Впервые опубликовано на developerWorks 15.11.2005

Подготовка к экзамену LPI 101: GNU и UNIX команды

Junior Level Administration (LPIC-1) тема 103

Использование командной строки

В этом разделе описываются следующие темы:

- Взаимодействие с командными интерпретаторами и командами
- Команды и последовательности команд
- Определение, использование и экспорт переменных окружения
- История команд и средства редактирования
- Запуск команд, находящихся в переменной окружения PATH так и вне нее
- Использование подстановки команд
- Применение команд рекурсивно к дереву каталогов
- Использование man-страниц (помощи) для поиска информации о командах

Данный раздел дает описание некоторых основных возможностей командного интерпретатора bash. Особый акцент делается на возможностях, необходимых для сертификации. Командный интерпретатор это богатая среда, и мы приветствуем ее дальнейшее самостоятельное изучение. По командным интерпретаторам UNIX и Linux написано много книг и bash в частности.

Командный интерпретатор bash

Интерпретатор bash один из нескольких интерпретаторов, доступных в Linux. Также он называется Bourne-again shell, в честь Стивена Борна, создателя ранней версии интерпретатора (/bin/sh).Bash по существу совместим с sh, но представляет много улучшений, как в функциональном плане, так и возможностям программирования. Он включает возможности интерпретаторов Korn (ksh) и C (csh), и разрабатывается как POSIX-совместимый интерпретатор.

Прежде, чем мы начнем изучать bash, напомним, что интерпретатор -- это программа, которая принимает и исполняет команды. Он также поддерживает возможности программирования, позволяя составлять сложные конструкции из обычных команд. Эти сложные конструкции или сценарии можно сохранить в файлы, которые в свою очередь сами являются новыми командами. Более того, множество команд на типичной Linux системе реализованы как сценарии командного интерпретатора.

Интерпретаторы содержат встроенные команды, такие как cd, break и exec. Другие команды являются внешними.

Интерпретаторы также используют три стандартных потока ввода/вывода:

- stdin это стандартный поток ввода, который обеспечивает ввод для команд.
- stdout это стандартный поток вывода, который обеспечивает отображение результатов выполнения команды в окне терминала.
- stderr это стандартный поток ошибок, который отображает ошибки, возникающие при работе команд.

Потоки ввода обеспечивают ввод для программ, обычно он связан с клавиатурой терминала. Выходные потоки печатают текстовые символы, обычно на терминал. Терминал изначально был ASCII печатной машинкой или видеотерминалом, но сейчас он часто представляет собой окно на графическом рабочем столе. Более подробно о том, как перенаправлять стандартные потоки ввода/вывода, можно посмотреть дальше в этом руководстве в разделеПотоки, программные каналы и перенаправление. Остальная часть этого раздела сосредоточится на перенаправлении на высоком уровне.

Далее мы будем предполагать, что вы знаете, как попасть в командную строку. Если нет, то статья developerWorks "Основные задачи начинающих разработчиков Linux поможет вам разобраться с этой и другими задачами.

Если вы используете Linux систему без графического интерфейса или же вы открыли окно терминала в графическом режиме, то увидите приглашение для ввода команд как в Листинге 1.

Листинг 1. Примеры типичных пользовательских приглашений

```
[db2inst1@echidna db2inst1]$
ian@lyrebird:~>
$
```

Если вы зайдете как пользователь root (или суперпользователь), то ваше приглашение может выгладить, как показано в Листинге 2.

Листинг 2. Примеры приглашений для пользователя root или суперпользователя

```
[root@echidna root]#
lyrebird:~ #
#
```

Пользователь root имеет значительную власть, поэтому пользуйтесь им с осторожностью. Если у вас привилегии пользователя root, то большинство приглашений начинаются со знака решетки (#). Приглашение для обычного пользователя, как правило, начинается с другого символа, обычно это знак доллара ($). Ваше приглашение может отличаться от того, что написано в примерах данного руководства. Ваше приглашение может включать ваше пользовательское имя, имя машины, текущий каталог, дату или время, когда было напечатано приглашение, и так далее.

Некоторые соглашения этого руководства

Руководства developerWorks по экзаменам LPI 101 и 102 включают код примеров из реальных Linux систем, с использованием приглашений по умолчанию для этих систем. В нашем случае приглашение пользователя root начинается с #, так что вы можете отличить его от приглашений обычных пользователей, которые начинаются со знака $. Это соглашение совпадает с тем, которое используется в книгах по

данному предмету. Внимательно смотрите приглашение командного интерпретатора в каждом примере.

Команды и последовательности

Вы находитесь в командном интерпретаторе, посмотрим, что вы можете теперь сделать. Основная функция командных интерпретаторов состоит в том, что он исполняет ваши команды, посредством которых вы взаимодействуете с Linux системой. В системах Linux (и UNIX) команды состоят из имени команды, опций и параметров. У некоторых команд нет ни опций, ни параметров, у других есть опции, но нет параметров, в то время как у третьих нет ни опций, ни параметров.

Если строка содержит символ #, то все последующие символы в ней игнорируются. Таким образом, символ # может означать как начало комментария, так и приглашение пользователя. Дальнейшая интерпретация будет очевидна из контекста.

Команда Echo

Команда echo выводит на терминал список своих аргументов как показано в Листинге 3.

Листинг 3. Примеры команды echo

```
[ian@echidna ian]$ echo Слово
Слово
[ian@echidna ian]$ echo И предложение
И предложение
[ian@echidna ian]$ echo Куда   подевались    пробелы?
Куда подевались пробелы?
[ian@echidna ian]$ echo "А     вот   и   пробелы." # и комментарий
А     вот   и   пробелы.
```

В третьем примере Листинга 3 все промежутки между словами на выходе команды стали одного размера в один пробел. Чтобы этого избежать вам потребуется заключить строку в кавычки, используя или двойные кавычки (") или одинарные ('). Bash использует символы разделители, как пробелы, символы табуляции и символы новой строки для разбиения входной строки на токены, которые затем передаются вашей команде. Заключение строки в кавычки подавляет ее разделение и таким образом она является единым токеном. В приведенном выше примере каждый токен после имени команды является параметром, таким образом, у нас получается соответственно 1, 2, 4 и 1 параметр.

У команды echo есть несколько опций. Обычно echo добавляет после своего вывода символ новой строки. Используйте опцию -n чтобы она не добавляла символ новой строки. Используйте опцию -e, чтобы команда интерпретировала escape-последовательности. Некоторые из них представлены в Таблице 3.

Таблица 3. Echo escape-последовательности

Escape последовательность	Значение
\a	Звонок
\b	Забой последнего символа
\c	Не добавлять символ новой строки (тоже самое, что и опция -n)
\f	Перевод страницы (очищает экран на видео дисплее)
\n	Новая строка
\r	Перевод каретки
\t	Горизонтальная табуляция

Escape-последовательности и перенос строки

Существует небольшая проблема при использовании обратного слеша в bash. Когда символ обратного слеша (\) не заключен в кавычки, то он сам служит escape-последовательностью для bash, предохраняя значение следующего символа. Это необходимо для особых метасимволов, которые мы рассмотрим чуть позже. Существует одно исключение из этого правила: обратный слеш, за которым следует перевод строки, заставляет bash проглотить оба символа и считать последовательность как запрос на продолжение строки. Это может быть полезным при разбиении длинных строк, особенно применительно к сценариям.

Чтобы последовательности, описанные выше, правильно обрабатывались командой echo или одной из многих других команд, которые используют похожие escape символы управления, вы должны заключить escape последовательности в кавычки или же включить их в строку, заключенную в кавычки, либо использовать еще один обратный слеш для верной интерпретации символов. Листинг 4 содержит примеры различных вариантов использования \.

Листинг 4. Примеры использования echo

```
[ian@echidna ian]$ echo -n Нет новой строки
Нет новой строки[ian@echidna ian]$ echo -e "Нет новой строки\c"
Нет новой строки[ian@echidna ian]$ echo "Строка в которой нажали
> клавишу Enter"
Строка в которой нажали
клавишу Enter
[ian@echidna ian]$ echo -e "Строка с escape символом\nновой строки"
Строка с escape символом
новой строки
[ian@echidna ian]$ echo "Строка с escape символом\nновой строки, но без
опции -e"
Строка с escape символом\nновой строки, но без опции -e
[ian@echidna ian]$ echo -e Метасимволы с двойным\\n\\tобратным слешем
Метасимволы с двойным
      обратным слешем
```

```
[ian@echidna ian]$ echo Обратный слеш \
> за которым следует Enter \
> служит как запрос на продолжение строки.
Обратный слеш за которым следует Enter служит как запрос на продолжение
строки.
```

Заметим, что bash отображает специальное приглашение (>), когда вы набрали строку с незавершенными кавычками. Ваша входная строка переносится на вторую строку и в неё включает символ новой строки.

Метасимволы Bash и операторы управления

Bash включает несколько символов, которые, будучи не заключенными в кавычки, также служат для разделения входной строки на слова. Кроме пробела такими символами являются '|', '&', ';', '(', ')', '<', и '>'. Некоторые из этих символов мы обсудим более подробно в других разделах этого руководства. А сейчас заметим, что если вы хотите включить метасимвол как часть вашего текста, то он должен быть заключен в кавычки или же ему должен предшествовать обратный слеш (\) как в Листинге 4.

Новая строка и соответствующие метасимволы или пары метасимволов также служат как операторы управления. Такими символами являются '||', '&&', '&', ';', ';;', '|" '(', и ')'. Некоторые из этих операторов управления позволяют вам создавать последовательности или списки команд.

