mnt/hdc1 xfs usrquota,grpquota,noauto 0 0 # mount /usr/users

Обратите внимание, что опции usrquota и grpquota не всегда включают поддержку квот. чтоюы удостовериться, что всё работает необходимо воспользоваться командой quotaon:

quotaon /usr/users

И дополнительно дам команду, которая отключает квотирование, если вы захотите от него отказаться в будущем:

quotaoff /usr/users

Итак, начиная с этого моменте прежде чем пробовать какие-то команды, убедитесь, что квотирование включено и работает.

Команда quota

Команда quota отображает степень заполнения диска пользователя, для всех смонтированных файловых систем. Опция -v показывает список файловых систем, где включено квотирование, но не указывает том, отведённый пользователю:

quota -v Disk quotas for user root (uid 0): Filesystem blocks quota limit grace files quota limit grace /dev/hdc1 0 0 0 3 0 0

В первой колонке, обзозначенной как blocks, показывается сколько пространства используется пользователем root, на каждой указанной в списке файловой системе. Следующие колонки quota и limit указывают лимиты для имеющиеся на текущем диске. Разницу между квотой и лимитом, а также значение колонки grace мы разберём позднее. Колонка files показывает сколько файлов принадлежит пользователю root на конкретной файловой системе. Следующие за ней колонки quota и limit относятся к ограничениям на количество файлов.

Просмотр квот

Любой пользователь может использовать команду quota для просмотра своего отчёта о доступных квотах как в предыдущем примере. Но только суперпользователь root имеет право просматривать информацию о квотах у всех остальных пользователей и групп. К примеру, мы говорим ,что у нас есть некая файловая система /dev/hdc1 смонтированная в /usr/users, с имеющимися там двумя пользователями jane и john. Для начала, давайте посмотрим использование диска Джейн и наложенные туда ограничения:

quota -v jane Disk quotas for user jane (uid 1003): Filesystem blocks quota limit grace files quota limit grace /dev/hdc1 4100 0 0 6 0 0

В данном случае мы видим, что у Джейн квоты установлены в нулевое значение, что означает отсутствие ограничений.

Команда edquota

А теперь давайте зададим пользователю jane некую квоту. Мы будем проделывать это с помощью команды edquota. прежде чем начать, взглянем сколько же свободного места есть в /usr/users:

df /usr/users Filesystem 1k-blocks Used Available Use% Mounted on /dev/hdc1 610048 4276 605772 1% /usr/users

Это не очень большая файловая система. Всего каких-то 600 мегабайт. Будет разумно, если мы дадим Джейн квоту, чтобы она не смогла использовать больше, чем ей выделено. Когда вы запускаете команду edquota, то появляется специальный временный файл для каждого пользователя или группы указанного в командной строке.

Команда edquota запускает редактор, который включает, добавляет или модифицирует квоты через этот временный файл:

edquota jane Disk quotas for user jane (uid 1003): Filesystem blocks soft hard inodes soft hard /dev/hdc1 4100 0 0 6 0 0

Аналогично выводу команды quota, рассмотренному выше, в колонках "blocks" и "inodes" указывается количество место и файлов используемых пользователем jane. Вы не можете редактировать количество блоков или инодов. Любое изменение будет отменяться системой. колонки "soft" и "hard" показывают размер квот Джейн, которые, как мы снова видим, являются неограниченными (как и тогда, число 0 означает отсутствие квот).

Разбираемся с edquota

Так называемое "мягкое" ограничение - это максимальное количество использованного дискового пространства выделенное пользователю Джейн на указанной файловой системе (другими словами, её квота). Если Джейн использует больше места, чем выделено ей согласно "мягкому" ограничению, то она получит письмо по электронной почте о превышении размера квоты. Так называемое "жёсткое" ограничение обозначает абсолютный лимит дискового пространства, который не может быть исчерпан. Если Джейн попытается использовать места больше, чем положено по этому лимиту, то она получит сообщение об ошибке "Дисковая квота исчерпана" и не сможет закончить начатую операцию.

