Основы Linux
от основателя Gentoo

Daniel Robbins
Chris Houser
Aron Griffis

Предисловие

Об этом руководстве

Добро пожаловать в первую из четырех частей обучающего руководства по основам Linux, разработанного чтобы подготовить вас к сдаче экзамена Linux Professional Institute 101. В нем вы познакомитесь с bash (стандартной оболочкой командного интерпретатора в Linux), узнаете о большинстве возможностей таких стандартных команд Linux, как ls, cp и mv, разберетесь в инодах, жестких и символьных ссылках, и многом другом. К концу этого руководства у вас сформируется некий фундамент знаний, и вы будете готовы к изучению основ администрирования Linux. К концу всего курса (8 частей), у вас будет достаточно навыков, чтобы стать системным администратором Linux и пройти сертификацию LPIC Level 1 от Linux Professional Institute, если конечно захотите.

Данная первая часть руководства отлично подходит для новичков в Linux, а также для тех пользователей, кто хочет освежить или улучшить свое понимание фундаментальных концепций Linux, таких, как копирование и перемещение файлов, создание символических и жестких ссылок, а также стандартных команд обработки текста, включая конвейеры и перенаправления. По ходу мы также дадим множество советов, подсказок и трюков, что делает это руководство насыщенным и практичным, даже для тех, кто уже имеет солидный опыт работы с Linux. Для начинающих большая часть этого материала будет новой, но более продвинутые пользователи Linux найдут это руководство отличным средством, чтобы разложить свои фундаментальные навыки по полочкам у себя в голове.

Тем, кто изучал первую версию этого руководства с целью, отличной от подготовки к экзамену LPI, возможно, что вам не нужно его перечитывать. Однако, тем же, кто планирует сдавать экзамены, стоит обязательно просмотреть данную исправленную версию.

Введение в bash

Оболочка

Если вы уже использовали Linux ранее, то наверняка знаете, что после входа в систему вас приветствует приглашение, которое выглядит примерно так:

```
$
```

На практике приглашение, которое вы видите, может немного отличаться. Например, оно может содержать имя хоста, имя текущей рабочей директории, или все вместе. Не зависимо от того, как выглядит ваше приглашение, есть одна несомненная вещь: программа, которая выводит это приглашение, называется оболочка интерпретатора команд (от англ. shell — оболочка, он же командная строка или терминал — прим. пер.), и, вероятнее всего, вашей командной оболочкой будет 'bash'.

А у вас запущен bash?

Вы можете убедиться, что используете bash, набрав:

```
$ echo $SHELL
/bin/bash
```

Если строчка выше выдает ошибку, или ответ не соответствует, возможно, что вы запустили другую оболочку. В этом случае большая часть этого руководства все еще будет полезна, но было бы значительно лучше для вас переключиться на bash, ради подготовки к экзамену 101.

О bash

Bash — это акроним от Bourne-again-shell, от англ. «ещё-одна-командная-оболочка-Борна» или «рождённая-вновь-командная оболочка» (тут игра слов Bourne/born — прим. пер.), и является оболочкой по умолчанию для большинства Linux-систем. Задача оболочки получать от вас команды, через которые вы взаимодействуете с Linux-системой. После того, как вы закончили ввод команд, вы можете выйти из оболочки (exit) или закончить сеанс (logout), в этом случае вы увидите приглашение входа в систему.

Кстати, вы также можете выйти из оболочки bash нажав control-D в приглашении.

Использование «cd»

Вы, возможно, уже обнаружили, что пялиться на приглашение bash — не самое впечатляющее занятие в мире. Ну что ж, давайте узнаем как путешествовать по нашей файловой системе. В приглашении введите пожалуйста следующую команду (без $):

```
$ cd /
```

Вы только что сообщили bash, что хотите работать в директории /, также известной, как корневая; все директории в системе имеют форму дерева, и / является его вершиной, т.е. корнем (в информатике деревья растут наоборот, корень вверху, а ветки спускаются вниз — прим. пер.). cd устанавливает директорию, в которой вы в данный момент работаете, также известную как «текущая рабочая директория».

Пути

Чтобы узнать текущую рабочую директорию в bash нужно набрать:

```
$ pwd
/
```

В примере с cd, аргумент / называется путь. Он сообщает cd куда мы хотим отправиться. В частности, аргумент / это абсолютный путь, что значит, что он задает расположение относительно корня дерева файловой системы.

Абсолютные пути

Ниже несколько из них:

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

Как можно заметить, у всех абсолютных путей есть одна общая черта, они начинаются с /. Указывая, допустим, /usr/local/bin в качестве аргумента для cd, мы сообщаем, что хотим попасть в / директорию, затем в usr директорию внутри нее, и так далее в local и bin, вниз по дереву. Абсолютные пути всегда отсчитываются начиная от / сперва.

Относительные пути

Другой тип пути называется «относительный путь». bash, cd, и другие команды всегда интерпретируют их относительно текущей директории. Относительные пути НИКОГДА не начинаются с /. Так, если мы сначала переместимся в /usr:

```
$ cd /usr
```

То, затем мы можем использовать относительный путь local/bin, чтобы попасть в директорию /usr/local/bin:

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

Использование ..

Относительные пути могут также содержать одну или более ".." директории. Директория ".." специальная; она указывает на родительскую директорию. Так, продолжая с примера выше:

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

Как видно, наша текущая директория теперь /usr/local. Мы смогли переместиться «назад» на одну директорию относительно текущей, где были до того.

Кроме того, мы также можем использовать ".." в существующем относительном пути, позволяющем нам переместиться в директорию «рядом» с той, в которой находимся:

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

Примеры относительных путей

Относительные пути могут быть чуточку более сложными. Ниже несколько примеров, попробуйте самостоятельно догадаться, где вы окажитесь после набора каждой из этих команд.

```
$ cd /bin
$ cd ../usr/share/zoneinfo

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

$ cd /usr/bin
$ cd ../bin/../bin
```

А теперь наберите их и проверьте свои предположения. ;)

Понимание .

Перед тем как мы закончим изучение cd, есть несколько моментов, которые необходимо прояснить. Во-первых, есть еще одна специальная директория ".", которая означает «текущая директория». Хотя она и не используется с командой cd, но часто используется для выполнения программы из текущей директории, как в следующем примере:

```
$ ./myprog
```

В данном случае будет запущена исполняемая программа myprog, находящаяся в текущей рабочей директории.

cd и домашняя директория

Если бы мы хотели переместиться в нашу домашнюю директорию, то могли бы набрать:

```
$ cd
```

Без каких либо аргументов cd переместит в вашу домашнюю директорию, которая будет /root для суперпользователя, или обычно /home/username (где username — имя пользователя в системе — прим.пер.) для любого другого пользователя. Но, что если мы хотим указать файл в нашей домашней директории? Может быть мы хотим передать путь к файлу в качестве аргумента нашей программе myprog. Если файл расположен в нашей домашней директории, мы можем набрать:

```
$ ./myprog /home/drobbins/myfile.txt
```

Однако, использования абсолютного пути вроде этого, не всегда удобно. К счастью, мы можем использовать символ ~ (тильда), чтобы проделать то же самое:

```
$ ./myprog ~/myfile.txt
```

Другие домашние директории пользователей

Bash воспримет одиночную ~ как указатель на вашу домашнюю директорию, но вы также можете использовать её для указания на домашние директории других пользователей. Например, если мы хотели сослаться на файл под названием fredsfile.txt в домашней директории пользователя fred, то могли бы набрать:

```
$ ./myprog ~fred/fredsfile.txt
```

Использование команд Linux

Знакомство с ls

А сейчас, мы быстренько пройдемся по команде ls. Скорее всего вы уже хорошо знакомы с этой командой, и знаете, что набрав ls получите список содержимого текущей рабочей директории:

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

Указав опцию -a, вы можете увидеть полный список, включая скрытые файлы и директории, начинающиеся с ".". Как видно в следующем примере, ls -a выводит также особые связывающие директории "." и "..":

```
$ 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
```

Развернутые списки директорий

Вы также можете задать одну и более директорий или файлов в командной строке с ls. Если вы укажите файл, то ls покажет вам только этот файл. А если зададите директорию, то ls выдаст ее содержимое. Опция -l очень удобна, когда необходимо посмотреть права доступа, владельца, время последнего изменения и размер в списке содержимого директории.

В следующем примере мы использовали опцию -l чтобы отобразить содержимое моей директории /usr:

```
$ 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 port -> 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 показывает информацию о правах доступа для каждого элемента. Чуть позже я объясню, как её интерпретировать. Следующая колонка содержит числа ссылок на каждый элемент файловой системы, позже мы вернемся к этому. Третья и четвертая колонки — это список владельцев и групп, соответственно. Пятая колонка — размер объекта. Шестая — время последнего изменения (mtime) объекта. И наконец, последняя колонка с именами объектов. Если файлы являются символическими ссылками, то вы увидите стрелку -> и путь, куда указывает эта символическая ссылка.

Смотрим на директории

Иногда вы захотите взглянуть на директорию, а не внутрь нее. В этом случае вы можете указать опцию -d, которая скажет ls рассматривать любую директорию, как внутреннюю:

```
$ 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 и инодный списки

Так вы можете использовать -d чтобы смотреть на директорию, но также можно использовать -R для противоположного: не только лишь глянуть внутрь директории, но и рекурсивно посмотреть все директории с файлами внутри нее! Мы не включим в руководство никакого примера вывода для этой опции (поскольку обычно он очень объемный), но возможно вы захотите попробовать несколько команд ls -R и ls -Rl, чтобы почувствовать как это работает.

Наконец, опция -i может использоваться для отображения числа инодов для объектов в списке файловой системы:

```
$ 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
```

Понятие инода

Каждому объекту файловой системы назначен уникальный индекс, называемый номером инода. Это может показаться банальным, но понятие инодов очень важно для понимания большинства операций в файловой системе. Рассмотрим например ссылки "." и "..", которые появляются в каждой директории. Чтобы полностью понять, чем на самом деле является директория "..", мы сперва взглянем на номер инода у /usr/local:

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

У директории /usr/local номер инода равен 5120. А теперь посмотрим номер инода у /usr/local/bin/..:

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

Как видно, директория /usr/local/bin/.. имеет такой же номер, как у /usr/local! Посмотрим, как можно справиться с этим шокирующим откровением. В прошлом мы полагали, что /usr/local сама является директорией. Теперь же, мы обнаружили, что фактически директория — это инод с номером 5120, и нашли, по меньшей мере, два элемента (называемых «ссылками»), которые указывают на данный инод. И /usr/local, и /usr/local/bin/.. — ссылки на 5120-ый инод. Хотя этот инод и существует только в одном месте на диске, тем не менее на него может быть множество ссылок.

На самом деле, мы даже можем увидеть общее количество ссылок ведущих на этот, 5120 инод, используя команду ls -dl:

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

Если взглянуть на вторую колонку слева, то видно, что на директорию /usr/local (инод 5120) ссылаются восемь раз. На моей системе на этот инод ведут следующие пути:

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

mkdir

Давайте быстренько пройдемся по команде mkdir, которая используется для создания новых директорий. Следующий пример создает три новых директории, tic, tac, и toe, все внутри /tmp:

```
$ cd /tmp
```

```
$ mkdir tic tac toe
```

По умолчанию, команда mkdir не создает для вас родительские директории; весь путь вплоть до последнего (создаваемого) элемента должен существовать. Так, если вы захотите создать вложенные директории won/der/ful, вам придется выполнить три отдельные команды mkdir:

```
$ 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
```

Однако, у mkdir есть очень удобная опция -p, которая говорит mkdir создавать любые отсутствующие родительские директории, как можете увидеть тут:

```
$ mkdir -p easy/as/pie
```

В целом очень просто. Чтобы узнать больше о команде mkdir наберите man mkdir и прочитайте инструкцию. Это же касается почти всех команд, рассмотренных здесь (например man ls), исключая cd, которая встроена в bash.

touch

Сейчас мы собираемся окинуть взглядом команды cp и mv, используемые для копирования, переименования и перемещения файлов и директорий. Но начнем обзор воспользовавшись командой touch, чтобы создать файл в /tmp:

```
$ cd /tmp
$ touch copyme
```

Команда touch обновляет «mtime» (время последней модификации —

прим. пер.) файла, если тот существует (вспомните шестую колонку в выводе ls -l). Если файл не существует, то новый, пустой файл будет создан. Сейчас у вас должен быть файл /tmp/copyme с нулевым размером.

echo

Теперь, когда файл существует, давайте добавим немного данных в него. Можно сделать это с помощью команды echo, которая принимает аргументы и печатает их на стандартный вывод. Сперва, команда echo сама по себе:

```
$ echo "firstfile"
firstfile
```

А сейчас, та же команда echo, но с перенаправлением вывода:

```
$ echo "firstfile" > copyme
```

Знак «больше» сообщает оболочке записывать вывод echo в файл по имени copyme. Этот файл будет создан, если не существовал, или перезаписан, если существует. Набрав ls -l, увидим, что файл copyme имеет размер в 10 байт, так как содержит слово firstfile и символ новой строки:

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

cat и cp

Чтобы вывести содержимое файла на терминал, используйте команду cat:

```
$ cat copyme
firstfile
```

Сейчас, мы можем воспользоваться основным вызовом команды cp для создания файла copiedme из оригинального copyme:

```
$ cp copyme copiedme
```

Ниже проверим, что это действительно разные файлы; у них отличаются номера инодов:

```
$ ls -i copyme copiedme
648284 copiedme   650704 copyme
```

mv

А сейчас давайте воспользуемся командой mv для переименования copiedme в movedme. Номер иноды останется прежний; однако, имя файла, указывающее на инод, изменится.

```
$ mv copiedme movedme
$ ls -i movedme
648284 movedme
```

Номер инода у перемещаемого файла остается прежним до тех пор, пока файл назначения находится в той же файловой системе, что и исходный файл. Мы подробнее рассмотрим файловую систему в третьей части нашего руководства.

Пока мы рассказываем об mv, давайте посмотрим, как еще можно использовать эту команду. mv, помимо возможности переименовать файлы, позволяет перемещать один или более файлов в другое место в иерархии директорий. Например, чтобы переместить /var/tmp/myfile.txt в директорию /home/drobbins (которая является моей домашней), я наберу (а мог бы воспользоваться ~ — прим. пер.):

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

После этого myfile.txt будет перемещен в /home/drobbins/myfile.txt. И если /home/drobbins располагается в другой файловой системе, нежели /var/tmp, команда mv скопирует myfile.txt в новую файловую систему и удалит его из старой. Как вы уже могли догадаться, когда myfile.txt

перемещается между файловыми системами, то myfile.txt на новом месте получает новый номер инода. Это все потому, что у каждой файловой системы свой независимый набор номеров инодов.

Мы также можем воспользоваться mv для перемещения нескольких файлов в одну директорию. К примеру, чтобы переместить myfile1.txt и myarticle3.txt в /home/drobbins, потребуется набрать:

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

Создание ссылок и удаление файлов

Жесткие ссылки

Мы уже упоминали термин «ссылка», когда рассказывали о взаимоотношениях между директориями (их именами) и инодами (индексным номерами, лежащими в основе файловой системы, которых мы не замечаем). Вообще в Linux существует два типа ссылок. Тип, о котором мы уже говорили ранее, называется «жесткие ссылки». Каждый инод может иметь произвольное число жестких ссылок. Когда уничтожается последняя жесткая ссылка, и не одна программа не держит файл открытым, то Linux автоматически удаляет его. Новые жесткие ссылки можно создать воспользовавшись командой ln:

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

Как видите, жесткие ссылки работают на уровне инодов, для указания конкретного файла. В Linux системах, для жестких ссылок есть несколько ограничений. В частности, можно создавать жесткие ссылки только на файлы, не на директории. Да-да, именно так; хотя "." и ".." являются созданными системой жесткими ссылками на директории, вам (даже от имени пользователя «root») не разрешается создавать любые свои собственные. Второе ограничение жестких ссылок состоит в том, что нельзя связать ими несколько файловых систем. Это значит, что у вас не получится создать жесткую ссылку с /usr/bin/bash на /bin/bash и если ваши директории / и /usr находятся в разных файловых системах (разделах — прим. пер.).

Символьные ссылки

В практике, символьные ссылки (или символические, иногда «симлинки» — от англ.) используются гораздо чаще, чем жесткие. Симлинки

— это файлы особого типа, которые ссылаются на другие файлы по имени, а не прямо по номеру инода. Они не спасают файлы от удаления; если файл, на который указывает ссылка, исчезает, то симлинк перестает работать, ломается.