Простейшая последовательность команд состоит из двух команд, разделенных точкой с запятой (;). Каждая следующая команда исполняется после предыдущей. В любой среде программирования команды возвращают код, свидетельствующий о нормальном или неудачном завершении программы; команды Linux обычно возвращают 0 в случае успешного завершения и ненулевое значение в случае неуспеха. Вы можете осуществлять обработку по условию, используя управляющие операторы && и ||. Если вы разделите две команды управляющим оператором &&, то вторая команда будет выполняться только в том случае, если первая возвратила на выходе ноль. Если вы разделили команды с помощью ||, то вторая команда будет исполняться, только если первая возвратила ненулевое значение. Листинг 5 содержит некоторые последовательности команд с использованием команды echo. Эти примеры не очень интересны, так как echo возвращает 0, но мы рассмотрим больше примеров, когда научимся использовать большее число команд.

Листинг 5. Последовательности команд

```
[ian@echidna ian]$ echo line 1;echo line 2; echo line 3
line 1
line 2
line 3
[ian@echidna ian]$ echo line 1&&echo line 2&&echo line 3
line 1
line 2
line 3
[ian@echidna ian]$ echo line 1||echo line 2; echo line 3
line 1
line 3
```

Выход

Вы можете выйти из командного интерпретатора с помощью команды exit. Дополнительно в качестве параметра вы можете задать код выхода. Если вы работаете с командным интерпретатором в терминальном окне в графическом режиме, то в этом случае оно просто закроется. Аналогично, если вы подсоединены к удаленной системе с помощью ssh или telnet (например), то соединение завершится. В интерпретаторе bash вы также можете нажать клавишу Ctrl и d для выхода.

Давайте рассмотрим еще один оператор управления. Если вы заключите команду или список команд в круглые скобки, то команда или последовательность команд будет выполняться в своей копии командного интерпретатора, таким образом, команда exit выходит из копии командного интерпретатора, а не из того интерпретатора, в котором вы работаете в данный момент. Листинг 6 содержит простые примеры совместно с использованием && и ||.

Листинг 6. Командные интерпретаторы и последовательности команд

```
[ian@echidna ian]$ (echo В копии интерпретатора; exit 0) && echo OK ||
echo Bad exit
В копии интерпретатора
OK
[ian@echidna ian]$ (echo В копии интерпретатора; exit 4) && echo OK ||
echo Bad exit
В копии интерпретатора
Bad exit
```

Переменные окружения

При работе в bash, вас окружает совокупность параметров, составляющих вашу среду, например, формат вашего приглашения, имя домашнего каталога, ваш рабочий каталог, название вашего интерпретатора, файлы, которые вы открыли, определенные вами функции и так далее. Ваша среда включает множество переменных которые можете устанавливать как вы, так и bash. Bash также позволяет вам создавать переменные оболочки, которые вы можете экспортироватьв свою среду для использования другими процессами, запущенными в интерпретаторе или другими интерпретаторами, которые вы можете запустить из текущего интерпретатора.

Как у переменных окружения, так и у переменных оболочки есть имя. Ссылаться на значение переменной можно, поставив перед именем переменной знак '$'. Некоторые наиболее общие переменные среды bash приведены в Таблице 4.

Таблица 4.Некоторые наиболее общие переменные среды bash

Имя	Значение
USER	Имя зашедшего в систему пользователя
UID	Цифровой идентификатор зашедшего в систему пользователя
HOME	Домашний каталог пользователя
PWD	Текущий рабочий каталог
SHELL	Имя командного интерпретатора

Имя	Значение
$	Идентификатор процесса (или PID) bash (или другого) процесса
PPID	Идентификатор процесса, который породил данный процесс (то есть идентификатор родительского процесса)
?	Код выхода последней команды

На Листинге 7 можно видеть некоторые переменные bash.

Листинг 7. Переменные среды и shell

```
[ian@echidna ian]$ echo $USER $UID
ian 500
[ian@echidna ian]$ echo $SHELL $HOME $PWD
/bin/bash /home/ian /home/ian
[ian@echidna ian]$ (exit 0);echo $?;(exit 4);echo $?
0
4
[ian@echidna ian]$ echo $$ $PPID
30576 30575
```

Вы можете создать или установить переменную оболочки, набрав сразу за именем переменной знак равно (=). Переменные чувствительны к регистру, таким образом, var1 и VAR1 -- это две разные переменные. По соглашению переменные, особенно экспортируемые переменные, пишутся в верхнем регистре, но это не обязательное требование. Формально, $$ и $? являются параметрами оболочки, а не переменными. Вы можете на них ссылаться, но не присваивать значения.

Когда вы создаете переменную оболочки, то часто захотите экспортировать ее в среду так, чтобы она стала доступна другим процессам, которые вы запускаете из интерпретатора. Переменные, которые вы экспортируете не доступны родительским интерпретаторам. Для экспортирования переменной используется команда export. Для удобства вы можете присвоить значение и экспортировать переменную за один шаг.

Чтобы проиллюстрировать присваивание и экспорт, создадим еще один bash из текущего bash интерпретатора, а затем запустим интерпретатор Korn из (ksh) созданного bash. Мы будем использовать команду ps для отображения информации о работающих процессах. Более подробно о команде ps мы узнаем, когда изучим понятие статус процесса далее в этом руководстве.

Не используете bash?

Интерпретатор bash принят по умолчанию во многих дистрибутивах Linux. Если вы работаете не с bash, то можете рассмотреть следующие способы, чтобы попрактиковаться в работе с bash.

- Используйте команду chsh -s /bin/bash чтобы изменить интерпретатор по умолчанию. Изменения вступят в силу во время вашего следующего захода в систему.
- Команда su - $USER -s /bin/bash создаст другой процесс, который будет являться дочерним по отношению к вашему текущему интерпретатору. Новый процесс запустит процесс входа в систему с командным интерпретатором bash.
- Создайте пользователя с командным интерпретатором bash для того, чтобы приготовиться к сдаче экзамена LPI.

Листинг 8. Переменные среды и shell

```
[ian@echidna ian]$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
30576 30575 -bash
[ian@echidna ian]$ bash
[ian@echidna ian]$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
16353 30576 bash
[ian@echidna ian]$ VAR1=var1
[ian@echidna ian]$ VAR2=var2
[ian@echidna ian]$ export VAR2
[ian@echidna ian]$ export VAR3=var3
[ian@echidna ian]$ echo $VAR1 $VAR2 $VAR3
var1 var2 var3
[ian@echidna ian]$ echo $VAR1 $VAR2 $VAR3 $SHELL
var1 var2 var3 /bin/bash
[ian@echidna ian]$ ksh
$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
16448 16353 ksh
$ export VAR4=var4
$ echo $VAR1 $VAR2 $VAR3 $VAR4 $SHELL
var2 var3 var4 /bin/bash
$ exit
$ [ian@echidna ian]$ echo $VAR1 $VAR2 $VAR3 $VAR4 $SHELL
var1 var2 var3 /bin/bash
[ian@echidna ian]$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
16353 30576 bash
[ian@echidna ian]$ exit
[ian@echidna ian]$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
30576 30575 -bash
[ian@echidna ian]$ echo $VAR1 $VAR2 $VAR3 $VAR4 $SHELL
/bin/bash
```

Примечание:

1. В начале этой последовательности у интерпретатора bash был PID 30576 .

2. У второго интерпретатора bash PID 16353, а его родительский PID 30576, то есть изначальный bash.
3. Мы создали переменные VAR1, VAR2, и VAR3 во втором экземпляре bash, но экспортировали только VAR2 и VAR3.
4. В интерпретаторе Korn, мы создали VAR4. Команда echo отображает значения только переменных VAR2, VAR3 и VAR4, и подтвердила, что VAR1 не была экспортирована. Вы не были удивлены, когда значение переменной SHELL не изменилось, хотя изменилось приглашение ввода? Вы не можете всегда полагаться на SHELL, чтобы определить в каком интерпретаторе идет работа, но команда ps позволит точно определить, что к чему. Заметим, что ps ставит дефис (-) перед первым экземпляром bash, чтобы дать нам понять, что это исходный командный интерпретатор.
5. Во втором экземпляре bash мы можем просмотреть VAR1, VAR2 и VAR3.
6. Наконец, когда мы возвращаемся в исходный интерпретатор, ни одна переменная в нем не существует.

Ранее мы обсуждали возможность использования кавычек как одинарных, так и двойных. Между ними есть существенная разница. Интерпретатор осуществляет подстановку shell переменных, находящиеся между двойными кавычками ($quot;), но не осуществляет подстановку, если используются одинарные ('). В предыдущем примере, мы создали новый экземпляр интерпретатора из другого и получили новый идентификатор процесса. Используя опцию -c вы можете передать команду в другой интерпретатор, который исполнит команду и произведет возврат. Если вы передаете строку в качестве команды в одинарных кавычках, то второй экземпляр интерпретатора их снимет и обработает строку. При использовании двойных кавычек подстановка переменных происходит до того как осуществится передача строки, поэтому результаты могут отличаться от того, что вы хотели ожидать. Интерпретатор и команда породят процесс, у которого будет свой PID. Листинг 9 иллюстрирует эти концепции. PID изначального интерпретатора bash выделен другим шрифтом.