Производим изменения

Итак, мы поменяли значение "мягкого" и "жёсткого" ограничения для пользователя Джейн и решили сохранить изменения в файл:

Disk quotas for user jane (uid 1003): Filesystem blocks soft hard inodes soft hard /dev/hdc1 4100 10000 11500 6 2000 2500

Запускаем команду quota, чтобы проверить изменения:

quota jane Disk quotas for user jane (uid 1003): Filesystem blocks quota limit grace files quota limit grace /dev/hdc1 4100 10000 11500 6 2000 2500

Копирование квот

Как вы помните, на этой файловой системе у нас ещё имеется пользователь Джон. Если мы желаем задать Джону те же квоты, что и Джейн, то необходимо использовать опцию -p с командой edquota, которая использует настройки Джейн в качестве прототипа для всех указанных пользователей. Это очень простой способ для задания квот группе пользователей:

edquota -p jane john # quota john Disk quotas for user john (uid 1003): Filesystem blocks quota limit grace files quota limit grace /dev/hdc1 0 10000 11500 1 2000 2500

Групповые ограничения

Мы так же можем использовать edquota, чтобы ограничить выделение дискового пространства основываясь на идентификаторе группы. К примеру, вот квота для группы users:

edquota -g users Disk quotas for group users (gid 100): Filesystem blocks soft hard inodes soft hard /dev/hdc1 4100 500000 510000 7 100000 125000

Смотрим изменившиеся квоты для группы:

quota -g users Disk quotas for group users (gid 100): Filesystem blocks quota limit grace files quota limit grace /dev/hdc1 4100 500000 510

Команда repquota

Просмотр квот для отдельно взятого пользователя используя quota бывает весьма утомителен, учитывая, что пользователей в системе может быть весьма много. Существует команда под названием repquota, которая собирает информацию обо всех квотах файловой системы и выводит симпатичный отчётец. Вот, к примеру, отчёт обо всех пользователях и группах имеющихся в /usr/users:

repquota -ug /usr/users *** Report for user quotas on device /dev/hdc1 Block grace time: 7days; Inode grace time: 7days Block limits File limits User used soft hard grace used soft hard grace -- root -- 0 0 0 3 0 0 john -- 0 10000 11500 1 2000 2500 jane -- 4100 10000 11500 6 2000 2500 *** Report for group quotas on device /dev/hdc1 Block grace time: 7days; Inode grace time: 7days Block limits File limits Group used soft hard grace used soft hard grace -- root -- 0 0 0 3 0 0 users -- 4100 500000 510000 7 100000 125000

Опции repquota

Существует парочка опций у этой команды, знать которые весьма нелишне. Команда repquota -a сообщает обо всех смонтированных на чтение/запись файловых системах, на которых выставлены квоты. Команда repquota -n не переводит идентификаторы групп и пользователей в их имена. Это поможет ускорить вывод больших отчётов.

Наблюдение за квотами

Если вы явлетесь системным администратором, то вам периодически надо посматривать за всем этим, чтобы не исчерпать выставленные лимиты. Самым простым сособом для этого является использоватние утилиты warnquota. Данная команда посылает уведомление по электронной почте при прохождении порога "мягкого" ограничения. Обычно, warnquota помещают в задачи планировщика cron.

Когда пользователь исчерпывает свой "мягкий" лимит, колонка grace в выводе команды quota отобразит срок действия - периода ожидания перед тем, как оно принудительно сработает.

Disk quotas for user jane (uid 1003): Filesystem blocks quota limit grace files quota limit grace /dev/hdc1 10800* 10000 11500 7days 7 2000 2500

По умолчанию срок срок такого периода для инодов и блоков - неделя.

Меняем срок периода ожидания

Значение периода ожидание можно сменить по своему желанию используя всё ту же edquota:

edquota -t

Полученные данные будут помещены во временный файл, который будет открыт текстовым редактором:

Grace period before enforcing soft limits for users: Time units may be: days, hours, minutes, or seconds Filesystem Block grace period Inode grace period /dev/hdc1 7days 7days

Текст в файле весьма исчерпывающ. Но убедитесь в том, что вы оставляете достаточно времени пользователям на получение письма, иначе все файлы будут удалены!

Проверка квот на этапе загрузки

Вы также можете проверять значение квот ещё при загрузке. Это можно проделать используя сценарий инициализации с использованием команды quotacheck. Пример такого скрипта доступен в Quota Mini HOWTO. Команда quotacheck также имеет возможность для исправления повреждённых квот. Познакомьтесь подробнее с этой командой с помощью man-страницы quotacheck(8).