Символические ссылки можно создать передав для ln опцию -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
```

В выводе ls -l символьные ссылки можно отличить тремя способами. Во-первых, обратите внимание на символ l в первой колонке. Во-вторых, размер символической ссылки равен количеству символов в ней (secondlink в нашем случае). В-третьих, последняя колонка в выводе показывает куда ведет ссылка с помощью интуитивного обозначения "->".

Симлинки детально

Символические ссылки в целом более гибкие, чем жесткие. Вы можете создавать символьные ссылки на любой объект файловой системы, включая директории. И благодаря тому, что их реализация основана на путях (не инодах), можно совершенно свободно создать символьную ссылку указывающую на объект другой файловой системы. Однако, сей факт также делает их сложными в понимании.

Предположим, что мы хотим создать ссылку в /tmp, которая указывает на /usr/local/bin. Нам следует набрать:

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

Либо, альтернативный вариант:

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

Как вы видите, обе символические ссылки указывают на одну директорию. Однако, если наша вторая символьная ссылка когда-нибудь будет перемещена в другую директорию, то она может «поломаться» из-за относительности пути:

```
$ 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
```

Потому, что директории /tmp/usr/local/bin не существует, мы больше не можем переместиться в bin2; другими словами, bin2 сейчас сломана.

По этой причине, избегать создания ссылок с относительной информацией о пути, иногда будет хорошей идеей. Тем не менее, существует множество случаев, где относительные символические ссылки крайне удобны. Рассмотрим пример в котором мы хотим создать альтернативное имя для программы в /usr/bin:

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

От имени суперпользователя мы хотим короткий синоним для keychain, такой, как kc. В этом примере у нас есть root-доступ, о чем свидетельствует измененное на "#" приветствие bash. Нам нужен root-доступ потому, что обычные пользователи не имеют прав создавать файлы в /usr/bin. От имени суперпользователя мы можем создать альтернативное имя для keychain следующим образом:

```
# 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 символьную ссылку под названием kc, которая указывает на файл /usr/bin/keychain.

Пока это решение будет работать, но создаст проблему, если мы решим 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 абсолютный путь для символической ссылки kc, то она все еще ссылается на /usr/bin/keychain, которого не существует с тех пор как мы переместили /usr/bin/keychain в /usr/local/bin.

Это привело к тому, что симлинк kc сейчас не работает. Как относительные, так и абсолютные пути в символьных ссылках имеют свои достоинства, и, в зависимости от вашей задачи, нужно использовать соответствующий тип пути. Часто, и относительный, и абсолютный путь, будут работать одинаково хорошо. Пример ниже будет работать, даже после перемещения обоих файлов:

```
# 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 набрав /usr/local/bin/kc. /usr/local/bin/kc указывает на программу keychain в той же директории, где находится kc.

rm

Итак, мы знаем как использовать cp, mv и ln, настало время узнать о том, как можно удалять объекты из файловой системы. Обычно это делается с помощью команды rm. Чтобы удалить файлы, просто укажите 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
```

Имейте ввиду, что под Linux, однажды удаленный файл, обычно исчезает на века. Поэтому многие начинающие системные администраторы используют опцию -i, когда удаляют файлы. Опция -i сообщает rm удалять файлы в интерактивном режиме — это значит спрашивать перед удалением любого файла. Например:

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

В примере выше команда rm запрашивает подтверждение на удаление

каждого из указанных файлов. В случае согласия, я должен был вводить «y» и нажать enter, дважды. Если бы я ввел «n», то файл бы остался цел. Или, если я сделал что-нибудь не так, я мог бы нажать Control-C и сбросить выполнение команды rm -i целиком — всяко до того, как это могло нанести какой-нибудь ущерб моей системе.

Если вы все еще учитесь пользоваться командой rm, то может быть полезным добавить при помощи вашего любимого текстового редактора следующую строку в ваш файл ~/.bashrc, и затем выйти (logout) и войти (login) в систему вновь. После этого, всякий раз, когда вы наберете rm, оболочка bash преобразует ее автоматически в команду rm -i. Таким образом, rm будет всегда работать в интерактивном режиме:

```
alias rm="rm -i"
```

rmdir

Для удаления директорий у вас имеется два варианта. Вы можете удалить все объекты внутри директории и затем воспользоваться rmdir для удаления самой директории:

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

Этот метод широко известен под названием «способ удаления директорий для лохов». Все реальные пацаны и админы-гуру съевшие пользователя собаку на этом деле, используют гораздо более удобную команду rm -rf, описанную далее.

Самый лучший способ удалить директорию состоит в использовании опций «рекурсивного принуждения» (recursive force) команды rm, чтобы приказать ей удалять указанную директорию, также как и объекты содержащиеся внутри:

```
$ rm -rf mydir
```

Обычно, rm -rf является наиболее предпочтительным методом для удаления древа директорий. Будьте очень осторожны, когда пользуетесь rm -rf, так как ее мощь может быть использована по обе стороны: добра и зла. =)

Использование джокеров

Знакомство с джокерами

В повседневном использовании Linux часто случается, когда вам нужно выполнить единичную операцию (например rm) на множестве объектов файловой системы за раз. В подобных ситуациях вписывать множество файлов в командную строку, зачастую, довольно обременительно:

```
$ rm file1 file2 file3 file4 file5 file6 file7 file8
```

Для решения этой проблемы, вы можете извлечь пользу из встроенной в Linux поддержки джокеров. Эта поддержка также называется «globbing» (по историческим причинам; в русском также известно как «универсализация файловых имен» — прим. пер.), позволяет указать множество файлов за раз, используя шаблон «дикой конкатенции». Bash и другие команды Linux интерпретируют этот шаблон просматривая диск в поисках файлов, которые ему удовлетворяют. Так, если у вас есть файлы file1 до file8 в текущей рабочей директории, то можете их удалить набрав:

```
$ rm file[1-8]
```

Или если просто хотите удалить все файлы с именами начинающимися с file, включай сам файл по имени file, вы можете набрать:

```
$ rm file*
```

Дикая конкатенция "*" совпадает с любым символом или набором символов, или даже с их отсутствием. Разумеет, glob-джокеры возможно использовать для гораздо большего, чем простое удаление файлов, как мы увидим в дальнейшем.

Неподходящие шаблоны

Если вам необходимо перечислить все объекты файловой системы в /etc, начинающиеся с «g», а также сам g, то можно ввести:

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

А сейчас о том, что случится если вы укажите шаблон, который не подходит ни под один объект файловой системы. В следующем примере мы попытались перебрать все файлы в /usr/bin, которые начинаются с «asdf» и оканчиваются на «jkl», потенциально включая и asdfjkl:

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

Вот что случится. Обычно, когда мы задаем шаблон, и если в него укладываются один или несколько файлов в подразумеваемой файловой системе, то bash заменяет наш шаблон на разделенный пробелами список всех подходящих объектов. Однако, когда с шаблоном нет совпадений, bash оставляет переданный аргумент с джокерами как есть. Так вот, затем ls не может найти файл /usr/bin/asdf*jkl и выдает нам ошибку. Основное правило тут такое: glob-шаблоны разворачиваются только если совпадают с объектами файловой системы. В противном случае, остаются не тронутыми и буквально передаются в вызов программы.

Синтаксис джокеров: * и ?

Так, мы уже посмотрели как работает globbing, теперь же стоит рассмотреть синтаксис джокеров. В качестве джокеров используются специальные символы:

* — совпадает с нулевым или большим количеством символов. Это значит: «тут может быть все что угодно, включая и ничего». Примеры:

- **`/etc/g*`** совпадает со всеми файлами в /etc, начинающимися с g и самим файлом g;
- **`/tmp/my*1`** совпадает со всеми файлами в /tmp, которые начинаются с my и заканчиваются 1, включая файл my1.

? — равен любому одному символу. Примеры:

- **`myfile?`** совпадает с любым файлом, чье имя составляет myfile и следующим за этим какой-либо один символ;
- **`/tmp/notes?txt`** совпадет, например, с /tmp/notes.txt и /tmp/notes_txt, если они существуют.

Синтаксис джокера: []

Этот джокер похож на **?**, но более точен. Чтобы его использовать, поместите любые символы, какие вам нужны, внутрь **[]**. Полученное выражение будет удовлетворять любому одному из этих символов. Вы также можете воспользоваться "-", для указания диапазона, и даже комбинации диапазонов. Примеры:

- **`myfile[12]`** совпадет с myfile1 и myfile2. Джокер сработает если хотя бы один из этих файлов существует в текущей директории;
- **`[Cc]hange[Ll]og`** совпадет с Changelog, ChangeLog, changeLog и changelog. Как можете заметить, использование скобочных джокеров очень удобно для указания вариантов с заглавными буквами;
- **`ls /etc/[0-9]*`** покажет все файлы в /etc, начинающиеся с десятичной цифры;
- **`ls /tmp/[A-Za-z]*`** отобразит все файлы в /tmp, которые начинаются с большой или маленькой латинской буквы.

Конструкция **[!]** эквивалентна конструкции **[]**, за исключением того, что вместо совпадения с символами внутри скобок, она удовлетворяет любому символу, который НЕ перечислен между [! и]. Пример:

- `rm myfile[!9]` удалит все файлы с названием myfile плюс один символ, кроме myfile9.

Предостережения о джокерах

Сейчас несколько предостережений, чтобы быть осторожными во время использования джокеров. Поскольку bash обрабатывает относящиеся

к джокерам символы (?, [,] и *) особым образом, вам надо особенно позаботиться, когда вы пишите аргумент для команды, содержащий эти символы. Например, если вы хотите создать файл, содержащий строку "[fo]*", то следующая команда может не дать желаемого результата:

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

Если шаблон [fo]* совпадет с какими-либо файлами в текущей рабочей директории, то вы обнаружите их имена внутри /tmp/mynewfile.txt, вместо ожидаемого [fo]*. Решение? Ну, одним из них будет огородить ваши символы в одиночные кавычки, которые сообщат bash не делать никаких раскрытий джокеров в них:

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

Используя этот подход, ваш новый файл будет содержать [fo]* буквально, как и предполагалось. Также, вы можете использовать экранирующую обратную косую черту, чтобы сообщить bash, что [,] и * должны интерпретироваться буквально, а не как джокеры:

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

Оба подхода (одиночные кавычки и экранирующая обратная косая черта) работают с одинаковым эффектом. Поскольку мы заговорили про обработку обратной косой черты, то неплохо бы было сейчас прояснить, что на случай, если вы хотите указать \ буквально, можно либо также взять ее в одиночные кавычки, либо набрать \\ (это будет обращено в \).

Примечание

Двойные кавычки работают также как и одиночные, но всё еще разрешают bash делать некоторую ограниченную обработку. Следовательно, одиночные кавычки — ваш лучший выбор, если вы действительно заинтересованны в буквальной передаче текста в команду. Получить больше информации о раскрытии джокеров можно набрав man 7 glob. Чтобы получить больше информации о кавычках в bash, наберите man 1 bash и прочитайте раздел под названием QUOTING. Если в ваши планы входит сдача экзаменов LPI, то рассматривайте это как свое домашнее задание. =)

Регулярные выражения

Что такое «регулярное выражение»?

Регулярное выражение (по англ. regular expression, сокр. «regexp» или «regex», в отечестве иногда зовется «регулярка» — прим. пер.) — это особый синтаксис используемый для описания текстовых шаблонов. В Linux-системах регулярные выражения широко используются для поиска в тексте по шаблону, а также для операций поиска и замены на текстовых потоках.

В сравнении с глоббингом

Как только мы начнем рассматривать регулярные выражения, возможно вы обратите внимание, что их синтаксис очень похож на синтаксис подстановки имен файлов (globbing), который мы рассматривали в первой части. Однако, не стоит заблуждаться, эта схожесть очень поверхностна. Регулярные выражения и глоббинг-шаблоны, даже когда они выглядят похоже, принципиально разные вещи.

Простая подстрока

После этого предостережения, давайте рассмотрим самое основное в регулярных выражениях, простейшую подстроку. Для этого мы воспользуемся «grep», командой, которая сканирует содержимое файла согласно заданному регулярному выражению. grep выводит каждую строчку, которая совпадает с регулярным выражением, игнорируя остальные:

```
$ 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
```

Выше, первый параметр для grep, это regex; второй — имя файла. grep считывал каждую строчку из /etc/passwd и прикладывал на нее простую regex-подстроку «bash» в поисках совпадения. Если совпадение

обнаруживалось, то grep выводил всю строку целиком; в противном случае, строка игнорировалась.

Понимание простой подстроки

В общем случае, если вы ищите подстроку, вы просто можете указать её буквально, не используя каких-либо «специальных» символов. Вам понадобиться особо позаботиться, только если ваша подстрока содержит +, ., *, [,] или \, в этом случае эти символы должны быть экранированы обратным слешем, а подстрока заключаться в кавычки. Вот несколько примеров регулярных выражений в виде простой подстроки:

- /tmp (поиск строки /tmp)
- "\[box\]" (поиск строки [box])
- "*funny*" (поиск строки *funny*)
- «ld\.so» (поиск строки ld.so)

Метасимволы

С помощью регулярных выражений используя метасимволы возможно осуществлять гораздо более сложный поиск, чем в примерах, которые недавно рассматривали. Один из таких метасимволов "." (точка), который совпадает с любым единичным символом:

```
$ grep dev.sda /etc/fstab
/dev/sda3      /          reiserfs  noatime,ro 1 1
/dev/sda1      /boot      reiserfs  noauto,noatime,notail 1 2
/dev/sda2      swap       swap      sw 0 0
#/dev/sda4     /mnt/extra     reiserfs  noatime,rw 1 1
```

В этом примере текст dev.sda не появляется буквально ни в одной из строчек из /etc/fstab. Однако, grep сканирует его не буквально по строке dev.sda, а по dev.sda шаблону. Запомните, что "." будет соответствовать любому единичному символу. Как вы видите, метасимвол "." функционально эквивалентен тому, как работает метасимвол "?" в glob-подстановках.

Использование []

Если мы хотим задать символ конкретнее, чем это делает ".", то можем использовать [и] (квадратные скобки), чтобы указать подмножество символов для сопоставления:

```
$ grep dev.sda[12] /etc/fstab
/dev/sda1  /boot  reiserfs   noauto,noatime,notail 1 2
/dev/sda2  swap   swap       sw 0 0
```

Как вы заметили, в частности, данная синтаксическая конструкция работает идентично конструкции "[]" при glob-подстановке имен файлов. Опять же, в этом заключается одна из неоднозначностей в изучении регулярных выражений: синтаксис похожий, но не идентичный синтаксису glob-подстановок, что сбивает с толку.

Использование [^]

Вы можете обратить значение квадратных скобок поместив ^ сразу после [. В этому случае скобки будут соответствовать любому символу который НЕ перечислен внутри них. И опять, заметьте что [^] мы используем с регулярными выражением, а [!] с glob:

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

Отличающийся синтаксис

Очень важно отметить, что синтаксис внутри квадратных скобок коренным образом отличается от остальной части регулярного выражения. К примеру, если вы поместите "." внутрь квадратных скобок, это позволит квадратным скобкам совпадать с "." буквально, также как 1 и 2 в примере выше. Для сравнения, "." помещенная вне квадратных скобок, будет интерпретирована как метасимвол, если не приставить "\". Мы можем получить выгоду из данного факта для вывода строк из /etc/fstab которые содержат строку dev.sda, как она записана:

```
$ grep dev[.]sda /etc/fstab
```

Также, мы могли бы набрать:

```
$ grep "dev\.sda" /etc/fstab
```

Эти регулярные выражения вероятно не удовлетворяют ни одной строчке из вашего /etc/fstab файла.

Метасимвол *

Некоторые метасимволы сами по себе не соответствуют ничему, но изменяют значение предыдущего символа. Один из таких символов, это * (звездочка), который используется для сопоставления нулевому или большему числу повторений предшествующего символа. Заметьте, это значит, что * имеет другое значение в регулярках, нежели в глоббинге. Вот несколько примеров, и обратите особое внимание на те случаи где сопоставление регулярных выражений отличается от glob-подстановок:

- ab*c совпадает с «abbbbc», но не с «abqc» (в случае glob-подстановки, обе строчки будут удовлетворять шаблону. Вы уже поняли почему?)
- ab*c совпадает с «abc», но не с «abbqbbc» (опять же, при glob-подстановке, шаблон сопоставим с обоими строчками)
- ab*c совпадает с «ac», но не с «cba» (в случае глоббинга, ни «ac», ни «cba» не удовлетворяют шаблону)
- b[cq]*e совпадает с «bqe» и с «be» (glob-подстановке удовлетворяет «bqe», но не «be»)
- b[cq]*e совпадает с «bccqqe», но не с «bccc» (при глоббинге шаблон точно так же совпадет с первым, но не со вторым)
- b[cq]*e совпадает с «bqqcce», но не с «cqe» (так же и при glob-подстановке)
- b[cq]*e удовлетворяет «bbbeee» (но не в случае глоббинга)
- .* сопоставим с любой строкой (glob-подстановке удовлетворяют только строки начинающиеся с ".")