Листинг 9. Кавычки и shell переменные

```
[ian@echidna ian]$ echo "$SHELL" '$SHELL' "$$" '$$'
/bin/bash $SHELL 19244 $$
[ian@echidna ian]$ bash -c "echo Expand in parent $$ $PPID"
Expand in parent 19244 19243
[ian@echidna ian]$ bash -c 'echo Expand in child $$ $PPID'
Expand in child 19297 19244
```

До сих пор все наши переменные заканчивались пробелом, таким образом, было понятно, где заканчивается имя переменной. На самом деле имя переменной может только состоять из букв, цифр или символа подчеркивания. Интерпретатор знает, что имя переменной заканчивается, как только встречается другой символ. Иногда необходимо использовать переменные в выражениях, где их значение может быть двусмысленным. В таких случаях вы можете использовать фигурные скобки, чтобы отделить имя переменной как показано в Листинге 10.

Листинг 10. Использование фигурных скобок с именами переменных

```
[ian@echidna ian]$ echo "-$HOME/abc-"
-/home/ian/abc-
[ian@echidna ian]$ echo "-$HOME_abc-"
--
[ian@echidna ian]$ echo "-${HOME}_abc-"
-/home/ian_abc-
```

Команда env

Команда env без каких-либо опций или параметров отображает текущие переменные среды. Вы также можете использовать ее, чтобы выполнить команду в предопределенной среде. Опция -i (или просто -) очищает текущую среду до того как выполнить команду, в то время как опция -u обнуляет переменные среды, которые вы не хотите передавать.

Листинг 11 содержит частичный вывод команды env без каких-либо параметров, а затем три примера, запускающие разные интерпретаторы без родительской среды. Внимательно их просмотрите прежде, чем мы их обсудим.

Листинг 11. Команда env

```
[ian@echidna ian]$ env
HOSTNAME=echidna
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=9.27.89.137 4339 22
SSH_TTY=/dev/pts/2
USER=ian
...
_=/bin/env
OLDPWD=/usr/src
[ian@echidna ian]$ env -i bash -c 'echo $SHELL; env'
/bin/bash
PWD=/home/ian
SHLVL=1
_=/bin/env
[ian@echidna ian]$ env -i ksh -c 'echo $SHELL; env'

_=/bin/env
PATH=/bin:/usr/bin
[ian@echidna ian]$ env -i tcsh -c 'echo $SHELL; env'
SHELL: Undefined variable.
```

Заметим, что bash установил переменную SHELL, но не экспортировал ее в среду, не смотря на то, что bash создал в среде три других переменных. В примере с ksh у нас содержится две переменных окружения, но наша попытка выдать на экран значение переменной SHELL приводит к появлению пустой строки. Наконец, tcsh не создал никаких переменных среды и выдал ошибку, когда мы попытались получить значение переменной SHELL.

Установка и обнуление переменных

Листинг 11 показал, как ведут себя интерпретаторы при обработке переменных и сред. Хотя это руководство уделяет внимание bash, следует знать, что не все интерпретаторы ведут себя одинаково. Более того, интерпретаторы ведут себя по-разному в зависимости от того, являются ли они исходными командными интерпретаторамиили нет. Сейчас мы просто скажем, что исходный командный интерпретатор это интерпретатор, который вы получаете при входе в систему; вы можете запустить другие командные интерпретаторы так, что они будут вести себя как исходные если пожелаете. Три интерпретатора в примере выше, запущенные с помощью команды env -i не являются исходными интерпретаторами. Попытайтесь передать опцию -l, чтобы увидеть разницу при запуске исходного командного

интерпретатора.

Давайте рассмотрим нашу попытку отобразить значение переменной SHELL в этих командных интерпретаторах:

- Когда bash запустился, он установил переменную SHELL, но не экспортировал ее автоматически в среду.
- Когда запустился ksh, он не установил переменную SHELL. Однако ссылка на неопределенную переменную среды эквивалентно ссылке на пустое значение.
- Когда запустился tcsh, то он не установил значение переменной SHELL. В этом случае поведение по умолчанию отлично от ksh (и bash) и в результате сгенерировалась ошибка, когда мы попытались получить доступ к переменной.

Вы можете использовать команду unset для обнуления переменной и удаления ее из списка shell переменных. Если переменная была экспортирована в среду, то она также будет удалена и из среды. Вы можете использовать команду set для управления поведением работы bash (или других интерпретаторов). Set является встроенной командой в интерпретаторе, поэтому опции зависят от конкретного интерпретатора. В bash опция -u сообщает bash, чтобы он не генерировал ошибку при ссылке на неопределенные переменные, а работал с ними как с пустыми значениями. Вы можете добавить различные опции к set с помощью - и отключить их с помощью +. Вы можете отобразить текущий список опций set с помощью echo $-.

Листнг 12. Unset и set

```
[ian@echidna ian]$ echo $-
himBH
[ian@echidna ian]$ echo $VAR1

[ian@echidna ian]$ set -u;echo $-
himuBH
[ian@echidna ian]$ echo $VAR1
bash: VAR1: unbound variable
[ian@echidna ian]$ VAR1=v1
[ian@echidna ian]$ VAR1=v1;echo $VAR1
v1
[ian@echidna ian]$ unset VAR1;echo $VAR1
bash: VAR1: unbound variable
[ian@echidna ian]$ set +u;echo $VAR1;echo $-

himBH
```

Вы можете использовать команду set без каких-либо опций, которая отобразит все ваши shell переменные и их значения (если есть). Есть также другая команда, declare, с помощью которой вы можете создавать, экспортировать и отображать значения shell переменных. О других опциях команд set и declare вы можете узнать из man-страниц. Мы рассмотрим man-страницы далее в этом разделе.

Команда exec

Последняя команда, которую мы рассмотрим в этом разделе это exec. Вы можете использовать команду exec, чтобы запустить другую команду, которая заместит текущий интерпретатор. В Листинге 13 порождается экземпляр bash, а затем используется exec, чтобы заместить его на интерпретатор Korn. После выхода из

интерпретатора Korn, вы оказываетесь в исходном интерпретаторе bash (в этом примере PID 22985).

Листинг 13. Использование exec

```
[ian@echidna ian]$ echo $$
22985
[ian@echidna ian]$ bash
[ian@echidna ian]$ echo $$
25063
[ian@echidna ian]$ exec ksh
$ echo $$
25063
$ exit
[ian@echidna ian]$ echo $$
22985
```

История команд

Если вы набирали команды, по мере того как читали руководство, то могли заметить, что часто используются почти одни и те же команды. Хорошая новость состоит в том, что bash может хранить историю ваших команд. По умолчанию история включена. Вы можете отключить ее с помощью команды set +o history и включить с помощью команды set -o history. Переменная среды HISTSIZE сообщает bash о том, сколько надо хранить строк. Набор других свойств определяет поведение и работу истории. Подробности смотрите в man-страницах bash.

Вот некоторые команды, которые вы можете использовать для работы с историей:

history
: Отображает всю историю

historyN
: Отображает последние N строк вашей истории

history -dN
: Удаляет строку N из вашей истории; это можно использовать, если, например, вы хотите удалить строку, содержащую пароль

!!
: Ваша последняя введенная команда

!N

!-N
: Команда, отстоящая на N шагов от текущей в истории (!-1 эквивалентно !!)

!#
: Текущая команда, которую вы набираете

!string
: Самая недавняя команда, которая начинается со строки string

!?string?
: Самая последняя команда, содержащая строку string

Вы можете использовать двоеточие (:), за которым следует определенное значение, чтобы получить доступ или изменить команду в истории. Листинг 14 показывает некоторые возможности истории.

Листинг 14. Управление историей

```
[ian@echidna ian]$ echo $$
22985
[ian@echidna ian]$ env -i bash -c 'echo $$'
1542
[ian@echidna ian]$ !!
env -i bash -c 'echo $$'
1555
[ian@echidna ian]$ !ec
echo $$
22985
[ian@echidna ian]$ !en:s/$$/$PPID/
env -i bash -c 'echo $PPID'
22985
[ian@echidna ian]$ history 6
1097  echo $$
1098  env -i bash -c 'echo $$'
1099  env -i bash -c 'echo $$'
1100  echo $$
1101  env -i bash -c 'echo $PPID'
1102  history 6
[ian@echidna ian]$ history -d1100
```

Команды в Листинге 14 делают следующее:

1. Вывод PID текущего интерпретатора
2. Запуск команды echo в новом экземпляре интерпретатора и вывод его PID
3. Запустить последнюю команду
4. Перезапустить команду, начинающуюся с 'ec'; произойдет запуск первой команды в этом примере
5. Запустить последнюю команду, начинающуюся с 'en', но заменить '$PPID' на '$$', поэтому на самом деле отобразится родительский PID
6. Отобразить последние 6 команд истории
7. Удалить команду под номером 1100, последняя команда echo

Вы можете редактировать истории в интерактивном режиме. Интерпретатор bash использует библиотеку readline для управления редактированием команд и истории. По умолчанию, клавиши и комбинации клавиш, которые используются для перемещения по истории или редактированию строк соответствуют тем, что используются в редакторе GNU Emacs. В Emacs комбинации клавиш обычно обозначаются как C-x или M-x, где x это обычная клавиша, а C и M это Control и Meta клавиши соответственно. На типичном PC клавишаCtrlсоответствует клавише Emacs Control, а клавиша Alt соответствует клавише Meta. В Таблице 5 содержатся некоторые доступные функции редактирования истории. Кроме комбинаций клавиш, показанных в Таблице 5, клавиши курсора, а также Home и End клавиши используются естественным образом для работы с историей. Дополнительные функции, а также возможности настройки опций с помощью файла инициализации readline (обычно это inputrc в вашем домашнем каталоге) можно найти в man-страницах.