Также помните об уже упомянутых командах quotaon и quotaoff. Вам необходимо будет поместить команду quotaon в сценарий загрузки, чтобы включить квоты. Дабы включить квотирование на всех поддерживаемых файловых системах, надо ввести:

quotaon -a

1 ... 15 16 17 18

Журналирование системы

Знакомимся с syslogd

Демон syslogd - служба предоставляющая развитый клиент-серверный механизм для журналирования сообщений от запущенных в системе программ. Syslogd принимает сообщения от демонов или программ, сортирует соообщения по типу и приоритету, затем записывает их в файл согласно заданным системным администратором правилам. В результате имеется ясный и единый подход к управлению системными журналами.

Чтение журналов

Давайте пройдёмся далее и посмотрим на содержимое журнала записанного программой syslog. После этого мы обратимся к конфигурации демона syslog. Стандарт FHS (вторая часть нашей серии руководств) определяет расположение файлов журналов в каталоге /var/log. Давайте воспользуемся командой tail, чтобы взглянуть последние 10 сообщений из файла messages:

cd /var/log # tail messages Jan 12 20:17:39 bilbo init: Entering runlevel: 3 Jan 12 20:17:40 bilbo /usr/sbin/named[337]: starting BIND 9.1.3 Jan 12 20:17:40 bilbo /usr/sbin/named[337]: using 1 CPU Jan 12 20:17:41 bilbo /usr/sbin/named[350]: loading configuration from

'/etc/bind/named.conf' Jan 12 20:17:41 bilbo /usr/sbin/named[350]: no IPv6 interfaces found Jan 12 20:17:41 bilbo /usr/sbin/named[350]: listening on IPv4 interface lo, 127.0.0.1#53 Jan 12 20:17:41 bilbo /usr/sbin/named[350]: listening on IPv4 interface eth0, 10.0.0.1#53 Jan 12 20:17:41 bilbo /usr/sbin/named[350]: running Jan 12 20:41:58 bilbo gnome-name-server[11288]: starting Jan 12 20:41:58 bilbo gnome-name-server[11288]: name server starting

Если вы помните наш обзор посвящённый обработке текстовых файлов, то знаете, что tail отображает последние строки в файле. В данном случае мы видим запущенный в системе сервер имён, который зовётся bilbo. Если бы мы развернули поддержку IPv6, то наверняка обратили своё внимание то, что он не может найти интерфейсы IPv6. Это указывает на потенциальную проблему. Дополнительно к этому мы также отмечаем, что пользователь только что запустил сессию окружения GNOME, видному по присутствию gnome-name-server.

Слежение за файлом журнала

Знающий системный администратор может воспользоваться командой tail -f, чтобы отслеживать текущий вывод в файл журнала:

```
# tail -f /var/log/messages
```

Например, теретически, мы хотим отследить потенциальную проблему с IPv6 используя команду указанную выше в отдельном терминале и перезапустив демон named для мгновенного получения сообщений от него. Данная техника весьма полезна при отладке. Ещё некоторые сисадмины любят оставлять в терминале постоянно запущенный tail -f, чтобы выборочно посматривать на события происходящие в системе.

Используем grep для журналов

Ещё одной полезной программой используемой для поиска по журналам событий является утилита grep, которая была описана во второй части. Для случая выше мы можем воспользоваться командой grep чтобы найти всё, что относится к слову "named":

```
# grep named /var/log/messages
```

Общий список журналов

Данная сводка показывает список обрабатываемых syslogd файлов журналов найденных в /var/log:

- messages: информационные сообщения и сообщения об ошибках для основным систмных программ и демонов.
- secure: Сообщения об авторизации и ошибки связанные с ними, хранится отдельно от файла messages для большей безопасности.
- maillog: Сообщения и ошибки связанные с почтой.
-

cron: Сообщения и ошибки связанные с планировщиком cron.

- spooler: Сообщения и ошибки связанные с UUCP.