- foo.* совпадет с любой подстрокой начинающийся с «foo» (в случае glob-подстановки этот шаблон будет совпадать со строками, начинающимися с четырех символов «foo.»).

Итак, повторим для закрепления: строчка «ac» подходит под регулярное выражение «ab*c» потому, что звездочка также позволяет повторение предшествующего выражения (b) ноль раз. И опять, ценно отметить для себя, что метасимвол * в регулярках интерпретируется совершенно иначе, нежели символ * в glob-подстновках.

Начало и конец строки

Последние метасимволы, что мы детально рассмотрим, это ^ и $, которые используются для сопостовления началу и концу строки, соответственно. Воспользовавшись ^ в начале вашего regex, вы «прикрепите» ваш шаблон к началу строки. В следующем примере, мы используем регулярное выражение ^#, которое удовлетворяет любой строке начинающийся с символа #:

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

Полностроочные регулярки

^ и $ можно комбинировать, для сопоставлений со всей строкой целиком. Например, нижеследующая регулярка будет соответсвовать строкам начинающимся с символа #, а заканчивающимся символом ".", при произвольном количестве символов между ними:

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

В примере выше мы заключили наше регулярное выражение в одиночные кавычки, чтобы предотвратить интерпретирование символа $ командной оболочкой. Без одиночных кавычек $ исчез бы из нашей

регулярки еще даже до того, как grep мог его увидеть.

FHS и поиск файлов

Стандарт иерархии файловой системы

Стандарт иерархии файловой системы (Filesystem Hierarchy Standard или сокр. FHS) — это документ который определяет схему директорий в Linux-системах. FHS разработан чтобы представить общую схему для упрощения независимой от дистрибутива разработки программного обеспечения, поскольку так все необходимое располагается одинаково в большинстве дистрибутивов. FHS определяет следующее дерево директорий (взято непосредственно из спецификации):

- / (корневая директория)
- /boot (статичные файлы загрузчика)
- /dev (файлы устройств)
- /etc (специфические для хоста конфигурационные файлы)
- /lib (основные разделяемые библиотеки и модули ядра)
- /mnt (точка монтирования для временных нужд)
- /pt (дополнительные пакеты ПО)
- /sbin (основные системные программы)
- /tmp (временные файлы)
- /usr (вторичная иерархия)
- /var (изменяемые данные)

Две независимые классификации в FHS

Спецификация FHS основывается на идее существования двух независимых классификаций файлов: разделяемые и неразделяемые, а также изменяемые и статичные. Разделяемые данные могут распределятся на несколько хостов; неразделяемые специфичны для конкретного хоста (как, например, конфигурационные файлы). Изменяемые данные могут изменяться; статичные не изменяются (за исключением установки и обслуживания системы).

Нижеследующая табличка резюмирует четыре возможные комбинации, с примерами директорий, которые попадают в данные категории. Опять же, эта таблица прямо из спецификации:

	разделяемые	неразделяемые
статичные	/usr /opt	/etc /boot
изменяемые	/var/mail /var/spool/news	/var/run /var/lock

Вторичная иерархия в /usr

Внутри /usr вы обнаружите вторичную иерархию, которая выглядит очень похоже на корневую файловую систему. Для /usr не критично существование во время включения машины, она может быть общим сетевым ресурсом (разделяема) или примонтирована с CD-ROM (статична). Большинство конфигурация Linux не используют «разделяемость» /usr, но ценно понимать полезность отличия между основной иерархией в корневой директории и вторичной иерархией в /usr.

Это все, что мы расскажем о стандарте иерархии файловой системы. Сам по себе документ довольно читабелен и вам стоит на него взглянуть. После его прочтения вы будете гораздо лучше понимать файловую систему Linux. Найти спецификацию можно здесь: http://www.pathname.com/fhs/.

Поиск файлов

Linux-системы зачастую содержат сотни тысяч файлов. Возможно, что вы достаточно умны, что никогда не теряете из виду ни один из них, но гораздо вероятнее, что временами вам требуется помощь для нахождения какого-либо файла. Для этого в Linux есть несколько разнообразных средств. Это введение поможет вам выбрать подходящее для решения вашей задачи.

PATH

Когда вы запускаете программу из командной строки, bash начинает просматривать список директорий в поисках программы которую вы указали. Например, когда вы вводите ls, bash в действительности не знает, что программа ls находится в /usr/bin. Вместо этого, он ссылается на переменную окружения называемую PATH, которая содержит список директорий разделенных двоеточием. Мы можем проверить значение PATH:

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

С таким значением PATH (у вас оно может быть другим) bash сначала проверит директорию /usr/local/bin, затем /usr/bin в поисках программы ls. Скорее всего, ls находится в /usr/bin, тогда на этой директории bash прекратит поиск.

Изменение PATH

Вы можете расширять переменную PATH, присваивая ей новое значение в командой строке:

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

Вы также можете удалять элементы из PATH, хотя это не так просто, поскольку вы не можете ссылаться в команде на существующий $PATH. Лучший вариант — это просто заново указать в PATH то, что вам нужно:

```
$ 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 сделать ваши изменения PATH доступными для процессов,

которые будут запускаться в командной оболочке, необходимо «экспортировать» их используя команду export:

```
$  export PATH
```

О команде «which»

Вы можете проверить, есть ли конкретная программа в вашем PATH используя which. В следующем примере мы видим, в каталогах PATH нашей системы, программы с названием sense нет:

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

В этом примере, ls успешно находится:

```
$ which ls
/usr/bin/ls
```

```
which -a
```

Наконец, вы должны знать о флаге -a, который укажет which показать вам все экземпляры программы в PATH:

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

whereis

Если вам необходимо больше информации о программе, чем просто ее расположение, вы можете воспользоваться командой whereis:

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

Здесь мы видим что ls находится в двух каталогах с общими исполняемыми файлами, /bin и /usr/bin. Кроме того, нам сообщили что есть

документация, которая находится в /usr/share/man. Это man-страница которую вы увидите, если введете man ls.

Программа whereis может использоваться для поиска расположения исходников и нестандартного поиска (имеется ввиду возможность искать файлы для которых отсутствуют маны, исходники или бинарники — прим. пер.). Также ей можно указать альтернативные пути для поиска. Обратитесь к man-странице для получения дополнительной информации.

find

Команда find это другой удобный инструмент в вашем арсенале. Используя find вы не ограничены лишь поиском программ; вы можете искать любые типы файлов, используя различные критерии поиска. Например, поищем в директории /usr/share/doc, файл который называется README:

```
$ 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
```

find и шаблоны

Вы можете использовать glob-шаблоны для аргументов -name, при условии что вы экранируете их кавычками или обратным слешем (таким образом они будут переданы команде в нетронутом виде, иначе они сначала будут развернуты bash'ем и уже после переданы команде). Давайте поищем все файлы README с расширением:

```
$ 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
[еще 578 строк опущено]
```

Игнорирование регистра в find

Конечно, вы можете игнорировать регистр при поиске:

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

Или, намного проще:

```
$ find /usr/share/doc -iname readme\*
```

Как видно, для поиска без учета регистра можно использовать опцию -iname.

find и регулярные выражения

Если вы знакомы с регулярными выражениями, вы можете использовать опцию -regex для поиска файлов с именами соответствующими шаблону. А также опцию похожую на -iname, которая называется -iregex и заставляет find игнорировать регистр в шаблоне. Пример:

```
$ 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
```

Однако в отличии от большинства программ, find требует чтобы регулярное выражение указывалось для всего пути, а не только его части. По этой причине, стоит в начале и конце шаблона ставить .*; простого использования xt в качестве шаблона будет недостаточно.

find и типы файлов

Опция -type позволяет искать в файловой системе файлы

определенного типа. Возможные аргументы для -type это: b (блочное устройство), c (символьное устройство), d (директория), p (именованый канал), f (обычный файл), l (символическая ссылка), и s (сокет). Например, поиск символической ссылки в /usr/bin, которая содержит в своем имени строку vim:

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

find и mtimes

Опция -mtime позволяет вам искать файлы основываясь на дате их последней модификации. Аргументом mtime является количество 24-часовых периодов, и наиболее полезным будет указывать перед аргументом плюс (означает «после») или минус (означает «перед»). Например, рассмотрим следующий сценарий:

```
$ 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
 Tue Jan 7 18:14:52 EST 2003
```

Вы можете найти файлы, которые были модифицированы за последние 24 часа:

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

Или файлы которые были изменены до текущего 24-часового периода:

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

Опция -daystart

Если вы дополнительно укажете опцию -daystart, периоды времени будут отсчитываться от начала сегодняшнего дня, а не от текущего времени. Например, здесь файлы созданные 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 позваляет искать файлы по их размеру. По-умолчанию, аргумент -size это количество 512-байтных блоков, но добавляя к опции суффикс, можно сделать вывод более понятным. Доступные суффиксы: b (512-байтные блоки), c (байт), k (килобайт), и w (2-байтные слова). Дополнительно, перед аргументом можно указать плюс («больше чем») или минус («меньше чем»).

Например, для поиска обычного файла в /usr/bin размер которого меньше 50 байт:

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

Работа с найденными файлами

Вы даже не представляете, что можно делать с найденными файлами! Итак, find может производить любые действия над файлами используя опцию -exec. Эта опция принимает строку команд для выполнения, которая оканчивается на ;, и заменяет все вхождения {} именем файла. Это проще всего понять на примере:

```
$ 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 видите, find это очень мощная команда. Она «выросла» за годы разработки UNIX и Linux. У find существует много других полезных опций. Вы можете прочитать о них в man-страничке.

locate

Мы уже рассмотрели which, whereis и find. Как вы уже наверно заметили, выполнение find может занять некоторое время, т.к. ей необходимо прочитать каждую директорию в которой выполняется поиск. Оказывается, что команда locate может ускорить процесс использую внешнюю базу данных, генерируемую updatedb (updatedb мы рассмотрим ниже).

Команда locate ищет совпадения любой части пути, а не только самого файла. Пример:

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

Использование updatedb

Во многих Linux системах есть «cron job» для периодического обновления базы. В случае если вызов locate вернул нижеописанную ошибку, вам необходимо запустить updatedb от root'a для генерации поисковой базы:

```
$ locate bin/ls
```

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

Работа программы updatedb может занять некоторое время. Если у вас шумный жесткий диск, вы услышите как он шуршит индексируя файловую систему. :)

slocate

Во многих Linux дистрибутивах, утилита locate была заменена на slocate. Как правило существует также ссылка на locate, так что вам не нужно запоминать, что именно имеется в системе. slocate означает «безопасный locate» (от англ. secure locate — прим. пер.). Он сохраняет информацию о правах доступа в поисковой базе, так что, обычные пользователи не смогут увидеть директории, которые они и так не смогли бы видеть. Используется slocate точно также как locate, но вывод программы может быть различными в зависимости от пользователя ее запустившего.

Управление процессами

Запуск xeyes

Для изучения управления процессами, какой-нибудь процесс необходимо сначала запустить. Убедитесь, что у вас запущен X (графический сервер — прим. пер.) и выполните следующую команду:

```
$ xeyes -center red
```

Вы увидите всплывающее окошко xeyes и красные глаза, следящие за курсором мыши. Также, обратите внимание, что у вас не появилось приглашения для ввода команд в терминале.

Остановка процесса

Чтобы вернуть приглашение, вы должны нажать Control-C (часто пишется как Ctrl-C или ^C):

Вы получили назад свое приглашение, но и окно xeyes исчезло. Фактически, процесс был «убит». Вместо завершения по Control-C, мы можем просто остановить процесс с помощью Control-Z:

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

На этот раз вы получите приглашение bash'a, а окно xeyes останется сверху. Если вы с ним немного поиграете, возможно заметите, что глаза заморожены на одном месте. Если окно xeyes будет перекрыто другим окном и затем снова открыто, вы увидите, что оно даже не перерисовалось. Процесс не делает ничего. Он на самом деле остановлен.

fg и bg

Чтобы процесс «растормошить» и запустить обратно, мы можем вывести его на передний план используя команду fg (от англ. foreground — прим. пер.):

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

А теперь продолжим его в фоне с помощью команды bg (от англ. backgroud — прим. пер.):

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

Прекрасно! Процесс xeyes сейчас запущен в фоновом режиме, а мы снова имеем приглашение bash.

Использование "&"

Если нам нужно сразу запустить xeyes в фоновом режиме (вместо использования Control-Z и bg), мы можем просто добавить "&" (амперсанд) в конец команды xeyes:

```
$ xeyes -center blue &
 [2] 16224
```

Несколько фоновых процессов

Теперь в фоне у нас одновременно работают красные и синие xeyes. Мы можем просмотреть список заданий с помощью jobs:

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

Число в левой колонке — это порядковый номер задания, который bash

присваивает ему при запуске. Плюс (+) у второго задания значит, что это «текущее задание», оно будет выведено на передний план при вводе fg. Вы также можете вывести на передний план конкретное задание указывая его номер; например, fg 1 сделает таковым красный xeyes. Следующая колонка это идентификатор процесса или сокращенно pid, любезно добавленный в вывод благодаря опции -l. Наконец, состояние обоих процессов «Running» (выполняется) и их командная строка справа.

Введение в сигналы

Чтобы убить, остановить, или продолжить процесс, Linux использует специальную форму взаимодействия, называемую сигналы. Отправляя сигнал некоторому процессу, вы можете его завершить, остановить, или сделать что-нибудь еще. Это то, что происходит на самом деле, когда вы нажимаете Control-C, Control-Z, или используете bg и fg — вы указываете bash отправить процессу определенный сигнал. Сигналы также можно отправить с помощью команды kill указав ей как параметр id процесса (pid):

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

Как можно заметить, kill не обязательно «убивает» процесс, хотя может и это. Используя опцию -s, kill может отправить процессу любой сигнал. Linux убивает, останавливает или продолжает процессы когда они получают SIGINT, SIGSTOP, или SIGCONT сигнал соответственно. Есть и другие сигналы, которые вы можете отправить процессам; некоторые сигналы могут обрабатываться внутри самих программ. Вы можете узнать о сигналах которые обрабатывает конкретная программа поискав в ее man'е секцию SIGNALS.

SIGTERM и SIGINT

Если вы хотите убить процесс, есть несколько вариантов. По-

умолчанию, kill отправляет SIGTERM, который отличается от SIGINT отправляемого по Control-C, но обычно имеет тот же эффект:

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

Полное убийство

Процесс может игнорировать оба сигнала, SIGTERM и SIGINT, либо по своему усмотрению, либо потому, что он остановлен, либо еще как-то «застрял». В этом случае, может быть необходимо использование большого молотка — сигнала SIGKILL. Процесс не может игнорировать SIGKILL:

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

nohup

Терминал в котором вы запускаете задания, называется терминалом управления заданиями. Некоторые шеллы (но не bash по-умолчанию), отправляют сигнал SIGHUP фоновым заданиям когда вы выходите, заставляя их завершаться. Для защиты процессов от такого поведения, используйте nohup когда запускаете процесс:

```
$ nohup make &
[1] 15632
$ exit
```

Используем ps для вывода списка процессов

Команда jobs, которую мы использовали ранее выводит только те процессы, которые были запущены в вашей сессии bash. Чтобы увидеть все процессы в вашей системе, используйте ps совместно 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 приведены только первые 5 процессов, поскольку обычно список процессов очень длинный. Команда дает вам «слепок» всего, что в данный момент выполняется на машине, однако в нем много лишней информации. Если бы вы, не указали ax, вы бы получили список только тех процессов, которые принадлежат вам, и которые есть в управляющем терминале. Команда ps x покажет все ваши процессы, даже те, которых нет в управляющем терминале. Если использовать ps a, то будет получен список процессов из терминалов всех пользователей.