Таблица 5. Редактирование истории с помощью команд emacs

Команда	Клавиша на клавиатуре PC	Описание
C-f	Стрелка вправо	Перейти на один знак вправо
C-b	Стрелка влево	Перейти на один знак влево
M-f	Alt-f	Перейти в начало следующего слова; В графических средах эта комбинация приводит к открытию меню File текущего окна
M-b	Alt-b	Перейти к началу предыдущего слова
C-a	Home	Перейти к началу строки
C-e	End	Перейти к концу строки
Backspace	Backspace	Удалить символ перед курсором
C-d	Del	Удалить символ, стоящий сразу после курсора (функции Del и Backspace можно поменять местами)
C-k	Ctrl-k	Удалить (убить) все до конца строки и сохранить для последующего использования
M-d	Alt-d	Удалить (убить) до конца слова и сохранить текст для последующего использования
C-y	Ctrl-y	Вставить текст, удаленный командой убить

Если вы предпочитаете управлять историей в режиме vi, то используйте команду set -o vi, чтобы переключиться в режим vi. Можете переключиться обратно в режим emacs с помощью команды set -o emacs. Когда вы извлекаете команду в режиме vi, то находитесь изначально в режиме вставки vi. Более подробно о редакторе vi смотрите в разделе Редактирование файлов в vi.

Пути

Одни команды bash являются встроенными, другие же наоборот внешними. Давайте теперь рассмотрим внешние команды и как их запускать, а также как отличить внутреннюю команду.

Где интерпретатор ищет команды?

Внешние команды представляют собой файлы в файловой системе. Дальше раздел Простое управление файлами этого руководства и руководства для Темы 104 раскрывают необходимые подробности. В системах Linux и UNIX все файлы являются частью огромного дерева, конем которого является /. В рассматриваемых выше примерах нашим текущим каталогом был домашний каталог пользователя. У обычных пользователей домашние каталоги находятся в /home каталоге, то есть /home/ian, в моем случае. Домашний каталог root находится в /root. После того как вы набрали команду, bash ищет ее в списке каталогов поиска по умолчанию, который представляет собой список каталогов, разделенных двоеточием и хранящийся в переменной окружения PATH.

Если вы хотите знать какая команда будет выполнена, если вы напечатаете определенную строку, то используйте команду which или type. В Листинге 15 показан мой путь по умолчанию, а также расположение нескольких команд.

Листинг 15. Поиск месторасположения команд

```
[ian@echidna ian]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/ian/bin
[ian@echidna ian]$ which bash env zip xclock echo set ls
alias ls='ls --color=tty'
/bin/ls
/bin/bash
/bin/env
/usr/bin/zip
/usr/X11R6/bin/xclock
/bin/echo
/usr/bin/which: no set in (/usr/local/bin:/bin:/usr/bin:/usr/X11R6/b
in:/home/ian/bin)
[ian@echidna ian]$ type bash env zip xclock echo set ls
bash is /bin/bash
env is /bin/env
zip is /usr/bin/zip
xclock is /usr/X11R6/bin/xclock
echo is a shell builtin
set is a shell builtin
ls is aliased to `ls --color=tty'
```

Заметим, что все каталоги в пути заканчиваются на /bin. Это общепринятое соглашение, но не требование. Команда which доложила нам, что команда ls является псевдонимом и что команда set не может быть найдена. В этом случае это можно интерпретировать, что команды либо не существует, либо она является встроенной. Команда type сообщила нам, что команда ls на самом деле является псевдонимом, а также она определила, что команда set является встроенной командой интерпретатора; также она сообщила, что есть встроенная команда echo, а также есть такая команда в /bin, которую мы нашли с помощью команды which. Эти две команды по-разному осуществляют свой вывод.

Мы видели, что команда ls, используемая для просмотра содержимого каталогов, на самом деле является псевдонимом. Псевдонимы представляют удобный способ

использования команд с различными наборами опций или же просто для альтернативного именования команды. В нашем примере опция --color=tty заставляет подсвечивать список файлов каталога в зависимости от типа файлов и каталогов. Попробуйте запустить dircolors --print-database, чтобы увидеть коды цветов, а также, какие цвета используются для конкретного типа файла.

У каждой из этих команд есть дополнительные опции. В зависимости от ваших требований вы можете использовать ту или иную команду. Я предпочитаю использовать which, когда уверен, что найду исполняемый файл и мне просто нужен его полный путь. Команда type выдает мне более точную информацию, которая мне иногда необходима в сценариях.

Запуск других команд

В Листинге 15 мы видели, что полный путь исполняемых файлов начинается с корневого каталога /. Например, программа xclock это на самом деле /usr/X11R6/bin/xclock, файл, расположенный в каталоге /usr/X11R6/bin. Если программа не находится в переменной PATH, то вы можете запустить ее, указав полный путь к программе и саму программу. Существует два вида путей, которые вы можете использовать:

- Абсолютные пути, которые начинаются с /, такие как мы видели в Листинге 15 (/bin/bash, /bin/env и так далее).
- Относительные пути эти пути относительно вашего текущего рабочего каталога, имя которого можно получить с помощью команды pwd. Такие команды не начинаются с /, но по крайней мере содержат один символ /.

Вы можете использовать абсолютные пути в независимости от вашего текущего рабочего каталога, но возможно будете использовать относительные пути, когда команда находится недалеко от текущего каталога. Предположим, что вы разрабатываете новую версию классической программы "Hello World!" в подкаталоге mytestbin вашего домашнего каталога. Возможно вы захотите использовать относительный путь и запустить команду как mytestbin/hello. Существует два специальных имени, которые вы можете использовать в указании пути; простая точка (.) ссылается на текущий каталог, и пара точек (..), которые ссылаются на родительский каталог текущего каталога. Так как ваш домашний каталог не находится в переменной окружения PATH (и так и должно быть), то вам понадобится указать явный путь к файлу, который вы хотите запустить из своего домашнего каталога. Например, если у вас есть копия программы hello в вашем домашнем каталоге, то для ее запуска можете просто использовать команду ./hello. Вы можете использовать как. так и .. как часть абсолютного пути, хотя одинарная .не очень полезна в данном случае. Вы можете использовать тильду (~) для ссылки на свой домашний каталог и ~usernamедля ссылки на домашний каталог пользователя username.Некоторые примеры приведены в Листинге 16.

Листнг 16. Абсолютные и относительные пути

```
[ian@echidna ian]$ /bin/echo Use echo command rather than builtin
Use echo command rather than builtin
[ian@echidna ian]$ /usr/../bin/echo Include parent dir in path
Include parent dir in path
[ian@echidna ian]$ /bin/./././echo Add a couple of useless path components
Add a couple of useless path components
[ian@echidna ian]$ pwd # See where we are
```

```
/home/ian
[ian@echidna ian]$ ../../bin/echo Use a relative path to echo
Use a relative path to echo
[ian@echidna ian]$ myprogs/hello # Use a relative path with no dots
-bash: myprogs/hello: No such file or directory
[ian@echidna ian]$ mytestbin/hello # Use a relative path with no dots
Hello World!
[ian@echidna ian]$ ./hello # Run program in current directory
Hello World!
[ian@echidna mytestbin]$ ~/mytestbin/hello # run hello using ~
Hello World!
[ian@echidna ian]$ ../hello # Try running hello from parent
-bash: ../hello: No such file or directory
```

Смена рабочего каталога

Также как вы можете исполнять программы из различных каталогов, вы можете изменять ваш текущий рабочий каталог, используя команду cd. Аргументом для cd должен быть абсолютный или относительный путь до каталога. В команде при указании путей вы также можете использовать ., .., ~, и ~username. Если вы наберете cd без параметров, то перейдете в домашний каталог. Передача в качестве параметра одиночного (-) означает переход в предыдущий рабочий каталог. Домашний каталог хранится в переменной окружения HOME, а предыдущий каталог хранится в переменной OLDPWD, поэтому cd эквивалентно cd $HOME, а cd - эквивалентно cd $OLDPWD. Обычно мы коротко говорим о смене каталога вместо полной смены текущего рабочего каталога.

Что касается команд, существует переменная среды CDPATH, которая содержит список каталогов, разделенных двоеточием, в которых должен происходить поиск (в дополнение к текущему рабочему каталогу), при разрешении относительных путей. Если решение использует путь из CDPATH, то cd напечатает на выходе полный путь найденного каталога. Обычно удачная смена каталога сопровождается появлением нового приглашения или немного модифицированного приглашения. Некоторые примеры показаны в Листинге17.

Листнг 17. Смена каталогов

```
[ian@echidna home]$ cd /;pwd
/
[ian@echidna /]$ cd /usr/X11R6;pwd
/usr/X11R6
[ian@echidna X11R6]$ cd ;pwd
/home/ian
[ian@echidna ian]$ cd -;pwd
/usr/X11R6
/usr/X11R6
[ian@echidna X11R6]$ cd ~ian/..;pwd
/home
[ian@echidna home]$ cd ~;pwd
/home/ian
[ian@echidna ian]$ export CDPATH=~
[ian@echidna mytestbin]$ cd /;pwd
/
[ian@echidna /]$ cd mytestbin
/home/ian/mytestbin
```

Рекурсивное применение команд

Многие Linux команды можно применять рекурсивно ко всем файлам в дереве каталогов. Например, у команды ls есть опция -R для рекурсивной выдачи списка каталогов, а у команд cp, mv, rm, и diff есть опция -r для рекурсивного применения. Раздел Простое управление файлами рассматривает рекурсию более подробно.