Файл syslog.conf

Ну вот, настало самое время познакомиться с содержимым конфигурационного файла для демона syslog - syslog.conf (Примечание: Если вдруг вы не найдёте файла syslog.conf у себя на компьютере, то просто прочтите эту информацию для общего развития. Хотя скорее всего, у вас просто используется другой демон для журналирования). Взглянув на этот файл, вы увидите записи сходные для с журналами выше, плюс некоторые дополнительные. Данный файл имеет следующий формат записи: "условие (facility).приоритет (priority) действие (action)", где все эти поля определяются следующим образом:

Условие: Указывает на подсистему выдающую сообщение. Ключевые слова для условия: auth, authpriv, cron, daemon, kern, lpr, mail, news, syslog, user, uucp и local0 вместе с local7.

Приоритет: Указывает минимальную степень важности сообщения, что означает, что сообщение с указанным приоритетом и выше будут обрабатываться заданным правилом. Ключевые слова: debug, info, notice, warning, err, crit, alert, и emerg.

Действие: поле "действие" может содержать имя файла, терминала (/dev/console, к примеру), удалённую машину обозначенную префиксом "@", разделённый запятыми список пользователей, или отправку сообщений всем подключенным пользователям. Наиболее частоиспользуемое действие - задание имени файла.

Перезпуск и дополнительная информация

Надеемся, что данный обзор конфигурационного файла поможет вам обрести более уверенные знания о системе syslog. Вам следует прочесть справочную страницу syslog.conf(5) для большей информации перед тем как начать выполнять какие-то действия. Дополнительно посоветуем вам посмотреть syslogd(8), содержащем массу подробной информации.

Небольшое примечание перед тем, как задействовать изменения сделанные в файле конфигурации демона syslog: перезапуск демона с помощью посылки сигнала SIGHUP является верным решением, но ещё можно использовать команду killall, для упрощения задачи:

```
# killall -HUP syslogd
```

Примечания к безопасности

Следует остерегаться ситуации, когда файлы журналов которые записывает syslog в случае их отсутствия создаётся самой программой. Вне зависимости от настроек umask, такой файл читается всеми подряд. Если вы заботитесь о безопасности, то вам необходимо изменить права доступа файла на значение, когда их может читать и записывать только пользователь root. Дополнительно к сказанному хотелось бы рассмотреть программу logrotate (о ней ниже), которая может создавать файлы журналов с правильными параметрами безопасности. Демон syslog же всегда сохраняет текущие атрибуты файла, так что нет нужды беспокоиться о том, что файл был создан не им.

Демон logrotate

Файлы находящиеся внутри /var/log имеют свойство расти со временем и забивать собой всё свободное пространство на файловой системе. Поэтому рекомендуем воспользоваться программой наподобие logrotate, чтобы периодически сжимать файлы журналов. Данная программа запускается, как правило, ежедневно согласно настройкам cron. Она может быть настроена для архивирования, ротации, удаления или отправки журнала по электронной почте.

По умолчанию конфигурация logrotate ротирует файлы журналов еженедельно, сохраняя журналы за последние 4 недели (номер можно задать в конфигурационном файле) и сжимать последние журналы в архив для сохранения места. К тому же, эта программа может быть настроена для того, чтобы посылать сигнал SIGHUP демону syslog для того, чтобы он понял, что пора очищать файлы журналов и создавать новые.

Более подробная информация о logrotate содержится в man-странице logrotate(8), в которой имеется описание программы и синтаксис конфигурационного файла.

Дополнительная тема - klogd

Перед тем как покончить с syslog, хотелось бы подбросить дополнительного материала для наиболее любознательных читателей. Эти советы могут помочь вам сэкономить нервов при изучении тем, связанных с syslog.

Во-первых, syslogd является частью пакета sysklogd, который имеет в своём составе демон называемый klogd. Задача klogd получать информацию и сообщения об ошибках от ядра и передача их демону syslogd для дальнейшей обработки. Сообщения получаемые klogd весьма похожи на те, чтомы видели при выводе dmesg. Отличие в том, что dmesg выводит текущее содержимое кольцевого буфера ядра, в то время как klogd собирает сообщения передавая их syslog, где они не пропадут, даже если кольцо перестанет работать.

Ещё одна дополнительная тема - альтернативные способы журналирования

Во-вторых, хотелось расказать о существующей альтернативе стандартному пакету sysklogd. Альтернативные пакеты более более эффективны, легче в конфигурировании и более богаты возможностями нежели syslogd. Наиболее популярные альтернативы - Syslog-ng и Metalog. Вы можете изучить их, если не хватает текущих возможностей syslogd.