Просмотр «леса» и «деревьев»

Вы также можете просмотреть и другую информацию о каждом процессе. Опция --forest позволяет легко просмотреть иерархию процессов и даст вам представление о том, как различные процессы в системе взаимосвязаны между собой. Если один процесс запускает другой процесс, то запущенный будет называться его потомком. В выводе --forest, родители находятся слева, а потомки появляются как ветки справа:

```
$ 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
```

«u» и «l» опции ps

Опции u и l могут быть использованы в любой комбинации с опциями a, x с целью получения более подробной информации 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 top

Если вы обнаружили, что запускаете ps несколько раз подряд, пытаясь рассмотреть происходящие изменения, возможно вам стоит воспользоваться top. Программа top отображает постоянно обновляющийся список процессов, наряду с другой полезной 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   gn-term
641    chouser       9      0      2936   2808   1416    S      0      0.1    1.0    2:13   sawfish
```

nice

Каждый процесс имеет свое значение приоритета, которое Linux использует для разделения времени CPU. Вы можете указать приоритет процесса при его запуске, с помощью команды nice:

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

С тех пор, как приоритет стал называться nice, он стал легче для запоминания, так, большее значение nice делает «хорошо» (nice — хорошо, замечательно; прим. пер.) другим процессам, позволяя им получить более приоритетный доступ к времени CPU. По-умолчанию, процессы запускаются с приоритетом 0, поэтому установка приоритета в 10 для oggenc значит, что он будет давать больше времени поработать другим процессам. Как правило, это означает, что oggenc даст возможность другим процессам выполняться со своей обычной скоростью, не зависимо от того, сколько времени процессора хочет сам oggenc. Вы могли видеть эти «уровни любезности» в колонке NI у ps и top ранее.

renice

Команда nice может изменять приоритет процессов только во время их запуска. Если вам необходимо изменить приоритет работающего процесса, воспользуйтесь командой 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     ?     2:14 sawfish
```

Обработка текста

Возвращаемся к перенаправлению

Ранее в этой серии руководств, мы видели пример использования >, оператора для перенаправления вывода команды в файл, как показано ниже:

```
$ echo "firstfile" > copyme
```

Помимо перенаправления вывода в файл, мы можем воспользоваться такой мощной фишкой оболочки как каналы (пайпы). Используя пайпы, мы можем передать вывод одной команды на вход другой. Рассмотрим следующий пример:

```
$ echo "hi there" | wc
1       2       9
```

Символ | используется для подключения выхода команды слева, ко входу команды справа от него. В примере выше, команда echo печатает в вывод «hi there» с символом перевода строки в конце. Этот вывод обычно появляется в терминале, но канал перенаправляет его на вход команде wc, которая показывает количество строк, слов и символов.

Пример с каналами (пайпами)

Вот другой простой пример:

```
$ ls -s | sort -n
```

В этом случае, ls -s обычно вывела бы текущую директорию на терминал, с указанием размера перед каждым файлом. Однако вместо этого, мы передаем вывод программе sort -n, которая численно отсортирует его. Это очень удобно для поиска файлов, которые занимают в директории больше всего места.

Следующие примеры посложнее, они демонстрируют мощь и удобство, которые можно получить используя каналы. Далее мы используем команды, которые еще не были рассмотрены, однако не заостряйте на них свое внимание. Вместо это, сконцентрируйтесь на понимании того, как работают пайпы и как вы можете использовать их в своей повседневной работе с Linux.

Распаковывающий канал

Для разархивации и распаковки файла, вы могли бы сделать следующее:

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

Недостаток такого метода — это создание промежуточного, разархивированного файла на диске. Поскольку tar может читать данные напрямую со своего входа (вместо указанного файла), мы можем получить тот же конечный результат используя пайп:

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

Вуухуу! Сжатый тарбол был распакован и мы обошлись без промежуточного файла.

Канал подлиннее

Вот еще один пример пайпа:

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

Мы используем cat для отправки содержимого myfile.txt команде sort. Когда sort получает данные на вход, она сортирует их построчно в алфавитном порядке, и отправляет в таком виде программе uniq. uniq удаляет повторяющиеся строки (кстати uniq, требует отсортированный список на входе) и отправляет результат на wc -l. Мы рассматривали

команду wc ранее, но без ее опций. Когда указывается опция -l, то команда выводит только количество строк, количество слов и символов в этом случае не выводятся. Вы увидите, что такой пайп распечатает количество уникальных строк в текстовом файле.

Попробуйте создать пару файлов в вашем текстовом редакторе. Используйте на них данный пайп и посмотрите на результат который вы получите.

Буря обработки текста начинается!

Теперь мы приступим к беглому осмотру команд Linux для стандартной обработки текстов. Поскольку сейчас мы рассмотрим множество программ, у нас не будет места для примеров по каждой из них. Вместо этого, мы призываем вас прочитать man-станицы приведенных команд (набрав man echo, например) и изучить каждую команду с ее опциями, потратив некоторое время на игру с ними. Как правило, эти команды печатают результат обработки на терминал, а не производят модификацию непосредственно файла. После этого беглого обзора, мы поглубже рассмотрим перенаправление ввода-вывода. Так что да, уже виден свет в конце тунеля. :)

echo печатает свои аргументы на терминал. Используйте опцию -e если хотите включить в вывод управляющие последовательности; например echo -e 'foo\nfoo' напечатает foo, затем перейдет на новую строку, затем снова напечатает foo. Используйте опцию -n чтобы запретить echo добавлять символ новой строки в конец вывода, как это сделано по-умолчанию.

cat напечатает содержимое указанного файла на терминал. Удобна как первая команда пайпа, например, cat foo.txt | blah.

sort выведет содержимое файла, указанного в командной строке, в алфавитном порядке. Естественно, sort также может принимать ввод из

пайпа. Наберите man sort чтобы ознакомиться с опциями команды, которые управлют вариантами сортировки.

uniq принимает уже отсортированный файл или поток данных (через пайп) и удаляет повторяющиеся строки.

wc выводит количество строк, слов и символов в указанном файле или во входном потоке (из пайпа). Введите man wc чтобы узнать, как настроить вывод программы.

head выводит первые десять строк файла или потока. Используйте опцию -n, чтобы указать, сколько строк должно отображаться.

tail печатает последние десять строк файла или потока. Используйте опцию -n, чтобы указать, сколько строк должно отображаться.

tac похожа на cat, но печатает все строки в обратном порядке, другими словами, последняя строка печатается в первую очередь.

expand конвертирует входные символы табуляции в пробелы. Опция -t указывает размер табуляции.

unexpand конвертирует входные пробелы в символы табуляции. Опция -t указывает размер табуляции.

cut используется для извлечения из входного файла или потока, полей разделенных указанным символом. (попробуйте echo 'abc def ghi jkl' | cut -d ' ' -f2,2 прим. пер.)

Команда **nl** добавляет к каждой входной строке ее номер. Удобно для распечатки.

pr разбивает файл на страницы и нумерует их; обычно используется для печати.

tr — инструмент трансляции (преобразования) символов; используется для отображения определенных символов во входном потоке на заданные символы в выходной поток.

sed — мощный потоко-ориентированный текстовый редактор. Вы можете узнать больше о sed из следующих руководств на сайте Funtoo:

awk — искусный язык построчного разбора и обработки входного потока по заданным шаблонам. Чтобы узнать больше о awk прочитайте следующую серию руководств на сайте Funtoo:

od разработан для представления входного потока в восьмеричном, шестнадцатеричном и т.д. формате.

split — эта команда используется для разделения больших файлов на несколько небольших, более управляемых частей.

fmt используется, чтобы выполнить «перенос» длинных строк текста. Сегодня она не очень полезна, поскольку эта возможность встроена в большинство текстовых редакторов, хотя команда достаточно хороша, чтобы ее знать.

paste принимает два или несколько файлов в качестве входных данных, объединяет построчно и выводит результат. Может быть удобно для создания таблиц или колонок текста.

join похожа на paste, эта утилита позволяет объединять два файла по общему полю (по-умолчанию первое поле в каждой строке).

tee печатает входные аргументы в файл и на экран одновременно. Это полезно, когда вы хотите создать лог для чего-либо, а также хотите видеть процесс на экране.

Буря закончилась! Перенаправление

Как и > в командной строке, вы можете использовать < для перенаправления файла, но уже на вход команде. Для многих команд, можно просто указать имя файла. К сожалению некоторые программы работают только со стандартным потоком ввода.

Bash и другие шелы поддерживают концепцию «herefile». Это позволяет давать входные данные команде в виде набора строк с последующей командой, означающей окончание ввода последовательности значений. Проще всего это показать на примере:

```
$ sort <<END
apple
cranberry
banana
END
apple
banana
cranberry
```

В приведенном выше примере, мы вводим слова apple, cranberry и banana, с последующим «END» для указания окончания ввода. Затем программа sort возвращается наши слова в алфавитном порядке.

Использование ">>"

Можно ожидать, >> будет в чем-то похожа на <<, но это не так. Она позволяет просто добавить вывод в файл, а не перезаписывать его каждый раз, как это делает >. Пример:

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

Уупс! Мы потеряли часть с «Hi»! А вот что мы имели ввиду:

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

Так то лучше!

Модули ядра

Знакомьтесь, «uname»

Команда uname дает множество интересной информации о вашей системе. Вот пример вывода на моей рабочей машине, после того, как я набрал uname -a, что говорит команде uname напечатать всю имеющуюся информацию:

```
$ uname -a
Linux inventor 2.4.20-gaming-r1 #1 Fri Apr 11 18:33:35 MDT 2003 i686
AMD Athlon(tm) XP 2100+ AuthenticAMD GNU/Linux
```

Подробнее о uname

Теперь, давайте посмотрим, какую же информацию о системе может дать uname

тип информации	аргумент	пример
имя ядра	-s	"Linux"
имя хоста	-n	"inventor"
релиз ядра	-r	"2.4.20-gaming-r1"
версия ядра	-v	"#1 Fri Apr 11 18:33:35 MDT 2003"
архитектура	-m	"i686"
процессор	-p	"AMD Athlon(tm) XP 2100+"
платформа	-i	"AuthenticAMD"
ос	-o	"GNU/Linux"

Интригующе! А что напечатает uname -a у вас?

Релиз ядра

А теперь небольшой трюк. Для начала выполните uname -r чтобы программа напечатала релиз ядра, которое работает в данный момент.

Теперь посмотрите в директорию /lib/modules и — опа! — Я уверен, что вы обнаружили каталог с точно таким же именем! OK, никакой магии, теперь самое время поговорить о значении каталогов в /lib/modules, а также объяснить, что такое модули ядра.

Ядро

Ядро Linux это сердце того, что обычно называют «Linux» — это кусок кода, который напрямую взаимодействует с вашим железом и абстрагирует от него обычные программы. Благодаря ядру, вашему текстовому редактору не нужно беспокоиться на какой диск, SCSI или IDE, а может даже в RAM, он производит запись. Редактор просто записывает в файловую систему, а ядро заботится обо всем остальном.

Введение в модули ядра

Итак, что такое модули ядра? Они представляют собой часть ядра, которая сохраняется на диске в специальном формате. По вашей команде, они подгружаются в работающее ядро и добавляют в него новую функциональность.

Поскольку модули ядра загружаются по требованию, вы можете иметь ядро поддерживающее дополнительную функциональность, которая в обычном состоянии будет выключена и недоступна.

Но «раз в сто лет», эти модули окажутся очень полезными и смогут быть загружены — часто автоматически — для поддержки диковинной файловой системы или устройства, которое вы редко используете.

Модули ядра вкратце

В общем, модули ядра позволяют по требованию добавить возможностей в работающее ядро. Без модулей, вам бы пришлось компилировать новое ядро и перезагружаться для того, чтобы добавить поддержку чего-нибудь нового.

lsmod

Для просмотра загруженных модулей на вашей системе используйте

команду lsmod:

```
# 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, на моей системе загружено достаточно немного модулей. vmnet и vmmon модули, обеспечиваю необходимую функциональность для VMWare Workstation, которая позволяет мне запускать виртуальные машины в окне рабочего стола. Модуль nvidia выпущен NVIDIA corporation и позволяет использовать 3D-ускорение в Linux.

Дальше у меня есть набор модулей, которые используются для поддержки USB устройств ввода — mousedev, hid, usbmouse, input, usb-ohci, ehci-hcd и usbcore. Имеет смысл сконфигурировать ваше ядро для поддержки USB модулей. Почему? Потому что USB девайсы это «plug and play» (подключай и работай) девайсы и если у вас есть поддержка USB в модулях, вы можете спокойно пойти и купить новое USB устройство, подключить его, и ваша система автоматически загрузит соответствующие модули для этого устройства. Это удобный способ сделать что-то.

Сторонние модули

Завершают этот список модули: emu10k1, ac97_codec и sound, которые вместе обеспечиваю поддержку моей звуковой карты Audigy.

Следует отметить, некоторые из моих модулей доступны прямо в исходниках ядра. Например, все USB-модули были скомпилированы из стандартных исходных текстов ядра Linux. Однако, nvidia, emu10k1 и VMWare-модули были получены из других источников. Это подчеркивает другую важную особенность модулей ядра — возможность сторонних производителей добавлять необходимую функциональность в ядро и включать ее прямо в запущенное ядро. Без перезагрузки.

depmod и компания

В моей папке /lib/modules/2.4.20-gaming-r1/, есть несколько файлов которые начинаются со строки «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 множество информации о различных зависимостях. В том числе, они содержат информацию о зависимостях для модулей — некоторые модули требуют загрузки других модулей перед тем как быть запущенными.

Как получить модули

Некоторые модули ядра разработаны для работы со специальными устройствами, как например emu10k1 — модуль для поддержки моей звуковой карты. Для этого типа модулей, приведенные выше файлы включают также информацию о PCI IDs и прочие идентификационные метки оборудования, которое они поддерживают. Эта информация может быть использована различными скриптами, например «hotplug» (который мы

рассмотрим в следующих руководствах) для автоматического определения оборудования и загрузки соответствующих модулей.

Использование depmod

Информация о зависимостях может становиться не актуальной, особенно в случае установки новых модулей. Чтобы ее обновить, просто введите depmod -a. Программа depmod просканирует модули из вашей папки /lib/modules и обновит информацию о зависимостях. Она делает это сканируя модули в /lib/modules и проверяя так называемые «symbols» внутри модулей.

Расположение модулей ядра

Итак, как выглядят модули ядра? Для ядра 2.4, все файлы модулей обычно находятся в /lib/modules и имеют имя оканчивающееся на ".o" (для 2.6 ".ko" — прим. ред.). Чтобы увидеть все модули из /lib/modules, введите 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
[список обрезан для краткости]
```

insmod vs. modprobe

Итак, как же подгрузить модуль в работающее ядро? Один из вариантов, использовать команду insmod и указать ей полный путь к модулю, который вы хотите загрузить:

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

Хотя, обычно модули загружают используя команду modprobe. Одна из приятных вещей, которую делает modprobe это автоматическая загрузка всех необходимых зависимостей для данного модуля. Кроме того, вам не нужно указывать полный путь и расширение загружаемого модуля.

rmmod и modprobe в действии

Давайте выгрузим наш модуль fat.o и загрузим его обратно используя modprobe:

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

Как видите, работа команды rmmod очень похожа на работу modprobe, но имеет противоположный эффект — она выгружает указанный модуль.

Ваши помощники modinfo и modules.conf

Можете воспользоваться командой modinfo, чтобы узнать пару интересных вещей о своих любимых модулях:

```
# modinfo fat
filename:    /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o
description: <none>
author:      <none>
license:     "GPL"
```

Также обратите внимание на файл /etc/modules.conf. Он содержит настройки для modprobe и позволяет изменять поведение modprobe. Например, указывать какие модули загруть до/после загрузки остальных, запускать скрипты до и после загрузки модуля, и многое другое.

Структура modules.conf

Синтаксис и функциональность modules.conf достаточно сложны, и мы

не будем в них сейчас углубляться (наберите man modules.conf чтобы узнать все подробности), но есть несколько вещей, которые вы должны знать об этом файле.

Во-первых, многие дистрибутивы генерируют этот файл автоматически из набора файлов в других директориях, таких как /etc/modules.d/. Например, в Gentoo Linux есть такая папка, и запуск команды update-modules прочитает все файлы из /etc/modules.d/ и объединит их в новый /etc/modules.conf. Поэтому, сделав свои изменения в файлах из /etc/modules.d/ запустите update-modules, если вы используете Gentoo. В Debian, процедура очень похожа, за исключением того, что папка называется /etc/modutils/.

Для ядер версии 2.6 аналогичные по функциональности файл и папка из каталога etc называются modprobe.conf и modprobe.d соответственно. Синтаксис там упрощен, смотрите man modprobe.conf.

Системная и сетевая документация

Типы системной документации в Linux

Существует три основных источника документации в Linux системах: страницы руководств (маны), info-страницы и поставляемая с приложениями документация в /usr/share/doc. В этом разделе мы рассмотрим каждый из этих источников, перед тем, как обращаться во внешний мир за дополнительной информацией.

Страницы руководств

Страницы руководств, или “man (от англ. manual — руководство) страницы”, ман-страницы, далее просто маны — это классическая форма справочной документации UNIX и Linux. В идеале, вы можете найти ман для любой команды, конфигурационного файла или библиотеки. Однако, на практике, Linux это бесплатное программное обеспечение и некоторые страницы руководств не были написаны или устарели. Тем не менее, маны остаются первым местом, куда следует обращаться за помощью.