Подстановка команд

У bash есть чрезвычайно мощная возможность передачи результата одной программы на вход другой; это называется подстановкой команды. Это можно сделать, заключив команду, результаты которой вам нужны, в апострофы (`). При использовании множественных вложенных команд можно заключать команду между $(и).

В предыдущем руководстве "Подготовка к экзамену LPI 101 (тема 102): Установка Linux и управление пакетами" мы видели, что команда rpm может сказать какому пакету принадлежит какая команда; здесь было удобно применять подстановку команды. Теперь вы знаете, что мы действительно это делали.

Подстановка команды является бесценными инструментом при написании сценариев, а также при использовании в командной строке. В Листинге 18 показан пример, как получить абсолютный путь каталога из относительного, как найти пакет, который предоставляет команду /bin/echo, и как (будучи root) просмотреть метки трех разделов на жестком диске. Последний использует команду seq для создания последовательности целых чисел.

Листинг 18. Подстановка команды

```
[ian@echidna ian]$ echo '../../usr/bin' dir is $(cd ../../usr/bin;pwd)
../../usr/bin dir is /usr/bin
[ian@echidna ian]$ which echo
/bin/echo
[ian@echidna ian]$ rpm -qf `which echo`
sh-utils-2.0.12-3
[ian@echidna ian]$ su -
Password:
[root@echidna root]# for n in $(seq 7 9); do echo p$n `e2label
/dev/hda$n`;done
p7 RH73
p8 SUSE81
p9 IMAGES
```

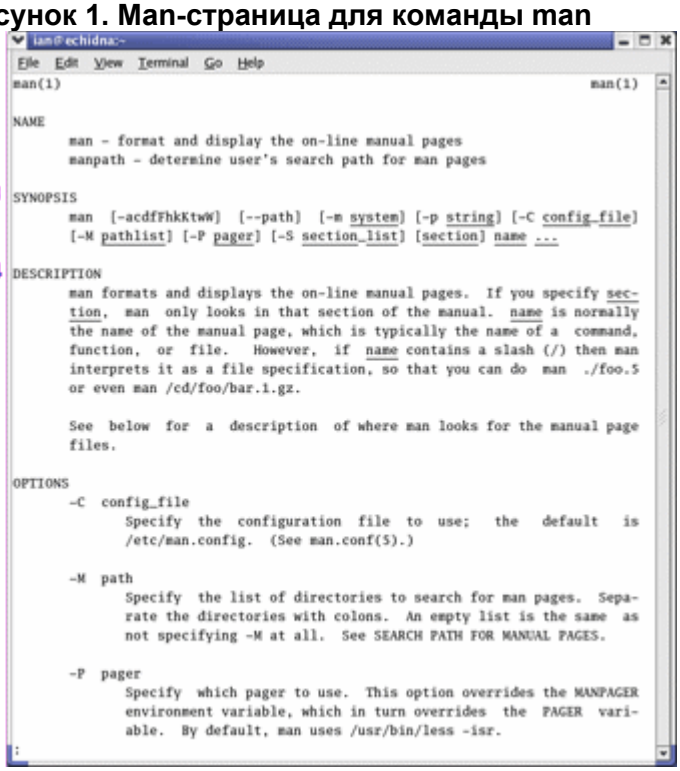
Man-страницы

В последней теме раздела этого руководства рассмотрим, как получить справку по командам Linux с помощью man-страниц и других видов документации.

Man-страницы и разделы

Главный (и традиционный) источник документации -- это man-страницы, доступ к которым можно получить с помощью команды man. На Рисунке 1 показана man-страница для команды man. Используйте команду man man для получения этой информации.

Рисунок 1. Man-страница для команды man



На Рисунке 1 представлены некоторые типичные пункты man-страниц:
Заголовок с именем команды, за которым в скобках следует номер раздела
Имя команды и другие похожие команды, которые обсуждаются в этой man-странице
Список опций и параметров примених к команде
Короткое описание команды
Подробное описание каждой опции

Также вы можете найти разделы по использованию, как сообщать ошибки, информацию об авторе, а также список других команд. Например, man-страница для man сообщает, что существуют дополнительные команды (и их руководства):

```
apropos(1), whatis(1), less(1), groff(1) и man.conf(5).
```

Обычно общими для man-страниц являются 8 разделов. Большинство страниц обычно ставиться при установке пакета, поэтому если пакет не установлен, то почти наверняка у вас будут отсутствовать его man-страницы. Кроме того, некоторые разделы страниц могут быть пустыми или почти пустыми. Наиболее общие разделы man-страниц это:

1. Команды пользователя (env, ls, echo, mkdir, tty)
2. Системные вызовы или функции ядра (link, sethostname, mkdir)
3. Библиотечные функции (acosh, asctime, btree, locale, XML::Parser)
4. Информация по оборудованию (isdn_audio, mouse, tty, zero)
5. Описание формата файолов (keymaps, motd, wvdial.conf)
6. Игры (заметим, что многие игры теперь работают в графическом режиме, поэтому могут иметь собственную систему помощи, а не man-страницы)
7. Разное (arp, boot, regex, unix utf8)
8. Системное администрирование (debugfs, fdisk, fsck, mount, renice, rpm)

Другие разделы могут включать 9 для документации по ядру Linux, n для новой

документации, o для старой документации и l для локальной документации.

Некоторые записи могут встречаться в нескольких разделах. Наши примеры показали, что mkdir содержится в разделах 1 и 2, а tty в разделах 1 и 4. Вы можете определить определенный раздел, например, man 4 tty или man 2 mkdir, или вы можете использовать опцию -a для получения списка всех разделов man-страниц.

Вы заметили на рисунке, что у man много опций, которые вы можете сами посмотреть. Сейчас давайте быстро взглянем на раздел команд "See also", имеющих отношение к man.

See also

Две важнейших команды, имеющих отношение к man, это whatis и apropos. Команда whatis ищет man-страницы для указанного вами имени и отображает информации об имени из соответствующих man-страниц. Команда apropos осуществляет поиск по ключевым словам в man-страниц и выводит те, которые содержат ваше слово. В Листинге 19 эти команды представлены.

```
Листинг 19. Примеры команд whatis и apropos
                [ian@lyrebird ian]$ whatis man
                man                   (1)  - format and display the on-
line manual pages
                man                   (7)  - macros to format man pages
                man [manpath]         (1)  - format and display the on-
line manual pages
                man.conf [man]        (5)  - configuration data for man
                [ian@lyrebird ian]$ whatis mkdir
                mkdir                 (1)  - make directories
                mkdir                 (2)  - create a directory
                [ian@lyrebird ian]$ apropos mkdir
                mkdir                 (1)  - make directories
                mkdir                 (2)  - create a directory
                mkdirhier             (1x)  - makes a directory hierarchy
```

Между прочим если вы не можете найти man-страницу для man.conf, то попытайтесь запустить man man.conf ig .

Вывод на экран команды man осуществляет специальная программа постраничного вывода. На большинстве Linux систем такой программой будет less. Другим вариантом может быть более старая программа more. Если вы хотите напечатать страницу, то определите опцию -t для форматирования страницы и печати, используя программу groff или troff.

У программы вывода less есть несколько команд, облегчающих поиск строк в отображаемом тексте. Используйте команду man less, чтобы узнать больше о /(поиск вперед), ?(поиск назад) и n (для последнего произведенного поиска), а также о многих других командах.

Другие источники документации

В дополнение к man-страницам, доступным из командной строки, фонд Free Software Foundation создал большое число info файлов, которые обрабатываются программой info. Она обладает большими возможностями навигации, в том числе и возможностью перехода в другую секцию. Наберите man info или info info, чтобы получить больше информации. Не все команды документированы в info, поэтому вы можете использовать как man-страницы так и info.

Существует несколько графических интерфейсов к man-страницам, как например xman (из проекта XFree86) и yelp (браузер помощи Gnome 2.0).

Если вы не можете найти справки по команде, попытайтесь запусить команду с опцией --help. Так вы, возможно, узнаете то что хотели или получите подсказку, где ещё можно поискать.

Текстовые потоки и фильтры

Фильтрация текста

Фильтрация текста -- это процесс преобразований над входным потоком текста до того как он будет выдан в выходной поток. Хотя как входной, так и выходной поток могут поступать из файла, в системах Linux и UNIX фильтрация преимущественно осуществляется через конвейер команд, когда вывод одной команды связывается или перенаправляется на ввод следующей команды. Программные каналы и перенаправления более подробно рассмотрены в разделе Потоки, программные каналы и перенаправления, но сейчас давайте взглянем на конвейер и простое перенаправление вывода с помощью операторов | и >.

Конвейер с помощью |

Вспомним из предыдущего раздела, что интерпретатор оперирует с тремя стандартными потоками ввода/вывода:

- stdin это стандартный поток ввода, через который поступает ввод командам.
- stdout это стандратный выходной поток, через который команды выводят свой выход.
- stderr это стандартный поток ошибок, через который выводятся ошибки в командах.

До сих пор, в этом руководстве, ввод представлял собой параметры, которые мы передавали командам, а вывод отображался на терминал. Многие команды обработки текста (фильтры) могут принимать входной поток, как из стандартного ввода, так и из файла. Чтобы использовать выход команды command1, как входной фильтр command2 вы должны соединить команды с помощью операции конвейерезации (|), как показано в Листинге 20.