В-третьих, вы можете журналировать сообщения сами, используя собственные скрипты и команду logger. Для получения подробностей смотрите man-страницу logger(1).

Итоги и ресурсы

Итоги

Поздравяем! Вы всё же дошли до кнца данного руководства! Ну, почти. Существует пара тем, которые мы не рассматривали в виду ограниченного количества места в нашем материале. К счастью у нас есть несколько хороших ресурсов которые помогут вам быстро своить эти темы. Советуем вам изучить эти рукводства, если вдруг соберётесь сдавать сертификационный экзамен LPIC первого уровня.

Нам не хватило места для того, чтобы рассмотреть в этом руковдстве весьма важную тему касающуюся резервного копирования системы. И опять же к счастью, на сайте IBM developerWorks есть нужное руководство на эту тему под назанием "Резервное копирование вашей машины с Linux". по этому руководству вы сможете научиться резервному копированию операционной системы с помощью варианта программы tar под названием star. Вы также узнаете как использовать команду mt для контроля функций ленточных накопителей.

вторую тему, которую мы так и не рассмотрели - работа с планировщиком задач. Впрочем, существует хорошая документация по планировщику cron, созданная в университете штата Индиана. Планировщик cron разработан для выполнения задач в определённое время и является крайне нужной вещью для системного администратора.

Далее, мы рассмотрим некоторое количество ресурсов, которые могут оказаться весьма полезными при изучении тем рассматриваеимых в данной работе.

Ресурсы

Дабы узнать побольше о возможностях ограничений по файловой системе (квотировании) в Linux, рекомендуем просмотреть Linux Quota mini-HOWTO. Также советуем прочитать man-страницы quota(1), edquota(8), repquota(8), quotacheck(8), и quotaon(8) имеющиеся в вашей системе.

Дополнительная информация о загрузчиках и процессе загрузки вообще, может быть найдена здесь:

- Getting to know GRUB tutorial с сайта IBM DeveloperWorks.
- LILO Mini-HOWTO
- Домашняя страница GRUB
- Опции командной строки ядра находящиеся в /usr/src/linux/Documentation/kernel-parameters.txt
- Документация по Sysvinit от компании RedHat.

Ресурсы по ReiserFS:

- Обзор файловых систем в Funtoo, часть 1. Журналирование и ReiserFS.
- Обзор файловых систем в Funtoo, часть 2. Использование ReiserFS в Linux.

Ресурсы по XFS и JFS:

- Домашняя страница XFS
- Веб-сайт проекта JFS на сайте компании IBM.

Не забывайте про существование проекта linuxdoc.org. Там вы найдёте большую коллекцию бесценных руководств, учебников, HOWTO, ЧаВО и man-страниц. Периодически просматривайте Linux Gazette и LinuxFocus.

Существует также руководство для системных администраторов лежащих в разделе "Руководства" на сайте проекта Linuxdoc.org, которая очень хорошо завершает эту серию статей. Рекомендуем вам прочитать его! Вы также можете поискать работу Эрика Рэймонда "HOWTO по основам Unix и Интернет".

Ещё имеется серия статей по оболочке Bash с множеством примеров написанная Дэниэлом Роббинсом, которая покажет вам как использовать конструкции bash для написания собственных сценариев. Данная серия статей (особенно первая и вторая часть) будет превосходным дополнительным материалом для подготовки к экзамену LPIC 1-го уровня:

- Bash в примерах, часть первая. Основы программирования в Bourne-again shell.
- Bash в примерах, часть вторая. Больше об основах bash.
- Bash в примерах, часть третья. Исследуем систему ebuild в Gentoo.

Если вы не знакомы с редактором vi, то мы крайне рекомендуем ознакомиться со статьёй Дэниэла "Введение в vi - шпаргалка-методичка". Данное руководство плавно и быстро введёт в основы по этому весьма мощному текстовому редактору. Рассмотрите этот крайне важный для изучения материал, если вы не знаете как пользоваться vi.

Автор оригинального текста: **Дэниэл Роббинс.**

Перевод: **Буданов Евгений** aka r0g3r.