Для доступа к ману просто введите man, а затем ваш запрос. Запустится программа-пейджер (просмотрщик, обычно less или more) со справочной информацией. Для того, чтобы закрыть её, нажмите клавишу q. Например, чтобы посмотреть информацию о команде ls, введите:

```
$ man ls
```

Знание структуры мана поможет быстро перемещаться к необходимой вам информации. Обычно, вы найдете следующие разделы в мане:

```
NAME (ИМЯ)              Имя и однострочное описание команды
SYNOPSIS (ОБЗОР)        Краткий обзор
DESCRIPTION (ОПИСАНИЕ)  Углублённое описание функционала команды
EXAMPLES (ПРИМЕРЫ)      Советы по использованию
SEE ALSO (СМОТРИ ТАКЖЕ) Связаные темы (обычно также ман-страницы)
```

Секции ман-страниц

Файлы, содержащие информацию манов хранятся в /usr/share/man (или в /usr/man на некоторых старых системах). В этой директории вы найдёте страницы руководств, упорядоченные по следующим секциям:

```
man1      Пользовательские программы
man2      Системные вызовы
man3      Библиотечные функции
man4      Специальные файлы
man5      Форматы файлов
man6      Игры
man7      Другое
```

Многочисленные ман-страницы

Некоторые темы существуют более чем в одной секции. Для демонстрации этого воспользуюемся командой whatis, показывающей доступные маны по теме:

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

В данном случае, man printf по-умолчанию обратится к странице в секции 1 (Пользовательские программы). Если мы пишем программу на языке C, нас вероятно больше интересует страница из секции 3 (Библиотечные функции). Вы можете вызывать ман необходимой секции указав это в командной строке, так для вызова printf(3) мы введём:

```
$ man 3 printf
```

Поиск нужного мана

Иногда сложно найти правильную ман-страницу по заданной теме. В этом случае можно воспользоваться man -k для поиска по разделам «ИМЯ» ман-страниц. Но будьте осторожны, поскольку осуществляется поиск по подстроке и что-то, навроде man -k ls, выдаст очень много всего! Вот пример использования уточняющего 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
words
```

Всё об “apropos”

Предыдущий пример не случаен. Во-первых, команда apropos полностью соответствует команде man -k. (На самом деле, я даже вам раскрою небольшой секрет. Когда вы запускаете man -k, фактически «за кулисами» стартует apropos). Во-вторых, существует команда makewhatis, которая сканирует все страницы в вашей Linux системе и создает базу данных для whatis и apropos. Обычно она запускается периодически из-под рута, чтобы поддерживать базу в актуальном состоянии:

```
# makewhatis
```

Для более подробной информации о команде «man» и её друзьях, вам стоит начать с изучения её собственной ман-страницы:

```
$ man man
```

MANPATH

По умолчанию, программа man будет искать ман-страницы в /usr/share/man, /usr/local/man, /usr/X11R6/man, и быть может в /opt/man. Возможно, вы захотите добавить новый пункт в этом пути поиска. Для этого, просто откройте /etc/man.conf в текстовом редакторе и добавьте строчку вроде такой:

```
MANPATH /opt/man
```

С этого момента, страницы руководств в директориях /opt/man/man* также будут найдены. Помните, что вам необходимо запустить makewhatis, чтобы добавить новые маны в базу whatis.

GNU info

Одно из ограничений страниц руководств это то, что они не поддерживают гипертекст, так что у вас не получится просто переходить от одного руководства к другому. Ребята из GNU увидели этот недостаток и ввели другой формат документации: инфо-страницы. Многие из программ GNU идут с расширенной документацией в формате инфо-страниц. Вы можете приступить к чтению инфо-страниц при помощи комманды «info»:

```
$ info
```

Простой вызов команды info даст список доступных инфо-страниц в вашей системе. Вы можете перемещаться по нему используя стрелки, переходить по ссылкам (которые обозначены звездочкой) с помощью клавиши Enter и выйти нажав q. Навигация основана на таковой в Emacs, так что если вы знакомы с этим редактором, вам будет легко освоиться. Чтобы познакомиться с Emacs, посмотрите руководство на developerWorks: Living in Emacs http://www-106.ibm.com/developerworks/edu/l-dw-linuxemacs-i.html.

Вы также можете указать нужную инфо-страницы в командной строке:

```
$ info diff
```

Чтобы получить больше информации об использовании просмотрщика инфо-страниц, попробуйте прочитать его собственную инфо-страницу. Вы сможете перемещаться по документу просто используя несколько клавиш о которых я уже упоминал:

```
$ info info
```

```
/usr/share/doc
```

Есть еще один источник помощи в вашей системе Linux. Большинство программ поставляются с дополнительной документацией в других форматах, таких как: простые текстовые файлы, PDF, PostScript, HTML.

Посмотрите в каталоге usr/share/doc (или /usr/doc на более старых системах). Вы найдете длинный список директорий, каждая из которых идет с определенным приложением на вашей системе. Поиск по этой документации может навести вас на очень ценную информацию, которая не доступна в манах или инфо-страницах, такую как учебники или дополнительная технической документация. Беглый взгляд указывает на то, что здесь очень много материала для чтения:

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

Фьюу! Вашим домашним заданием на этот вечер будет прочитать всего лишь половину (3791) этих документов. Учтите, завтра будет опрос. ;-)

Linux Documentation Project

В дополнение к системной документации, в интернете существует ряд отличных ресурсов посвященных Linux. "Linux Documentation Project" (LDP) http://www.tldp.org/ — это группа добровольцев, которые занимаются составлением полного набора свободной документации по Linux. Данный проект существует чтобы собрать различные части документации по Linux в определенным месте, где её будет легко искать и использовать.

Обзор LDP

LDP состоит из следующих разделов:

- Guides (руководства) — большие, очень серьезные пособия, такие как The Linux Programmer's Guide (http://www.tldp.org/LDP/lpg/index.html Руководство программиста Linux)
- HOWTOs — помощь по конкретной теме, например DSL HOWTO http://www.tldp.org/HOWTO/DSL-HOWTO/index.html
- FAQs — сборники ответов на часто задаваемые вопросы, навроде этого Brief Linux FAQ http://www.tldp.org/FAQ/faqs/BLFAQ
- Man pages — помощь по конкретной команде (это те же самые маны,

что вы видите в вашей системе, когда используете команду man).

Если вы не уверены какой раздел смотреть, вы можете воспользоваться богатыми возможностями поиска, который позволит найти всё, что есть по теме.

LDP в добавок предоставляет доступ к списку ссылок и ресурсов, таких как Linux Gazette http://www.tldp.org/LDP/LG/current/ и Linux Weekly News http://www.lwn.net/, а также к спискам рассылки и архивам новостей.

Списки рассылки

Списки рассылки являются, вероятно, самым важным средством взаимодействия разработчиков Linux. Зачастую проекты разрабатываются участниками живущими на большом расстоянии друг от друга, возможно даже на противоположных сторонах земного шара. Списки рассылки представляют метод взаимодействия, в котором каждый разработчик проекта может связаться со всеми остальными и вместе дискутировать посредством электронной почты. Один из самых известных списков рассылки разработчиков, это Linux Kernel Mailing List http://www.tux.org/lkml/ (список рассылки ядра Linux).

Еще о списках рассылки

В дополнение к разработке, списки рассылки могут предоставлять возможность задавать вопросы и получать ответы от знающих разработчиков или даже других пользователей. К примеру, отдельные дистрибутивы часто предоставляют список рассылки для новичков. Вы можете проверить на сайте вашего дистрибутива информацию о том, какие списки рассылки он предлагает.

Если вы уделили время чтобы прочитать LKML FAQ по ссылке выше, то возможно заметили, что подписчики на списки рассылок часто недружелюбно относятся к вопросам, которые часто повторяются. Всегда

разумно поискать в архивах рассылки перед тем, как задавать свой вопрос. Есть шансы, что это сэкономит и ваше время тоже!

Группы новостей

Новостные группы (англ. newsgroups) в интернете похожи на списки рассылки, но основаны на другом протоколе, который называется NNTP (Network News Transfer Protocol, что в переводе «Сетевой протокол передачи новостей»), а не на обмене электронной почтой. Чтобы иметь возможность общаться, вам придется установить NTTP-клиент, например slrn или pan. Основным преимуществом является тот факт, что вы можете принять участие в дискуссии, когда вам это нужно, а не постоянно смотреть как она ломится в ваш почтовый ящик :-)

Наибольший интерес представляют новостные группы начинающиеся с comp.os.linux. Посмотреть список групп вы можете на сайте LDP http://www.tldp.org/links/#ng.

Сайты поставщиков и прочие

Сайты различных дистрибутивов Linux зачастую предоставляют обновленную документацию, инструкции по установке, информацию о совместимости или несовместимости с оборудованием и другие средства поддержки, такие как поиск по базе знаний. Например:

- Redhat Linux http://www.redhat.com/
- Debian Linux http://www.debian.org/
- Gentoo Linux http://www.gentoo.org/
- SuSE Linux http://www.suse.com/
- Caldera http://www.caldera.com/
- Turbolinux http://www.turbolinux.com/

Поставщики аппаратного и программного обеспечения

В последние годы, многие поставщики устройств и программного

обеспечения добавили поддержку Linux для своих продуктов. На их сайтах вы можете найти информацию о том, какое оборудование поддерживает Linux, найти инструменты разработки программ, исходники, скачать драйвера для Linux под конкретное устройство, а также, узнать о других всевозможных Linux-проектах. Например:

- IBM и Linux http://www.ibm.com/linux/
- HP и Linux http://www.hp.com/products1/linux/
- Sun и Linux http://www.sun.com/linux/
- Oracle и Linux http://technet.oracle.com/tech/linux/content.html.

Модель прав доступа в Linux

Один пользователь, одна группа

В этом разделе мы рассмотрим права доступа в Linux и модель владения (ownership). Мы уже видели, что каждый файл принадлежит одному пользователю и одной группе. Это сама суть модели прав доступа в Linux. Вы можете узнать, какому пользователю и группе принадлежит файл в выводе команды ls -l.

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

В данном примере исполнимый файл /bin/bash принадлежит пользователю root и группе wheel. Модель прав доступа позволяет задать три независимых уровня прав на каждый объект файловой системы — для владельца, для группы и для всех остальных пользователей.

Понимание «ls -l»

Давайте рассмотрим вывод команды ls -l. Взглянем на первую колонку листинга:

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

Первое поле -rwxr-xr-x содержит символическое представление прав на данный файл. Первый знак (-) в этом поле определяет тип файла, в данном случае это обычный файл. Другие возможные значения:

'd' директория
'l' символическая ссылка
'c' устройство символьного ввода-вывода
'b' устройство блочного ввода-вывода
'p' FIFO
's' сокет

Три тройки

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

Остальная часть поля состоит из трех троек символов. Первая тройка представляет права владельца файла, вторая представляет права группы файла и третья права всех остальных пользователей.

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

Выше r означает, что чтение (просмотр данных содержащихся в файле) разрешено, w означает запись (изменение, а также удаление данных) разрешено и x означает исполнение (запуск программы разрешен). Собрав все воедино мы видим, что кому угодно разрешено читать содержимое и исполнять этот файл, но только владельцу (root) разрешено как либо модифицировать этот файл. Так что если нормальным пользователям разрешено копировать содержимое этого файла, то только root может изменять или удалять его.

Кто я?

Перед тем, как мы узнаем как изменить владельца или группу которой принадлежит файл, давайте сперва рассмотрим, как узнать вашего текущего пользователя и группу к которой вы принадлежите. Если вы не использовали команду su недавно, ваш текущий пользователь это тот, которым вы вошли в систему. Если вы часто используете su, вы можете не помнить пользователя под которым вы работаете в данный момент. Чтобы узнать под каким пользователем вы работаете, наберите whoami:

```
# whoami
root
# su drobbins
$ whoami
drobbins
```

В каких группах я состою?

Чтобы увидеть к каким группам вы принадлежите используйте команду groups:

```
$ groups
 drobbins wheel audio
```

Из этого примера видно, что я состою в группах drobbins, wheel, и audio. Если вы хотите посмотреть, в каких группах состоит другой пользователь, то передайте его имя в качестве аргумента.

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

Изменение пользователя и группы владельца

Чтобы изменить владельца или группу файла (или другого объекта) используется команды chown или chgrp соответственно. Сначала нужно передать имя группы или владельца, а потом список файлов.

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

Вы также можете изменить пользователя и группу одновременно используя команду chown в другой форме:

```
# chown root:wheel /etc/passwd
```

Вы не можете использовать команду chown без прав суперпользователя, но chgrp может быть использована всеми, чтобы изменить группу-владельца файла на ту группу, к которой они принадлежат.

Рекурсивное изменение прав

Команды chown и chgrp могут быть использованы с параметром -R, что

позволяет рекурсивно изменить владельца или группу у всех объектов в данной директории и ниже. Пример:

```
# chown -R drobbins /home/drobbins
```

Знакомство с chmod

chown и chgrp используются для изменения владельца и группы объекта файловой системы, но кроме них существует и другая программа, называемая chmod, которая используется для изменения прав доступа на чтение, запись и исполнение, которые мы видим в выводе команды ls -l. chmod использует два и более аргументов: метод, описывающий как именно необходимо изменить права доступа с последующим именем файла или списком файлов, к которым необходимо применить эти изменения:

```
$ chmod +x scriptfile.sh
```

В примере выше в качестве метода указано +x. Как можно догадаться, метод +x указывает chmod, что файл необходимо сделать исполняемым для пользователя, группы и для всех остальных. Если мы решим отнять все права на исполнение файла, то сделаем вот так:

```
$ chmod -x scriptfile.sh
```

Разделение между пользователем, группой и всеми остальными

До сих пор, наши примеры команды chmod влияли на права доступа всех трех наборов прав доступа — пользователя, группы и всех остальных пользователей. Часто бывает удобно изменить только один или два набора за раз. Чтобы сделать это, просто используйте специальный символ для обозначения набора прав доступа, который вам необходимо изменить, со знаком + или — перед ним. Используйте u для пользователя, g для группы и o для остальных пользователей.

```
$ chmod go-w scriptfile.sh
```

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

Сброс разрешений

Помимо переключения бит, отвечающих за права доступа, в состояние вкл/выкл, мы можем задать конкретные значения для всех сразу. Используя оператор равенства мы можем указать chmod, что хотим задать только указанные права доступа:

```
$ chmod =rx scriptfile.sh
```

Этой командой мы установили все биты чтения и исполнения и сбросили все биты записи. Если вы хотите задать значения конкретной тройки бит, то можете сделать это, указав ее символьное наименование перед оператором равенства:

```
$ chmod u=rx scriptfile.sh
```

Числовые режимы

До сих пор, мы использовали то что называется символическим способом указания прав доступа для команды chmod. Однако есть еще один достаточно распространенный способ указания прав: использование четырехзначных восьмеричных чисел. Этот синтаксис, называется числовым синтаксисом прав доступа, где каждая цифра представляет тройку разрешений. Например, в 1777, 777 устанавливают флаги о которых мы говорим в этом разделе, для владельца, группы, и остальных пользователей. 1 используется для указания специального бита прав доступа, который мы рассмотрим позже (смотрите «Неуловимая первая цифра» в конце раздела). Эта таблица показывает как транслируются права доступа на числовые значения.

Режим	Число
rwx	7
rw-	6

```
r-x       5
r--       4
-wx       3
-w-       2
--x       1
---       0
```

Числовой синтаксис прав доступа

Числовой синтаксис прав доступа особенно полезен когда требуется указать все разрешения для файла, как показано в следующем примере:

```
$ 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 назначили права доступа 0755, что равносильно комбинации прав -rwxr-xr-x.

umask

Когда процесс создает новый файл, он указывает, какие права доступа нужно задать для данного файла. Зачастую запрашиваются права 0666 (чтение и запись всеми), что дает больше разрешений, чем необходимо в большинстве случаев. К счастью, каждый раз, когда в Linux создается новый файл, система обращается к параметру, называемому umask. Система использует значение umask чтобы понизить изначально задаваемые разрешения на что-то более разумное и безопасное. Вы можете просмотреть текущие настройки umask набрав umask в командной строке:

```
$ umask
0022
```

В Linux-системах значением по умолчанию для umask является 0022, что позволяет другим читать ваши новые файлы (если они могут до них добраться), но не изменять их. Чтобы автоматически обеспечивать больший уровень защищенности для создаваемых файлов, можно изменить настройки umask:

```
$ umask 0077
```

Такое значение umask приведет к тому, что группа и прочие не будут иметь совершенно никаких прав доступа для всех, вновь созданных файлов. Итак, как работает umask? В отличие от «обычного» назначения прав доступа к файлу, umask задает какие права доступа должны быть отключены. Снова посмотрим на таблицу соответствия значений чисел и методов:

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

Воспользовавшись этой таблицей мы видим, что последние три знака в 0077 обозначают ---rwxrwx. Теперь вспомните, что umask показывает системе, какие права доступа отключить. Совместив первое и второе становится видно, что все права для группы и остальных пользователей будут отключены, в то время как права владельца останутся нетронутыми.