Листинг 20. Связывание выхода command 1 со входом command2

```
command1 | command2
```

У любой из команд могут быть опции или аргументы, как вы увидите далее в этом разделе. Вы можете также использовать | для перенаправления вывода command2 в этом конвейере на вход другой команде, command3. Конструируя длинные конвейеры из команд, каждая из которых выполняет свою задачу, можно понять философию выполнения задач в Linux и UNIX. Также иногда вы будете видеть знак дефиса (-) вместо имени файла в качестве аргумента команды, в том значении, что ввод будет поступать из stdin, а не из файла.

Перенаправление вывода с помощью >

Приятно создавать конвейеры из нескольких команд и наблюдать результат на терминале, но бывают случаи, когда надо сохранить вывод в файл. Это можно сделать с помощью оператора перенаправления вывода (>).

В этой части руководства мы будем использовать небольшие файлы, поэтому давайте создадим каталог lpi103 и перейдем в него. Мы будем использовать > для перенаправления вывода echo в файл text1. Все это показано в Листинге 21. Заметим, что вывод не отображается на терминале, потому что он был перенаправлен в файл.

Листинг 21. Перенаправление вывода command 1 в файл

```
[ian@echidna ian]$ mkdir lpi103
[ian@echidna ian]$ cd lpi103
[ian@echidna lpi103]$ echo -e "1 apple\n2 pear\n3 banana">text1
```

Теперь, имея в арсенале простые инструменты для создания конвейера и перенаправления, взглянем на наиболее распространенные утилиты обработки текста в UNIX и Linux. Этот раздел описывает некоторые простые возможности; чтобы узнать больше, смотрите соответсвующие страницы руководств по этим командам.

Cat, tac, od и split

Вы создали файл test1, теперь вы захотите посмотреть его содержимое. Используйте команду cat (сокращение от catenate),чтобы отобразить содержимое файла на stdout. Листинг 22 проверяет содержимое файла, созданного выше.

Листинг 22. Вывод содержимого файла с помощью cat

```
[ian@echidna lpi103]$ cat text1
1 apple
2 pear
3 banana
```

Команда cat принимает ввод из stdin, если вы не определите имя файла (или если

напишите - как имя файла). Давайте используем эту возможность, а также перенаправление вывода, чтобы создать еще один текстовый файл как в Листинге 23.

Листинг 23. Создание текстового файла с помощью cat

```
[ian@echidna lpi103]$ cat>text2
9       plum
3       banana
10      apple
```

В Листинге 23 cat продолжает читать из stdin до конца файла. Используйте комбинацию Ctrl-d (нажмите Ctrl, а затем нажмите d), чтобы послать сигнал конца файла. Такая же комбинация клавиш используется для выхода из bash. Заметим, что клавиша tab позволяет выровнить в столбец имена фруктов.

Случайно вам захотелось отобразить файл в обратном порядке. Разумеется, для этого тоже существует текстовый фильтр под названием tac (перестановка букв в cat). Листинг 24 отображает как новый файл text2, так и старый text1 в обратном порядке. Заметим, как просто соединились два файла.

Листинг 24. Реверсивное отображение с помощью tac

```
[ian@echidna lpi103]$ tac text2 text1
10      apple
3       banana
9       plum
3 banana
2 pear
1 apple
```

Теперь положим, что вы отобразили два текстовых файла с помощью cat и tac и заметили разницу в выравнивании. Чтобы понять, почему это так, необходимо взглянуть на управляющие символы в файле. Так как они не имеют графического представления, то нам необходимо создать дамп файла в формате, который позволит вам найти и интерпретировать эти особые символы. Пакет текстовых утилит GNU включает команду od (или OctalDump) специально для этой цели.

У команды od есть несколько опций, как например -A для управления основанием смещений файла и -t для управления формой отображения содержимого файла. Основание может быть o, (восьмиричное - по умолчанию), d (десятичное), x (шестнадцатиричное) или n (смещения не отображаются). Вы можете отобразить файл в виде восьмиричном, шестнадцатиричном, десятичном, с плавающей точкой, ASCII с escape последовательностями или именованными символами (nl для новой строки, ht для горизонтальной табуляции и так далее). В Листинге 25 представлены некоторые доступные форматы дампа файла text2.

Листинг 25. Дамп файлов с помощью od

```
[ian@echidna lpi103]$ od text2
0000000 004471 066160 066565 031412 061011 067141 067141 005141
0000020 030061 060411 070160 062554 000012
0000031
[ian@echidna lpi103]$ od -A d -t c text2
0000000   9  \t   p   l   u   m  \n   3  \t   b   a   n   a   n   a  \n
0000016   1   0  \t   a   p   p   l   e  \n
0000025
[ian@echidna lpi103]$ od -A n -t a text2
   9  ht   p   l   u   m  nl   3  ht   b   a   n   a   n   a  nl
   1   0  ht   a   p   p   l   e  nl
```

Замечение:

1. Опция -A утилиты cat предоставляет альтернативный способ увидеть, где завершаются строки и символы табуляции. Для получения подробной информации смотрите man-страницы.
2. Если у вас есть знания об устройстве ЭВМ, то возможно вас заинетересует утилита hexdump, которая являются частью другого набора утилит. Она здесь не рассматривается, поэтому обратитесь к man-страницам.

Наши тестовые файлы очень малы, но иногда требуется разбить большие файлы на несколько небольших. Например, вы хотите разбить большой файл на куски объемом с CD, чтобы их записать да CD и отправить по почте кому-нибудь, кто может создать для вас DVD. Команда split сделает это таким образом, что cat можно будет использовать для простого воссоздания файла. По умолчанию, файлы на выходе команды split имеют префикс 'x' в имени, за которым следует суффикс 'aa', 'ab', 'ac', ..., 'ba', 'bb' и так далее. Существуют опции, позволяющие изменять эти умолчания. Вы также можете управлять размером выходных файлов, как в строках, так и в байтах. В Листинге 26 происходит разделение двух текстовых файлов с разными префиксами в именах выходных файлов. Мы разделим text1 на файлы, содержащие не более двух строк, а text2 на файлы размером не более 18 байт. Затем мы используем cat для отображения различных частей, а также для отображения всего файла, используя подстановку, которая рассмотрена в разделе шаблоны и подстановки позже в этом руководстве.

Листинг 26. Разделение и воссоединение с помощью split и cat

```
[ian@echidna lpi103]$ split -l 2 text1
[ian@echidna lpi103]$ split -b 18 text2 y
[ian@echidna lpi103]$ cat yaa
9       plum
3       banana
10[ian@echidna lpi103]$ cat yab
        apple
[ian@echidna lpi103]$ cat y*
9       plum
3       banana
10      apple
```

Заметим, что получившийся файл yab не содержит символ новой строки, поэтому

наше приглашение было смещено, когда мы использовали cat для его отображения.

Wc, head и tail

Cat и tac отображают файл целиком. Для маленьких файлов, с которыми имеем дело мы, это нормально, но предположим, что у вас большой файл. Для начала вы можете использовать команду wc (Word Count), чтобы посмотреть размер файла. Команда wc отображает число строк, слов и байт в файле. Вы также можете узнать число байт с помощью ls -l. Листинг 27 показывает длинный формат списка каталогов для двух файлов, а также вывод команды wc.

Листинг 27. Использование wc с текстовыми файлами

```
[ian@echidna lpi103]$ ls -l text*
-rw-rw-r--    1 ian      ian            24 Sep 23 12:27 text1
-rw-rw-r--    1 ian      ian            25 Sep 23 13:39 text2
[ian@echidna lpi103]$ wc text*
       3       6      24 text1
       3       6      25 text2
       6      12      49 total
```

Различные опции позволяют вам контролировать вывод команды wc илиотображать другую информацию, как например максимальная длина строки. Более подробно написано в man-страницах.

Две команды позволяют вам отобразить как начало (head) так и конец файла (tail). Это команды соответственно head и tail. Их можно использовать как фильтры, или же они могут принимать имя файла как аргумент. По умолчанию они отображают первые (или последние) 10 строк файла или потока.В Листинге 28 использована команда dmesg для отображения сообщений о загрузке совместно с wc, tail и head чтобы узнать, что всего имеется 177 сообщений, затем отображаются последние 10 строк, и, наконец, отображаются шесть сообщений, начиная с 15 от конца. Некоторые строки были обрезаны в этом выводе (об этом свидетельствует ...).

Листинг 28. Использование wc, head и tail для отображения загрузочных сообщений

```
[ian@echidna lpi103]$
[ian@echidna lpi103]$ dmesg | wc
     177    1164    8366
[ian@echidna lpi103]$ dmesg | tail
i810: Intel ICH2 found at IO 0x1880 and 0x1c00, MEM 0x0000 and ...
i810_audio: Audio Controller supports 6 channels.
i810_audio: Defaulting to base 2 channel mode.
i810_audio: Resetting connection 0
ac97_codec: AC97 Audio codec, id: ADS98 (Unknown)
i810_audio: AC'97 codec 0 Unable to map surround DAC's (or ...
i810_audio: setting clocking to 41319
Attached scsi CD-ROM sr0 at scsi0, channel 0, id 0, lun 0
sr0: scsi3-mmc drive: 0x/32x writer cd/rw xa/form2 cdda tray
Uniform CD-ROM driver Revision: 3.12
[ian@echidna lpi103]$ dmesg | tail -n15 | head -n 6
```

```
agpgart: Maximum main memory to use for agp memory: 941M
agpgart: Detected Intel i845 chipset
agpgart: AGP aperture is 64M @ 0xf4000000
Intel 810 + AC97 Audio, version 0.24, 13:01:43 Dec 18 2003
PCI: Setting latency timer of device 00:1f.5 to 64
i810: Intel ICH2 found at IO 0x1880 and 0x1c00, MEM 0x0000 and ...
```

Другим популярным использованием tail является слежение за файлом с помощью опции -f, обычно построчно. Это может полезным, если у вас есть фоновый процесс, который генерирует вывод в файл, и вы хотите проверить, что он сделал. В этом режиме tail будет работать, пока вы не прекратите его работу (с помощью Ctrl-c), отображая строки по мере того, как они будут поступать в файл.

Expand, unexpand и tr

Когда мы создали файлы text1 и text2, то использовали в text2 символы табуляции. Иногда требуется заменить символы табуляции на другие символы и наоборот. Команды expand и unexpand этим и занимаются. Опция -t в обеих командах позволяет установить шаг табуляции. Таким образом, каждый символ табуляции заменяется на этот шаг. В Листинге 29 показано, как заменить символы табуляции в text2 на пробелы, а также странная последовательность expand и unexpand которая переупорядочивает текст в text2.

Листинг 29. Использование expand и unexpand

```
[ian@echidna lpi103]$ expand -t 1 text2
9 plum
3 banana
10 apple
[ian@echidna lpi103]$ expand -t8 text2|unexpand -a -t2|expand -t3
9           plum
3           banana
10       apple
```

К сожалению, вы не можете использовать unexpand, чтобы заменить пробелы в text1 на символы табуляции, так как unexpand необходимо по крайней мере два пробела для преобразования их в символ табуляции. Однако вы можете использовать команду tr которая переводит символы из одного набора (set1) в соответствующие символы другого набора(set2). В Листинге 30 показано, как использовать tr, чтобы преобразовать пробелы в символы табуляции. Так как tr это фильтр, то входные данные для него вы генерируете с помощью команды cat.Этот пример также иллюстрирует применение - для обозначения стандартного ввода в cat.

Листинг 30. Использование expand и unexpand

```
[ian@echidna lpi103]$ cat text1 |tr ' ' '\t'|cat - text2
1       apple
2       pear
3       banana
9       plum
3       banana
10      apple
```

Если вы не уверены в том, что происходит в последних двух примерах, то наберите od, чтобы проверить каждую стадию конвейера; например

```
cat text1 |tr ' ' '\t' | od -tc
```

Pr, nl и fmt

Команда pr используется для форматирования файлов перед печатью. Заголовок по умолчанию включает имя файла и даты и время создания файла, а также номер страницы и двух пустых строк сносок. Когда выходной поток создается из нескольких файлов или из входного потока, то текущая дата и время появляются вместо имени файла и даты создания. Вы можете напечатать файлы параллельно в столбцах и управлять с помощью опций различными возможностями форматирования. Как обычно, смотрите man-страницы, чтобы узнать подробности.

Команда nl нумерует строки, что может быть полезно при печати файлов. Вы также можете нумеровать строки с помощью опции -n команды cat. На Листинге 31 показано как напечатать наш файл text 1, а затем как пронумеровать text2 и напечатать его параллельно с text1.

Листинг 31. Нумерация и форматирование для печати

```
[ian@echidna lpi103]$ pr text1 | head


2005-09-23 12:27                      text1                      Page 1


1 apple
2 pear
3 banana


[ian@echidna lpi103]$ nl text2 | pr -m - text1 | head


2005-09-26 11:48                                                 Page 1


     1  9       plum                1 apple
     2  3       banana              2 pear
     3  10      apple               3 banana
```

Другой полезной командой форматирования текста является fmt, которая форматирует текст так, что он подходит по определенным размерам. Вы можете соединить несколько коротких строк, а также разделить на несколько строк. В Листинге 32 мы создаем файл text3 с одной большой строкой текста, используя возможности истории с помощью !#:*, чтобы не печатать наше предложение четыре раза. Мы также создадим файл text4, содержащий по одному слову на строке. Затем мы будем использовать cat, чтобы отобразить их в неформатированном виде, включая символ '$' для обозначения конца строк. Наконец, мы используем fmt для

форматирования их с максимальной шириной строкой в 60 символов. Снова, обращайтесь к man-страницам за более подробной информацией.

Листинг 32. Форматирование по максимальной длине строки

```
[ian@echidna lpi103]$ echo "This is a sentence. " !#:* !#:1-$>text3
echo "This is a sentence. " "This is a sentence. " "This is a sentenc
e. " "This is a sentence. ">text3
[ian@echidna lpi103]$ echo -e "This\nis\nanother\nsentence.">text4
[ian@echidna lpi103]$ cat -et text3 text4
This is a sentence.  This is a sentence.  This is a sentence.  This i
s a sentence. $
This$
is$
another$
sentence.$
[ian@echidna lpi103]$ fmt -w 60 text3 text4
This is a sentence.  This is a sentence.  This is a
sentence.  This is a sentence.
This is another sentence.
```

Sort и uniq

Команда sort сортирует ввод, согласно схеме локали (LC_COLLATE) в системе. Команда sort также может соединять файлы и проверять, является ли файл отсортированным или нет.

Листинг 33 иллюстрирует применение команды sort для сортировки двух файлов, после того как мы преобразовали пробелы в символы табуляции в text1. Так как порядок сортировки происходит по символам, то результаты могут вас удивить. К счастью команда sort может сортировать как по числовым значениям, так и по символам. Порядок сортировки можно определить как для целой записи, так для каждого поля. Пока вы не укажите другой разделитель, поля будут разделяться пробелами и табуляторами. Второй пример в Листинге 33 показывает сортировку первого поля по цифрам, а второго поля по буквам с помощью схемы упорядочивания (в алфавитном порядке). Он также иллюстрирует применение опции -u для уничтожения любых дублируемых строк.

Листинг 33. Сортировка по символам и числам

```
[ian@echidna lpi103]$ cat text1 | tr ' ' '\t' | sort - text2
10      apple
1       apple
2       pear
3       banana
3       banana
9       plum
[ian@echidna lpi103]$ cat text1|tr ' ' '\t'|sort -u -k1n -k2 - text2
1       apple
2       pear
3       banana
9       plum
10      apple
```

Заметим, что у нас все еще есть две строчки, содержащие фрукт "apple". Другая команда uniq дает нам дополнительный контроль над выявлением дублирующих строк. Команда uniq обычно работает с отсортированными файлами, но удаляет последовательные одинаковые строки из любого файла, отсортированного или нет. Команда uniq также может игнорировать некоторые поля. В Листинге 34 показана сортировка двух файлов по второму полю (имени фрукта), а затем уничтожение идентичных по второму полю строк.

Листинг 34. Использование uniq

```
[ian@echidna lpi103]$ cat text1|tr ' ' '\t'|sort -k2 - text2|uniq -f1
10      apple
3       banana
2       pear
9       plum
```

Сортировка производилась согласно схеме упорядочивания, поэтому uniq выдало результат "10 apple", а не "1 apple". Попытайтесь добавить сортировку первого поля по числам, чтобы увидеть изменения.

Cut, paste и join

Давайте рассмотрим еще три команды, которые работают с полями в текстовых данных. Особенно эти команды полезны при работе с табулированными данными. Первая команда это cut, которая извлекает поля из текста. По умолчанию символом разделителем является табулятор. Листинг 35 использует cut для разделения двух столбцов text2, а затем использует пробел как выходной разделитель, что является довольно специфичным способом преобразования символов табуляции в пробелы.

Листинг 35. Использование cut

```
[ian@echidna lpi103]$ cut -f1-2 --output-delimiter=' ' text2
9 plum
3 banana
10 apple
```

Команда paste вставляет строки из двух или более файлов параллельно, подобно тому, как команда pr объединяет два файла с помощью опции -m. Листинг 36 демонстрирует результат вставки двух текстовых файлов.

Листинг 36. Вставка файлов

```
[ian@echidna lpi103]$ paste text1 text2
1 apple 9       plum
2 pear  3       banana
3 banana        10      apple
```

Эти примеры показывают простую вставку, но paste может вставлять данные из

одного или нескольких файлов различными способами. За подробностями обращайтесь к man-страницам.

Последняя команда манипулирования с полями это join, которая объединяет файлы на основе соответствия полей. Эти файлы должны быть отсортированы по объединяемому полю. Так как text2 не отсортирован по числовому порядку, мы можем отсортировать его, а затем объединить две строки, которые имеют одинаковое поле (а данном случае 3). Давайте также создадим новый файл, text5, отсортировав text 1 по второму полю (имени фрукта), а затем заменив пробелы на символы табуляции. Если мы затем отсортируем text2 и объединим его с text 5 по второму полю, то получим два совпадения (яблоко и банан). Листинг 37 иллюстрирует эти примеры.

Листинг 37. Объединение файлов по полям

```
[ian@echidna lpi103]$ sort -n text2|join -1 1 -2 1 text1 -
3 banana banana
[ian@echidna lpi103]$ sort -k2 text1|tr ' ' '\t'>text5
[ian@echidna lpi103]$ sort -k2 text2 | join -1 2 -2 2 text5 -
apple 1 10
banana 3 3
```

Поле, используемое для объединения, указывается отдельно для каждого файла. Вы можете, например, объединить файл по полю 3 с файлом по полю 10.