Знакомство с suid и sgid

В момент вашего входа в систему запускается новый процесс оболочки. Вы уже знаете об этом, но можете не знать о том, что этот новый процесс оболочки (обычно это bash) работает от имени вашего пользователя. И таким образом программа bash может обращаться ко всем файлам и директориям, владельцем которых вы являетесь. В действительности мы, как пользователи, полностью зависим от программ, выполняющих операции от нашего имени. И поскольку программы, которые вы запускаете, наследуют ваш пользовательский идентификатор, они не могут обращаться объектам файловой системы, к которым вам не

предоставлен доступ. К примеру, обычные пользователи не могут напрямую изменять содержимое файла passwd потому что флаг записи отключен для всех пользователей кроме root:

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

Однако, обычным пользователям тоже нужно иметь возможность хотя бы опосредованно менять содержимое /etc/passwd когда им понадобится сменить пароль. Но если пользователь не может изменить этот файл, как это сделать?

suid

К счастью, в модели прав доступа Linux имеются два специальных бита, называемых suid и sgid. Когда для запускаемой программы установлен бит suid, она будет работать от имени владельца исполняемого файла, а не от имени того, кто запустил программу. Теперь можем вернуться к вопросу с /etc/passwd. Если посмотрим на исполняемый файл passwd, увидим, что его владельцем является пользователь root:

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

Обратите внимание, что вместо x в триплете прав доступа владельца стоит s. Это означает что для этой конкретной программы установлены биты suid и права на запуск. По этой причине при запуске программы passwd она будет работать от имени пользователя root (со всеми правами доступа суперпользователя), а не пользователя, запустившего её. И поскольку passwd работает с правами суперпользователя, она способна редактировать /etc/passwd без каких либо сложностей.

Предупреждения о suid/sgid

Мы увидели как работает suid, sgid работает похожим образом. Она

позволяет программе наследовать права доступа группы, а не текущего пользователя.

Важно!

Немного разрозненная, но в то же время очень важная информация о suid и sgid. Во-первых, биты suid и sgid занимают те же поля в выводе команды ls -l. Если бит x тоже задан, соответствующие биты будут показаны как s (в нижнем регистре). Однако, если бит x не задан то он будет отображаться как S (в верхнем регистре).

Еще одно важное замечание: suid и sgid бывают удобны во многих ситуациях, но неправильное их использование может привести к появлению уязвимостей в защите системы. Лучше всего иметь как можно меньшее количество suid программ. Команда passwd — одна из немногих, которая должна быть suid.

Изменение suid и sgid

Способ установки и удаления битов suid и sgid чрезвычайно прост. Вот так мы задаем бит suid:

```
# chmod u+s /usr/bin/myapp
```

А в следующем примере мы снимаем флаг sgid с директории. Вы увидите, как бит sgid работает с директориями немного ниже:

```
# chmod g-s /home/drobbins
```

Права и директории

До текущего момента мы рассматривали права доступа с точки зрения обычных файлов. Когда речь заходит о директориях, появляются некоторые отличия. Директории используют те же флаги прав доступа, но их интерпретация имеет немного другой смысл.

Если для директории задан флаг чтения, то вы можете просматривать список содержимого директории; флаг записи означает, что вы можете создавать файлы в директории; и флаг исполнения означает, что вы можете войти в директорию и обращаться ко всем поддиректориям внутри. Без флага исполнения у вас не будет доступа к объектам файловой системы внутри директории. Без флага чтения объекты файловой системы внутри директории нельзя просмотреть, но к объектам внутри директории все еще можно обратиться, если вы знаете полный путь к объекту на диске.

Директории и флаг sgid

В случае же, если для директории установлен бит sgid, все объекты файловой системы, создаваемые внутри, наследуют группу директории. Эта возможность бывает кстати, когда вам необходимо создать дерево директорий и все они должны принадлежать одной группе. Это можно сделать вот так:

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

Теперь любые пользователи группы mygroup могут создавать файлы и директории внутри /home/groupspace и им также будет автоматически задана принадлежность группе mygroup. В зависимости от настроек umask для данного пользователя новые объекты файловой системы могут быть или не быть читаемыми, изменяемыми или исполняемыми другими пользователями группы mygroup.

Директории и удаление

По умолчанию директории в Linux ведут себя не самым удобным во многих ситуациях образом. Обычно кто угодно может переименовать или удалить файл внутри директории если у них есть права на запись в этой

директории. Для директорий, которыми владеют отдельные пользователи, такое поведение обычно не вызывает проблем.

Однако для директорий, которыми пользуется большое количество пользователей, в особенности /tmp и /var/tmp, это может вызвать целую кучу проблем. Все потому, что кто угодно может писать в эти директории, кто угодно может удалять и переименовывать чьи угодно файлы — даже если они им не принадлежат! Очевидно, довольно сложно использовать /tmp даже для временного хранения чего угодно, когда любой пользователь в любой момент может напечатать rm -rf /tmp/* и уничтожить файлы всех остальных.

Хорошая новость в том, что в Linux существует так называемый sticky бит. Когда для /tmp установлен sticky бит (командой chmod +t), единственные, кто могут удалить или переименовать файлы в /tmp — это либо владельцы этих файлов либо суперпользователь.

Неуловимый первый знак

В завершение этого раздела мы наконец обратим внимание на первый знак, используемый в численном синтаксисе. Он используется для задания битов sticky, suid и sgid:

suid	sgid	sticky	режим
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

Ниже приведен пример того, как использовать 4-значный режим для установки прав доступа на директорию, которая будет использоваться рабочей группой:

```
# chmod 1775 /home/groupfiles
```

В качестве домашней работы выясните что значит 1755 в настройках прав доступа. :)

Управление аккаунтами в Linux

Знакомьтесь, /etc/passwd

В этом разделе мы познакомимся с механизмом управления аккаунтами в Linux и начнем с файла /etc/passwd, в котором определены все пользователи, которые существуют в системе. Вы можете посмотреть свой файл /etc/passwd, набрав команду less /etc/passwd. Каждой строкой в /etc/passwd определяется аккаунт пользователя. Вот пример из моего /etc/passwd:

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

Как видите, в одной строке не так уж много информации. Каждая из них содержит несколько полей, разделённых ":". Первое поле отвечает за имя пользователя (drobbins), второе поле содержит «x». На устаревших Linux-системах второе поле содержало зашифрованных пароль для аутентификации, но фактически, сейчас все Linux-системы хранят эту информацию в другом файле. Третье поле отвечает за числовой пользовательский идентификатор, связанный с конкретным пользователем, а четвертое поле ассоциирует этого пользователя с конкретной группой; скоро мы увидим, где определена группа 1000. Пятое поле содержит текстовое описание аккаунта, в нашем случае это имя пользователя. Шестое поле определяет домашний каталог пользователя, седьмое — устанавливает стартовую оболочку пользователя, которая будет автоматически запускаться когда пользователь входит в систему.

/etc/passwd, советы и хитрости

Вы вероятно заметили, что в системе намного больше пользовательских аккаунтов, которые определены в /etc/passwd, чем тех, которые логинятся в систему на самом деле. Всё это потому, что различные компоненты Linux используют некоторые аккаунты для повышения безопасности. Обычно, такие системные аккаунты имеют идентификатор (uid) меньший 100, и у многих из них в качестве стартовой оболочки

установлена /bin/false. Так как эта программа ничего не делает, кроме как выходит и возвращает код ошибки, это эффективно препятствует использованию этих аккаунтов в качестве обычных аккаунтов для логина — т.е. они предназначены только для внутрисистемного пользования.

/etc/shadow

Итак, сами пользовательские аккаунты определены в /etc/passwd. Системы Linux вдобавок к /etc/passwd содержат его файл-компаньон /etc/shadow. Он, в отличие от /etc/passwd, доступен для чтения только суперпользователю и содержит зашифрованную информацию о паролях. Взглянем на образец строки из /etc/shadow:

```
drobbins:$1$12345678901234567890123456789O1:11664:0:-1:-1:-1:-1:0
```

Каждая строка определяет информацию о пароле конкретного аккаунта, поля в ней разделены знаком ":". Первое поле определяет конкретный пользовательский аккаунт, которому соответствует данная «теневая» запись. Во втором поле содержится зашифрованный пароль. Оставшиеся поля описаны в таблице ниже:

- поле 3 — количество дней с 01.01.1970 до момента, когда пароль был изменен
- поле 4 — количество дней до того, как будет разрешено сменить пароль («0» — «менять в любое время»)
- поле 5 — количество дней до того, как система заставит пользователя сменить пароль ("-1" — «никогда»)
- поле 6 — количество дней до истечения срока действия пароля, когда пользователь получит предупреждение об этом ("-1" — «не предупреждать»)
- поле 7 — количество дней после истечения срока действия пароля, по прошествии которых аккаунт будет автоматически отключен системой ("-1" — «не отключать»)
- поле 8 — количество дней, прошедшее с момента отключения этого

аккаунта ("-1" — «этот аккаунт включен»)

- поле 9 — зарезервировано для будущего использования

/etc/group

Теперь взглянем на файл /etc/group, который определяет группы в системе Linux. Вот примерная строка из него:

```
drobbins:x:1000:
```

Формат полей файла /etc/group следующий: первое поле определяет имя группы, второе поле — это поле остаточного пароля, которое сейчас просто зарезервировано x, и третье поле определяет числовой идентификатор для конкретной группы. Четвертое поле (которое пусто в примере выше) определяет всех членов группы.

Вспомните, что в нашем образце строки из /etc/passwd есть «ссылка» на группу с идентификатором 1000. Мы сможем поместить пользователя drobbins в группу drobbins, даже несмотря на отсутствие имени drobbins в четвертом поле /etc/group.

Примечания о группах

Замечание насчет соответствия пользователей с группами: на некоторых системах каждый новый логин-аккаунт связан с группой, имеющей то же имя (и обычно идентификатор). На других системах все логин-аккаунты будут принадлежать к одной группе пользователей. Какой из этих методов выбрать зависит от вас. Создание соответствующей группы для каждого пользователя имеет преимущество в том, что позволяет им более легко контролировать их собственный доступ просто помещая доверенных друзей в свою личную группу.

Ручное создание пользователей и групп

Теперь, я покажу как создать аккаунты для пользователя и группы. Лучший путь узнать как это сделать это добавить нового пользователя в

систему вручную. Для начала убедитесь что вашей переменной окружения EDITOR соответствует ваш любимый редактор:

```
# echo $EDITOR
vim
```

Если это не так, то вы можете установить переменную EDITOR, набрав что-то, вроде:

```
# export EDITOR=/usr/bin/emacs
# vipw
```

Теперь ваш редактор должен быть запущен с уже загруженным /etc/passwd экране. Изменяя системные файлы passwd и group обязательно используйте команды vipw и vigr. Они имеют повышенные меры предосторожности, оберегая ваши файлы от участи быть испорченными.

Редактирование /etc/passwd

Итак, у вас уже есть готовый файл /etc/passwd, добавьте теперь следующую строку:

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

Мы только что добавили пользователя «testuser» с идентификатором 3000. Мы определили его в группу с таким же идентификатором, которую еще не создали. Но мы можем добавить его к уже имеющейся группе пользователей, если нужно. У этого пользователя установлен комментарий, гласящий «LPI tutorial test user», домашний каталог установлен как "/home/testuser", а командная оболочка — как "/bin/false", в целях безопасности. Если бы мы создавали не тестовый аккаунт, мы бы установили командную оболочку как "/bin/bash". Отлично, теперь сохраните файл и выходите.

Редактирование /etc/shadow

Сейчас нам нужно добавить запись в /etc/shadow для этого пользователя. Для этого наберите vipw -s. Вас как всегда встретит ваш любимый редактор в котором уже открыт файл /etc/shadow. Теперь скопируйте строку существующего пользовательского аккаунта (того, у которого есть пароль и запись которого длиннее стандартных записей системных аккаунтов)

```
drobbins:$1$12345678901234567890123456789O1:11664:0:-1:-1:-1:-1:0
```

Замените имя пользователя в скопированной строке на имя вашего пользователя и убедитесь что все поля (особенно старый пароль) установлены как вам надо:

```
testuser:$1$12345678901234567890123456789O1:11664:0:-1:-1:-1:-1:0
```

Теперь сохраните и закройте.

Установка пароля

Вы вернетесь к командной строке. Теперь, самое время задать пароль для вашего нового пользователя.

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

Редактирование /etc/group

Теперь /etc/passwd и /etc/shadow готовы и самое время как следует настроить /etc/group. Для этого, наберите:

```
# vigr
```

Перед вами появится ваш /etc/group файл, готовый для редактирования. Итак, если ранее вы решили добавить созданного пользователя к уже имеющейся группе, то вам не понадобиться создавать

новую группу в /etc/groups. Если это не так, вам нужно добавить новую группу для этого пользователя, введите следующую строку:

```
testuser:x:3000:
```

Теперь сохраните и закройте.

Создание домашней директории

Мы почти закончили. Выполните следующие команды для создания домашнего каталога testuser'a:

```
# cd /home
# mkdir testuser
# chown testuser:testuser testuser
# chmod o-rwx testuser
```

Наш каталог пользователя на месте и аккаунт готов к использованию. Уже почти готово. Если вы собираетесь использовать этот аккаунт, вам надо будет воспользоваться vipw для смены стартовой оболочки на /bin/bash, так, чтобы пользователь смог войти.

Утилиты администрирования учетных записей

Вы уже знаете как вручную добавить новые аккаунты и группы, давайте же теперь рассмотрим различные, экономящие время, утилиты для управления аккаунтами под Linux. Из-за некоторых ограничений мы не будем рассматривать множество деталей, описывающих эти команды. Запомните — вы всегда можете получить больше информации о какой-либо команде если посмотрите её man-страничку. Если вы планируете сдавать LPIC 101 экзамен, вам следует провести побольше времени на ознакомление с каждой из этих команд.

newgrp — По умолчанию, любой файл, который создает пользователь, сразу же присваивается к группе, в которой он состоит, определенной в /etc/passwd. Если пользователь принадлежит к другим группам, он или она

может набрать newgrp thisgroup чтобы стать членом группы thisgroup. Затем, любые новые созданные файлы унаследуют членство в thisgroup.

chage — Команда chage используется для просмотра и изменения настроек срока действия паролей, сохраненных в /etc/shadow.
gpasswd — Основная утилита управления группами.

groupadd/groupdel/groupmod — Используются для добавления/удаления/изменения групп в /etc/group.

useradd/userdel/usermod — Используются для добавления/удаления/изменения пользователей в /etc/passwd. Эти команды могут выполнять и другие полезные функции. Смотрите man для получения дополнительной информации.

pwconv/grpconv — Используются для преобразования passwd и group файлов старого образца в новые shadow passwords. Фактически, все Linux системы уже используют shadow passwords, так что вам никогда не придется использовать эти команды.

Настройка пользовательского окружения

Знакомство с «fortune»

У вашего окружения есть много полезных опций, которые вы можете изменить по своему усмотрению. Однако до сих пор мы не обсуждали как восстанавливать эти настройки каждый раз, когда вы входите в систему, исключая то, чтобы каждый раз набирать их заново. В этом разделе мы рассмотрим настройку вашего окружения посредством редактирования стартовых конфигурационных файлов.

Для начала, давайте покажем дружелюбное сообщение когда вы будете входить в систему. Чтобы увидеть пример такого сообщения, запустите fortune:

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

(приложение fortune может быть не установлено, запустите установку в пакетном менеджере вашего дистрибутива, например apt-get install fortune)

.bash_profile

Теперь давайте сделаем так чтобы fortune запускалось при каждой авторизации. Используя любимый текстовый редактор отредактируйте файл .bash_profile в вашей домашней директории. Если такого файла не существует, создайте его. Вставьте в его начало:

```
fortune
```

Попробуйте выйти из системы и зайдите обратно. До запуска менеджера дисплея, такого как например xdm, gdm или kdm, вы увидите веселое приветствие, когда войдете:

```
mycroft.flatmonk.org login: chouser
Password:
```

```
Freedom from incrustations of grime is contiguous to rectitude.
$
```

Оболочка входа.

При запуске bash проходит файл .bash_profile в вашей домашней директории, запуская каждую строчку как будто набирая ее в командной строке. Это называется интерпретацией файла (file sourcing).

Bash может работать различным образом в зависимости от того, как он запущен. Если он запущен как оболочка входа, то будет работать, как описано выше — сначала обработая общесистемный /etc/profile, а затем ваш личный ~/.bash_profile.

Существуют два способа запуска bash в качестве оболочки входа. Первый используется когда вы впервые входите в систему: bash запускается с именем процесса -bash. Можно увидеть это в выводе списка процессов:

```
$ 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
```

Вы, вероятно, увидите более длинный список, но но в нем будет как минимум одна строка с чертой перед именем вашей оболочки, как -bash в примере выше. Эта черта используется оболочкой, чтобы определить, была ли она запущена при авторизации.

Понимание --login

Второй способ запустить bash как оболочку входа — при помощи опции —login. Время от времени эта возможность используется эмуляторами терминала (вроде xterm) чтобы их сессия bash выглядела как при изначальном входе в системе.

После авторизации можно запустить много копий вашей оболочки. У

тех из них которые запущены без опции --login или не имеют черточки перед именем процесса, поведение будет иным нежели при запуске с авторизацией. Они предоставляют вам командную строку, однако, называются они интерактивными оболочками. Если bash запущена интерактивно, без авторизации, она проигнорирует /etc/profile и ~/.bash_profile вместо этого обработает ~/.bashrc.

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

Проверка на интерактивность

Иногда bash обрабатывает ваш ~/.bashrc не будучи запущенным интерактивно, например при использовании команд типа rsh или scp. Это важно помнить потому как вывод текста, как в примере с командой fortune выше, может испортить эти неинтерактивные сессии bash. Хорошей идеей является использование переменной PS1 для проверки факта интерактивности текущей сессии перед выводом текста:

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

/etc/profile и /etc/skel

Как системный администратор, вы отвечаете за файл /etc/profile. Так как им руководствуются все кто впервые входят в систему, важно держать его в рабочем состоянии. Это также очень мощный инструмент, используемый для того чтобы заставить вещи работать правильно для новых пользователей сразу после того как они войдут используя свою новую учетную запись.

Однако, существует вариант, при котором, настройки с одной стороны,

могут быть выставлены в значения по-умолчанию для новых пользователей, а с другой, могут быть легко ими изменены при необходимости. Как раз для этого и существует директория /etc/skel. При использовании команды useradd для создания новой учетной записи, все файлы из /etc/skel копируются в домашний каталог нового пользователя. Это означает, что вы можете положить, например, .bash_profile и .bashrc в /etc/skel для более комфортного начала работы нового пользователя.

export

Переменные в bash могут быть помечены таким образом, что они будут устанавливаться во всех вновь запускаемых командных оболочках. Это означает, что они обозначены как внешние (export). Вы можете заставить bash отобразить список всех таким образом обозначенных внешних переменных в 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"
```

Экспортирование переменных

Если переменная не отмечена на экспорт, её значение не будет задано для новых запускаемых оболочек. Но вы можете отметить переменную на экспорт, передав её встроенной команде export:

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

```
bar
```

В этом примере были заданы обе переменные FOO и BAR, но только BAR была помечена для экспорта. Когда был запущен новый bash он потерял значение переменной FOO. Если вы выйдите из этого нового bash, вы увидите что первоначальные значения обоих переменных FOO и BAR не изменились.

```
$ exit
$ echo $FOO $BAR
foo bar
```

Export и set -x

В связи с вышеописанным поведением, переменные могут быть указаны в ~/.bash_profile или /etc/profile и помечены для экспорта, для того, чтобы в дальнейшем не было необходимости указывать их снова. Но есть несколько опций которые не могут быть экспортированы, и поэтому они должны быть заданы в ~/.bashrc и в вашем профиле последовательно. Эти опции настраиваются при помощи встроенной команды set:

```
$ set –x
```

Опция -x заставляет bash вывести на экран каждую команду, которую он собирается выполнить:

```
$ echo $FOO
$ echo foo
foo
```

Это может быть очень полезно для понимания непредвиденного поведения команд при использовании кавычек или похожих странностей. Чтобы выключить опцию -x, используйте set +x. Обратитесь к странице документации man за всеми опциями встроенной команды set.

Установка переменных с «set»

Команда set может также использоваться для задания значений переменных, но при этом указание самой этой команды не является обязательным. Команда в bash «set FOO=foo» делает то же самое, что и «FOO=foo». Сброс значения переменной осуществляется встроенной unset:

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

Unset vs. FOO=

Это не то же самое, что установка переменной пустым значением, хотя порой это сложно объяснить. Один из способов эту разницу заметить — вызвать команду set без параметров, чтобы вывести список всех текущих переменных:

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

Использование set без параметров похоже на использование встроенной команды export за исключением того, что set отображает все переменные, а не только обозначенные как внешние.

Экспортирование переменных для изменения поведения программ.

Часто поведение команд можно изменить установкой переменных окружения. Так же, как в случае новых сессий bash, запускаемые программы из вашей командной строки будут видеть только переменные окружения, помеченные на экспорт. Например, команда man проверяет переменную PAGER, чтобы выяснить какую программу использовать для постраничного

просмотра текста.

```
$ PAGER=less
$ export PAGER
$ man man
```

Когда переменная PAGER установлена в less, вы будете видеть сначала одну страницу, а нажатие пробела будет перемещать вас а следующую страницу. Если вы измените переменную PAGER в cat, то весь текст отобразится сразу, без остановок на страницах.

```
$ PAGER=cat
$ man man
```

Использование «env»

К сожалению, если вы позабудете установить PAGER обратно в less, программа man (как и некоторые другие программы) будет продолжать вывод весь запрошенный текст без остановок. Если вы хотели задать PAGER значение cat только на один раз, то могли бы воспользоваться командой env:

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

В этом примере переменная PAGER была использована со значением cat в программе man, но сама по себе переменная окружения PAGER осталась неизменной в сессии bash.

Файловые системы, разделы и блочные устройства

Введение в блочные устройства

В этом разделе мы будем рассматривать аспекты работы Linux с дисками, включая файловые системы, разделы и блочные устройства. Как только вы познакомились с преимуществами и недостатками дисков и файловых систем, мы с вами разберем процесс настройки разделов и файловых систем на Linux.

Вначале ознакомимся с «блочными устройствами». Наиболее известным блочным устройством, вероятно, будет первый диск IDE в системе Linux, который будет называться: /dev/hda

Если в вашей системе есть SCSI диски (или, что вероятнее, вы используете современным драйвер libATA — прим. ред.), то он будет называться: /dev/sda

Уровни абстрагирования

Блочные устройства представляют абстрактный интерфейс к диску. Пользовательские программы могут использовать эти блочные устройства для взаимодействия с диском, не беспокоясь о том, что у вас за диски: IDE, SCSI, или какие-то другие. Программы могут легко адресовать место на диске, как последовательность блоков по 512 байт с произвольным доступом.

Разделы

В Linux файловые системы (ФС) создаются при помощи специальной команды mkfs (или mke2fs, mkreiserfs, и др.), указывая в качестве аргумента конкретное блочное устройство.

Однако, хотя и возможно использовать блочные устройства, представляющие весь диск целиком, такие как /dev/hda или /dev/sda, для

единственной ФС, это редко применяется на практике. Вместо этого дисковые блочные устройства разделяются на более удобные блочные устройства меньшего размера, называемые разделами. Разделы создаются с помощью средства под названием fdisk, которое используется для создания и редактирования таблиц разделов, расположенных на каждом диске. Таблица разделов определяет, как именно разбито пространство на целом диске.

Введение в fdisk

Мы можем взглянуть на таблицу разделов диска запустив fdisk, указав в качестве аргумента блочное устройство представляющие диск целиком.

Примечание:

Альтернативные средства для доступа к таблице разделов: cfdisk, parted и partimage. Я рекомендую вам избегать использования cfdisk (несмотря на то, что может быть сказано в руководстве по fdisk) т. к. оно иногда неправильно рассчитывает геометрию диска.

```
# fdisk /dev/hda
# fdisk /dev/sda
```

Важно!

Не сохраняйте и не вносите каких-либо изменений в дисковую таблицу разделов, если один из них содержит файловую систему, используемую в настоящий момент или хранящую важные данные. Эти действия, скорее всего, приведут к потере данных на диске.

Внутри fdisk

После запуска fdisk, вас поприветствует приглашение, которое выглядит примерно так:

```
Command (m for help):
```

Введите p для отображения текущей таблицы разделов вашего диска:

```
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):
```

Данный диск сконфигурирован для размещения семи файловых систем Linux (каждая на соответсвующем разделе, помеченном как «Linux»), а также раздела подкачки (помечен как «Linux swap»).

Обзор блочных устройств и разделов

Обратите внимание на названия блочных устройств слева, соответвующих разделу, начиная с /dev/hda1 по /dev/hda9. В начале эры ПК, программы разметки позволяли делать максимум четыре раздела (называемых первичными). Этого было слишком мало, и для обхода этого ограничения был придуман расширенный раздел. Расширенный раздел очень похож на первичный, и засчитываются в лимит для 4-х первичных разделов. Но, расширенный раздел может содержать любое количество т. н. логических разделов внутри себя, эффективно обходя ограничение на четыре раздела.

Разметка диска

Все разделы от hda5 и далее — это логические разделы. Номера с hda1 по hda4 зарезервированы для первичных или расширенного разделов.

В нашем примере, разделы с hda1 по hda3 являются первичными разделами. hda4 это расширенный раздел, который содержит логические разделы от hda5 до hda9. Вы не будет использовать /dev/hda4 для хранения ФС — он просто действует как контейнер для разделов hda5 — hda9.

Типы разделов

Кроме того, обратите внимание, что каждый раздел имеет "Id", также называемый типом раздела. Всякий раз, когда вы создаете новый раздел, вы должны убедиться, что тип раздела установлен правильно. Значение 83 является верным для разделов ФС Linux, а 82 — для разделов подчкачки. Для установки значения типа используется опция “t” в fdisk. Ядро Linux использует настройки типа раздела для автоопределения на диске во время загрузки устройств файловых систем и подкачки.

Использование fdisk для создания разделов

Теперь, когда вы имеете представление о дисковых разделах в Linux, пришло время, чтобы начать процесс создания разделов на диске и ФС для установки Linux. Мы настроим разделы на диске, а затем создадим файловые системы на них. На этом этапе мы полностью очистим диск от данных, и будем его использовать для установки новой копии Linux системы.

Важно!

Для выполнения этих действий, у вас должен быть жесткий диск, который не содержит никакой важной информации, так как, на этом этапе, данные на диске будут удалены. Если это всё для вас в новинку, вы можете только прочитать эти шаги, или воспользоваться загрузочным диском с Linux на тестовой системе (например в виртуальной машине — прим. ред.), так что

данные не будут в опасности.

Как будет выглядеть диск после разбивки

После того, как мы пройдем процесс создания разделов на вашем диске, ваша таблица разделов будет выглядеть 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):
```

Комментарий к примеру

В новой, предложенной нами конфигурации, у нас есть три раздела. Первым является маленький раздел (/dev/hda1) в начале диска, называемый загрузочным разделом. Цель загрузочного раздела – это хранение всех важных данных, связанных с загрузкой – загрузчик GRUB, а также ваше Linux ядро (ядра). Загрузочный раздел обеспечивает нам безопасное место для хранения любой информации, связанной с загрузкой. При нормальной работе загрузочный раздел должен оставаться отмонтированным для безопасности. Если у вас SCSI диски (или используется современная библиотека libATA — прим. ред.), загрузочный раздел, скорее всего, будет называться /dev/sda1.

Рекомендовалось держать загрузочный раздел (содержащий всё необходимое для загрузки) в начале диска. Это не обязательно, так как берет свои истоки из прошлого, когда загрузчик LILO не мог загружать ядро с файловых систем, которые располагались за 1024 цилиндром диска.

Второй раздел (/dev/hda2) используется для подкачки. Ядро

использует дисковое пространство подкачки как виртуальную память, когда места в ОЗУ мало. Размер раздела сравнительно не очень большой, как правило около 512 МБ. Для систем SCSI (а также с новой libATA — прим. ред.) этот раздел будет называться /dev/sda2.

Третий раздел (/dev/hda3) большого размера и занимает весь остальной диск. Этот раздел будет нашим корневым разделом, и будет служить для хранения главной файловой системы Linux. Для дисков SCSI (или новой libATA — прим. ред.) этот раздел будет называться /dev/sda3.

Начало работы

Теперь, чтобы создать разделы по примеру выше, введите fdisk /dev/hda или fdisk /dev/sda в зависимости от того, используете ли вы диски IDE или SCSI (или современную libATA — прим. ред.) соответственно. Затем введите “p” для просмотра текущей таблицы разделов. Есть ли что-то на диске, что требуется сохранить? Если да, остановитесь сейчас. Если вы продолжите, вся существующая информация на диске будет уничтожена.

Важно!
Нижеследующие инструкции уничтожат все существующие данные на диске! Если на диске есть какие-либо данные, убедитесь, что информация не является для вас критически важной. Также убедитесь что вы выбрали правильный диск, чтобы ошибочно не стереть данные с другого диска.

Удаление существующих разделов

Теперь самое время удалить все существующие разделы. Чтобы это сделать, введите “d” и нажмите Enter. Вам будет предложено выбрать номер раздела, который будет удален. Чтобы удалить существующий раздел /dev/hda1 вы должны ввести:

```
Command (m for help): d
```

```
Partition number (1-4): 1
```

Раздел будет запланирован для удаления. Он больше не будет отображаться, если вы введете “p”, но он не будет удален, пока вы не сохраните свои изменения. Если вы ошиблись и хотите отменить действия, введите “q”, и нажмите Enter, и ваш раздел не будет удален.

Теперь, предполагая, что вы в самом деле хотите удалить все разделы в вашей системе, наберите “p”, чтобы вывести еще раз список разделов, а затем введите “d” и номер раздела для удаления. В итоге вы получите пустую таблицу разделов:

```
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

Теперь, когда таблица разделов в памяти пуста, мы готовы для создания раздела загрузки. Чтобы это сделать, введите “n” для создания нового раздела, затем введите “p” чтобы сообщить fdisk, что вы хотите первичный раздел. После чего, введите “1” для создания первого первичного раздела. На вопрос о первом цилиндре нажмите Enter. На вопрос о последнем цилиндре введите “+100M” чтобы создать раздел размером 100 МБ. Вывод проделанных действий:

```
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 введите “p”, вы должны увидеть нижеследующую таблицу разделов:

```
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

Теперь, давайте создадим раздел подкачки. Чтобы это сделать введите “n” для создания нового раздела, затем “p” чтобы сообщить fdisk что вы хотите создать первичный раздел. Затем введите “2” для создания второго первичного раздела, /dev/hda2 в нашем примере. Затем будет предложено ввести номер первого цилиндра, нажмите Enter, когда будет предложено ввести номер последнего цилиндра, введите “+512M” для создания раздела подкачки, размером 512 МБ. После того, как вы сделаете это, введите “t” для установки типа раздела, и затем введите “82” для установки типа ”Linux swap”. После завершения этих шагов, введите “p” для просмотра таблицы разделов, она должна быть похожей на эту:

```
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 загрузочным

В завершении мы должны установить флаг «загрузочный» на наш загрузочный раздел и записать изменения на диск. Для отметки раздела /dev/hda1 как «загрузочного» раздела, введите в меню “a” и затем “1” как номер раздела. Если вы введете сейчас “p”, вы увидите что /dev/hda1 содержит символ “*” в столбце Boot. Теперь давайте запишем наши изменения на диск. Для этого введите “w” и затем Enter. Ваши разделы диска сейчас правильно сконфигурированы для установки Linux.

Замечание:
Если fdisk запрашивает перезагрузку, пожалуйста, сделайте это для того, чтобы ваша система определила новую настройку разделов.

Расширенные и логические разделы

В приведенном выше примере мы создали один первичный раздел который будет содержать ФС для хранения всех наших данных. Это означает что после установки Linux, главная файловая система будет смонтирована в “/” и будет содержать дерево директорий которое содержит все наши файлы.

Хотя это общий подход, есть и другой подход, с которым вы тоже должны быть знакомы. Этот подход использует несколько разделов, как место для нескольких ФС, и которые вместе образовывают дерево файловой системы. Например, довольно распространено помещать /home и /var в отдельные ФС.

Мы могли бы создать hda2 как расширенный, а не как первичный раздел. Затем мы бы создали логические разделы hda5, hda6, hda7 (технически они будут располагаться внутри hda2), которые бы содержали ФС для /, /home и /var соответственно.

Вы можете узнать больше про эти типы мультисистемной

конфигурации, изучив ресурсы, приведенные ниже.

Создание файловых систем

Теперь, когда разделы уже созданы, пришло время установить ФС на загрузочные и корневые разделы так, чтобы они могли использоваться для хранения данных. Мы также настроим раздел подкачки для обслуживания данных подкачки.

Линукс поддерживает различные типы ФС; каждый из них имеет свои достоинства и недостатки и свои характеристики. Мы рассмотрим создание файловых систем ext2, ext3, XFS, JFS и ReiserFS в этом руководстве. Перед созданием ФС на нашем примере, мы кратко рассмотрим различные файловые системы доступные в Linux.

Файловая система ext2

ext2 является проверенной годами файловой системой Linux, но она не обладает средствами журналирования метаданных, что означает, что время на проверку файловой системы во время запуска может быть довольно большим. В настоящее время существует широкий выбор журналируемых файловых систем, которые могут быть проверены на целостность очень быстро, и потому предпочтительны, нежели их не журналируемые аналоги. Журналируемая ФС позволяет избежать долгих задержек при старте системы, когда целостность вашей ФС нарушена (например, в случае сбоя электроснабжения — прим. ред.).

Файловая система ext3

ext3 – журналируемая версия файловой системы ext2, которая обеспечивает журналирование метаданных для быстрого восстановления, а также другие режимы журналирования, такие как полное журналирование всех данных и упорядоченное журналирование. ext3 – очень хорошая и надежная ФС. Она предлагает достойную производительность в большинстве случаев. Поскольку она мало использует «деревья» в своем внутреннем устройстве, она плохо масштабируется, это означает, что этот тип ФС не лучший выбор для очень больших файловых систем, или в условиях, когда вы будете обрабатывать большие файлы или большое количество файлов в одном каталоге. Но при использовании её в условиях, под которые она проектировалась, ext3 прекрасная файловая система.

Одна из приятных особенностей ext3 – это то, что существующие системы ext2 могу быть обновлены «на месте» до ext3 довольно просто. Это позволяет плавно обновлять существующие системы Linux, которые уже используют ext2.

Файловая система ReiserFS

ReiserFS – это файловая система, основанная на B-дереве, которая имеет очень хорошую производительность и значительно превосходит ext2 и ext3 при работе с небольшими файлами (файлы менее 4 кБ), часто в 10-15 раз. А также ReiserFS отлично масштабируется и имеет журналирование метаданных. Начиная с ядра версии 2.4.18 и выше, ReiserFS является стабильной и рекомендуется, как в качестве ФС общего назначения, так и в крайних случаях, таких как создание больших файловых систем, использование для множества маленьких файлов, для огромных файлов, а также для каталогов с десятками тысяч файлов. Мы рекомендуем ФС ReiserFS для использования по умолчанию для всех не загрузочных разделов.

Файловая система XFS

XFS – это файловая система с журналированием метаданных. Она обладает конкретным набором возможностей и оптимизирована для масштабирования. Мы рекомендуем использовать эту файловую систему исключительно на Linux системах с высококлассными SCSI и/или Fibre Channel накопителями и источниками бесперебойного питания. Поскольку XFS агрессивно кэширует данные в ОЗУ, неподходяще спроектированная программа (т. е. та, которая не принимает должной предосторожности при записи на диск (таких совсем немного)) может потерять приличную порцию данных, если система неожиданно даст сбой.

Файловая система JFS

JFS является созданной в IBM высокопроизводительной журналируемой файловой системой. В последнее время она стала предустановленной, и мы бы хотели накопить больший опыт её использования, прежде чем выявлять сильные и слабые стороны этой файловой системы.

Рекомендации к файловым системам

Если вы ищете надежную журналируемую файловую систему, используйте ext3. Если вы ищете хорошую файловую систему общего назначения с высокой производительностью и поддержкой журналирования – используйте ReiserFS; ext3 и ReiserFS проверенные, усовершенствованные и рекомендуемые для общего назначения системы.

Основываясь на нашем примере выше, мы будем использовать следующие команды чтобы инициализировать все наши разделы для использования:

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

Мы выбираем ext3 для нашего загрузочного раздела /dev/hda1, так как

это надежная журналируемая файловая система поддерживается всеми основными загрузчиками. Мы использовали mkswap для раздела подкачки /dev/hda2 – выбор тут очевиден. И для нашей главной корневой файловой системе на /dev/hda3 выберем ReiserFS, так как эта стабильная ФС с журналированием предлагающая отличную производительность. Теперь будем инициализировать разделы нашего диска.

Создание раздела подкачки

mkswap – команда для инициализации раздела подкачки:

```
# mkswap /dev/hda2
```

В отличии от обычных файловых систем, разделы подкачки не монтируются. Вместо этого, их активируют используя команду swapon:

```
# swapon /dev/hdc6
```

Стартовые скрипты вашей Linux системы позаботятся об автоматической активации разделов подкачки. Таким образом, команда swapon, как правило требуется только тогда, когда нужно немедленно добавить раздел подкачки, который вы только что создали. Для просмотра какие разделы подкачки сейчас используются, наберите cat /proc/swaps.

Создание файловых систем ext2, ext3, ReiserFS

Для создание файловой системы ext2 можно использовать команду mke2fs:

```
# mke2fs /dev/hda1
```

Если вы хотите использовать ext3, можно использовать команду mke2fs -j:

```
# mke2fs -j /dev/hda3
```

Для создание файловой системы ReiserFS используется команда mkreiserfs:

```
# mkreiserfs /dev/hda3
```

Создание файловых систем XFS и JFS

Для создания файловой системы XFS используется команда mkfs.xfs:

```
# mkfs.xfs /dev/hda3
```

Примечание:
Вы можете добавить к команде mkfs.xfs дополнительные флаги: “-d agcount=3 -l size=32m”. Флаг “-d agcount=3” снизит количество групп распределения. XFS будет настаивать на использовании по крайней мере одной группы распределения на 4 ГБ в разделе, так, например, если у вас есть раздел на 20ГБ, вам необходимо минимальное значение “agcount=5”. Флаг “-l size=32m” увеличивает размер журнала до 32 МБ, увеличивая производительность.

Прим. ред: Информация в данном руководстве несколько устарела. На самом деле, ещё, по меньшей мере, более 6 лет назад, максимальный размер группы распредления (allocation group) в XFS увеличен до терабайта.

Для создания файловой системы JFS, используется команда mkfs.jfs:

```
# mkfs.jfs /dev/hda3
```

Монтирование файловых систем

После того как файловая система создана, мы можем её примонтировать, используя команду mount:

```
# mount /dev/hda3 /mnt
```

Чтобы смонтировать файловую систему, в качестве первого аргумента

необходимо указать раздел блочного устройства, и «точку монтирования» – в качестве второго. Новая файловая система будет «привита» в точке монтирования. Это также приводит к эффекту скрытия любых файлов которые находятся в директории /mnt в родительской файловой системе. Позже когда файловая система отмонтирована, эти файлы снова появятся. После выполнения команды монтирования, любые файлы созданные или скопированные внутри /mnt будут находится на новой файловой системе ReiserFS, которую вы смонтировали.

Скажем, мы хотим смонтировать наш загрузочный раздел внутрь /mnt. Мы можем это сделать, выполнив следующие шаги:

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

Теперь, наш загрузочная файловая система доступна внутри /mnt/boot. Если мы создадим файлы внутри /mnt/boot, то они будут находится на нашем ext3 разделе, который физически расположен на /dev/hda1. Если мы создадим файлы внутри /mnt, но не внутри /mnt/boot, то они будут находится на нашей ReiserFS системе которая находится на /dev/hda3. И если мы создадим файлы за пределами /mnt, они не будут хранится на нашей файловой системе (загрузочном разделе), а на файловой системе текущей Linux системы или загрузочном диске.

Чтобы просмотреть какие файловые системы сейчас смонтированы, введите mount без аргументов. В выводе команды mount мы видим одну из наших запущенных Linux систем, которая содержит разделы, настроенные аналогично нашему примеру:

```
/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)
```

Вы также можете просмотреть аналогичную информацию введя cat /proc/mounts. Корневая файловая система /dev/hda3 будет смонтирована автоматически ядром при загрузке и получит символьное имя /dev/hda3. На нашей системе и /dev/hda3, и /dev/root указывают на одно и то же блочное устройство, лежащее в основе, используя символическую ссылку:

```
# 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
```

Еще немного о монтировании

Итак, что это за файл /dev/ide/host0...? У систем, как моя, которые используют devfs файловую систему управления устройствами для /dev, есть более длинные официальные названия для разделов и дисковых блоковых устройств, которые Linux использовал в прошлом. Для примера, /dev/ide/host0/bus1/target0/lun0/part7 это официальное название для /dev/hdc7, а сам /dev/hdc7 – это всего лишь символьная ссылка (симлинк) указывающая на блочное устройство. Вы можете определить используется ли в вашей системе devfs, проверив существование файла /dev/.devfsd; если существует – значит devfs активна.

Когда используется команда mount для подключения файловых систем, она пытается автоматически определить тип файловой системы. Иногда это не работает и вам нужно указать это вручную, используя опцию -t, как в примере ниже:

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

или

```
# mount /dev/hda3 /mnt -t reiserfs
```

Опции монтирования

Также возможно настраивать атрибуты для монтируемых ФС с помощью опций монтирования. К примеру, вы можете смонтировать файловую систему в режиме «только чтение» используя опцию “ro”:

```
# mount /dev/hdc6 /mnt -o ro
```

С /dev/hdc6 смонтированной только для чтения, никакие файлы в /mnt не смогу быть изменены – только прочитаны. Если ваша ФС уже смонтирована для «чтения/записи» и вы хотите переключить её в режим «только чтение», вы можете использовать опцию remount избежав отключения и подключения ФС снова:

```
# mount /mnt -o remount,ro
```

Заметьте, что нам не нужно было указывать конкретное блочное устройство, т. к. ФС уже смонтирована и mount знает что /mnt ассоциирована с /dev/hdc6. Чтобы снова можно было производить запись, мы должны перемонтировать её в режиме «чтение/запись»:

```
# mount /mnt -o remount,rw
```

Заметьте, что эти команды перемонтирования не будут успешно завершены, если в /mnt имеются открытые каким-либо процессом файлы или директории. Для ознакомления со всеми опциями команды mount в Linux, введите man mount.

Знакомство с fstab

До сих пор мы видели как разбивать диск и монтировать разделы вручную с загрузочного диска. Но, как только мы установим систему Linux, как мы будем настраивать её, чтобы она монтировала нужные ФС в нужное время? Например скажем, что мы установили Gentoo Linux на нашей конфигурации ФС. Каким образом наша система знает, как найти корневую

директорию на /dev/hda3? И если какие-либо другие ФС – как, например, раздел подкачки – нужно монтировать при загрузке, как она узнает которые из них?

Итак, ядру Linux сообщается загрузчиком, какая используется корневая ФС, и мы рассмотрим загрузчики Linux позже в этом руководстве. Но для всего остального, ваша Linux система содержит файл, называемый /etc/fstab который сообщает ядру о доступных для монтирования файловых системах. Давайте взглянем на него.

Образец fstab

Давайте взглянем на образец файла /etc/fstab:

```
<fs> <mountpoint> <type>    <opts>            <dump/pass>

/dev/hda1 /boot     ext3 noauto,noatime 1 1
/dev/hda3 /    reiserfs  noatime 0 0
/dev/hda2 none swap sw   0 0
/dev/cdrom     /mnt/cdrom     iso9660   noauto,ro,user 0 0
# /proc should always be enabled
proc /proc     proc defaults  0 0
```

Каждая не комментированная строка выше в /etc/fstab определяет раздел блочного устройства, точку монтирования, тип ФС и её опции, используемые при монтировании, а также два числовых поля. Первое числовое поле используется, чтобы сообщить какие файловые системы требуют резервного копирования с помощью команды dump. Конечно, если вы не планируете использовать dump в вашей системе, то вы можете благополучно игнорировать это поле. Последнее поле используется программой проверки целостности ФС fsck, и устанавливает порядок в соответсвии с которым должны проверяться ваши файловые системы во время загрузки. Мы коснемся fsck еще раз позднее.

Посмотрите на строку /dev/hda1; вы видите, что /dev/hda1 это ФС ext3, которая должна быть смонтирована в точку /boot. Теперь взгляните на

опции монтирования в столбце opts. Опция “noauto” сообщает системе не монтировать /dev/hda1 автоматически при загрузке; без этой опции /dev/hda1 будет автоматически смонтирована в /boot во загрузки системы.

Также обратите внимание на опцию “noatime”, которая отключает запись atime (время последнего доступа) информации на диск. Эта информация в основном не требуется, и выключение всех обновлений atime даст положительный эффект на производительности системы.

Теперь взглянем на строку /proc и заметим опцию “defaults”. Используйте “defaults”, если вы хотите чтобы ФС была смонтирована со стандартными опциями. Т. к. /etc/fstab содержит множество полей, мы не можем просто оставить поле опций пустым.

Также примите во внимание строку /dev/hda2 в /etc/fstab. Эта строка определяет /dev/hda2 как устройство подкачки. Поскольку устройства подкачки не монтируются как ФС, им не назначается и точка монтирования. Благодаря этой записи в /etc/fstab, наше устройство подкачки /dev/hda2 будет автоматически включаться, когда запускается система.

С записью /dev/cdrom, в файле /etc/fstab монтирование CD-ROM будет легче. Вместо того чтобы вводить:

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

Мы теперь можем ввести:

```
# mount /dev/cdrom
```

На самом деле, /etc/fstab позволяет нам получить приемущество от использования опции “user”. Эта опция монтирования сообщают системе, что данная конкретная ФС может монтироваться любым пользователем. Это очень удобно для съемных носителей, таких как CD-ROM. Без этой опции

монтирования, только пользователь root смог бы использовать CD-ROM.

Размонтирование файловых систем

Как правило, все подключенные ФС размонтируются системой автоматически при выключении или перезагрузке. Когда ФС отмонтирована, то все закешированные в памяти данные ФС сброшены на диск.

Тем не менее, также возможно размонтировать ФС вручную. Перед тем, как ФС будет отключена, вы должны убедиться, что ней нет открытых файлов какими-либо процессами. Затем, используйте команду umount, определив как аргумент имя устройства или точку монтирования:

```
# umount /mnt
```
или
```
# umount /dev/hda3
```

После размонтирования, любые файлы в /mnt, которые были сокрыты прежде подключенной ФС, станут доступны.

Введение в fsck

Если в вашей системе произойдет сбой или она заблокируется по какой-то причине, система не будет иметь возможности корректно отмонтировать ваши ФС. Когда это случиться, они окажутся в непредсказуемом состоянии и их целостность может быть нарушена. Когда система перезагрузится, программа fsck определит, что ФС не были корректно размонтированы и захочет произвести проверку целостности ФС, перечисленных в /etc/fstab.

Важно!
Для ФС, которая будет проверена с помощью fsck, должно быть не нулевое значение в поле “pass” (последнее поле) в /etc/fstab. Обычно, для корневой ФС значение устанавливается в “1”, указывая на то, что она должна быть

проверена в первую очередь. У всех других ФС, которые должны быть проверены во время загрузки, значение поля “pass” должно быть “2” или выше. Для некоторых журналируемых ФС, таких как ReiserFS, безопасно иметь значение “0”, поскольку сам программный код журналирования (а не внешний fsck) заботится о сохранении целостности ФС.

Иногда, мы можете обнаружить, что после перезагрузки fsck не может полностью восстановить частично поврежденную ФС. В таких случаях, всё, что вам нужно сделать, это перевести систему в однопользовательский режим и запустить fsck вручную, передав в качестве аргумента блочное устройство раздела. Поскольку fsck будет производить восстановление его ФС, то может спросить вас об исправлении конкретных дефектов ФС. В основном, вам стоит отвечать “y” (да) на все эти вопросы, разрешая fsck делать свое дело.

Проблемы с fsck

Одна из проблем со сканированием fsck состоит в том, что оно может занять длительное время до завершения, поскольку совокупность метаданных файловой системы (внутренняя структура данных) должна быть просканирована, чтобы убедиться в их целостности. Для особо больших ФС, не редко, для полного завершения fsck требуется более часа.

Для того, чтобы решить эту проблему, были спроектированы новые типы ФС, называемые журналируемые файловые системы. Журналируемые ФС пишут на диск журнал последних изменений метаданных файловой системы. В случае сбоя, драйвер ФС проверяет журнал. Так как журнал содержит точный отчет о последних изменениях на диске, то только эти части метаданных ФС требуют проверки на ошибки. Благодаря этому важному отличию, проверка журналируемой системы на целостность обычно занимает только считанные секунды, независимо от размера ФС. Поэтому журналируемые ФС завоёвывают популярность в сообществе Linux.