Sed

Sed это streameditor (потоковый редактор). Несколько статей developerWorks, а также много книг и глав посвящено sed (смотри Ресурсы).Sed чрезвычайно мощный инструмент, а задачи, которые можно выполнить с его помощью, ограничены лишь вашим воображением. Это небольшое введение должно пробудить у вас интерес к sed, но оно не является полным или расширенным.

Как и другие команды, которые мы рассмотрели, sed может работать как фильтр или получать ввод из файла. Вывод осуществляется на стандартный поток вывода. Sed загружает строки из ввода в пространство шаблонов, применяет команды редактирования sed к содержимому пространства шаблонов, а затем осуществляет на стандартный вывод пространства шаблонов. Sed может скомпоновать несколько строк в пространстве шаблонов, и может результат записать в файл, записать только частичный вывод, или же вообще ничего не записывать.

Sed использует синтаксис регулярных выражений (смотри Поиск с помощью регулярных выражений далее в этом руководстве) для поиска и избирательной замены текста в пространстве шаблонов, а также выбора тех строк, над которыми необходимо провести набор команд редактирования. Специальный буфер предоставляет временное хранилище для текста. Буфер может заменить пространство шаблонов, может быть добавлен к пространству шаблонов или же вести обмен с пространством шаблонов. Sed имеет ограниченный набор команд, но при использованию синтаксиса регулярных выражений и буфера может предоставлять потрясающие возможности. Набор команд sed обычно называется sed-сценарием.

В Листинге 38 представлено три простых sed-сценария. В первом мы используем

команду s (замещения) буквы 'a' в нижнем регистре на верхний в каждой строке. Этот пример замещает только первые вхождения 'a', поэтому во втором примере мы добавили флаг 'g' (от глобальный) для замещения всех вхождений буквы. В третьем сценарии мы рассматриваем команду d (удалить) для удаления строки. В нашем примере мы используем адрес второй строки, чтобы показать, что только ее необходимо удалить. Мы разделяем команды точкой с запятой (;) а затем используем глобальное замещение, которое мы использовали во втором сценарии для замены 'a' на 'A'.

Листинг 38. Начало работы с sed-сценариями

```
[ian@echidna lpi103]$ sed 's/a/A/' text1
1 Apple
2 peAr
3 bAnana
[ian@echidna lpi103]$ sed 's/a/A/g' text1
1 Apple
2 peAr
3 bAnAnA
[ian@echidna lpi103]$ sed '2d;$s/a/A/g' text1
1 apple
3 bAnAnA
```

В дополнение работе с одиночными строками, sed может работать с группой строк. Начало и конец диапазона разделяется запятой (,) и это может быть, как и номер строки, каретка (^), означающая начало файла, так и знак доллара ($), означающий конец файла. Зная адрес или диапазон адресов, вы можете сгруппировать несколько команд и заключить их в фигурные скобки ({ и }) так, что эти команды будут выполнены только на определенном диапазоне строк. Листинг 39 иллюстрирует два способа глобальной подстановки, применимой только к двум последним строкам файла. Он также иллюстрирует применение опции -e для исполнения команд в пространстве шаблонов. При использовании круглых скобок команды необходимо разделять запятыми.

Листинг 39. Sed с использованием адресации

```
[ian@echidna lpi103]$ sed -e '2,${' -e 's/a/A/g' -e '}' text1
1 apple
2 peAr
3 bAnAnA
[ian@echidna lpi103]$ sed -e '/pear/,/bana/{' -e 's/a/A/g' -e '}' text1
1 apple
2 peAr
3 bAnAnA
```

Sed-сценарии могут также храниться в файлах. На самом деле вы захотите так сделать для часто используемых сценариев. Помните, раньше мы использовали команду tr для замены пробелов в text1 на символы табуляции. Давайте теперь сделаем это с помощью sed-сценария, хранящегося в файле. Мы будем использовать команду echo для создания файла. Результаты представлены в Листинге 40.

Листинг 40. Использование sed-сценария

```
[ian@echidna lpi103]$ echo -e "s/ /\t/g">sedtab
[ian@echidna lpi103]$ cat sedtab
s/ /    /g
```

```
[ian@echidna lpi103]$ sed -f sedtab text1
1       apple
2       pear
3       banana
```

Существует множество таких удобных сценариев как в Листинге 40. Смотри Ресурсы, чтобы узнать дополнительную информацию.

Наш последний пример sed использует команду = для вывода номеров строк, а затем фильтрации вывода опять через sed для имитации эффекта команды nl для нумерации строк. Листинг 41 использует = для вывода номеров строк, а затем использует команду N для помещения каждой второй строки в пространство шаблонов в продолжение первой, и наконец удаляет символ новой строки (\n) между двумя строками в пространстве шаблонов.

Листинг 41. Нумерация строк в sed

```
[ian@echidna lpi103]$ sed '=' text2
1
9       plum
2
3       banana
3
10      apple
[ian@echidna lpi103]$ sed '=' text2|sed 'N;s/\n//'
19      plum
23      banana
310     apple
```

Не совсем то, что мы хотели! Мы хотели, чтобы нумерация была выровнена по столбцу, а также, чтобы номера и строки файла отделяли несколько пробелов. В Листинге 42 мы вводим несколько строк команд (заметим второе приглашение >). Изучите пример и посмотрите объяснение ниже.

Листинг 42. Нумерация строк с sed - второй подход

```
[ian@echidna lpi103]$ cat text1 text2 text1 text2>text6
[ian@echidna lpi103]$ ht=$(echo -en "\t")
[ian@echidna lpi103]$ sed '=' text6|sed "N
> s/^/      /
> s/^.*\(......\)\n/\1$ht/"
     1  1 apple
     2  2 pear
     3  3 banana
     4  9   plum
     5  3   banana
     6  10  apple
     7  1 apple
     8  2 pear
     9  3 banana
    10  9   plum
    11  3   banana
    12  10  apple
```

Шаги, которые мы предприняли:

1. Мы использовали cat, чтобы создать файл из 12 строк с помощью двух копий text1 и text2. Нет никакого интереса в форматировании чисел, если все они одного порядка.
2. Bash использует клавишу tab для автозавершения команды, поэтому полезно иметь уже заготовленный символ табуляции, чтобы использовать его, когда он действительно понадобиться. Мы используем команду echo, чтобы вывести и сохранить его в shell переменной 'ht'.
3. Мы создаем поток, содержащий номера строк и данные как мы делали до этого и фильтруем его через вторую копию sed.
4. Мы вторые строки вместе с первыми в пространство шаблонов
5. Мы предваряем нашу строку с нумерацией в начале пространства шаблонов (обозначаемого ^) шестью пробелами.
6. Затем замещаем строку с пробелами и символом новой строки символом табуляции. Заметим, что левая часть команды 's' использует '\(' и '\)' для обозначения символов, которые мы хотим использовать в правой части. В правой части мы ссылаемся на первое такое найденное множество (и только такое в данном примере) как \1. Заметим, что наша команда находится между двойных кавычек ("), так что данная подстановка будет определена для $ht.

Последняя версия (версия 4) sed содержит документацию в формате info и включает много превосходных примеров. Их нет в старой версии 3.02. GNU sed примет команду sed --version и отобразит свою версию.

Простое управление файлами

Просмотр каталогов

Как мы сказали ранее, при обсуждении путей в разделе об использовании командной строки, все файлы в системах Linux и UNIX® являются частью большого дерева файловой системы, корнем которого является /.

Просмотр записей каталогов

Если вы проработали предыдущий раздел, то создали каталог lpi103 в своем домашнем каталоге. Имена файлов и каталогов могут быть как абсолютными, что значит, что они начинаются с / так и относительными к текущему рабочему каталогу, что значит, что они не начинаются с /. Абсолютное имя файла или каталога состоит из /, за которым следует последовательность из 0 или больше имен каталогов, каждый из которых отделяется /, а затем конечного имени. Если вы знаете относительное имя файла или каталога, то просто соедините абсолютное имя текущего рабочего каталога, / и относительное имя. Например, каталог lpi103, который мы создали в прошлом разделе, был создан в моем домашнем каталоге /home/ian, поэтому его полный или абсолютный путь /home/ian/lpi103. Листинг 43 показывает три разных способа использования команды ls для просмотра списка файлов каталога.

Листинг 43. Просмотр записей каталога

```
[ian@echidna lpi103]$ pwd
/home/ian/lpi103
[ian@echidna lpi103]$ echo $PWD
/home/ian/lpi103
[ian@echidna lpi103]$ ls
sedtab  text2  text4  text6  xab  yab
text1   text3  text5  xaa    yaa
[ian@echidna lpi103]$ ls "$PWD"
sedtab  text2  text4  text6  xab  yab
text1   text3  text5  xaa    yaa
[ian@echidna lpi103]$ ls /home/ian/lpi103
sedtab  text2  text4  text6  xab  yab
text1   text3  text5  xaa    yaa
```

Как вы видите, можно передать имя каталога в качестве параметра команде ls и она выдаст содержимое этого каталога.

Подробности о списке файлов

На устройстве хранения файл или каталог представляет собой коллекцию блоков. Информация о файле содержится в inode, который хранит информацию о владельце, последнем времени доступа, размере, файл это или каталог, права доступа. Номер inode также известный как последовательный номер файла уникален в переделах определенной файловой системы. Мы можем использовать опцию -l (или --format=long) для отображения некоторой информации, хранящейся в inode.

По умолчанию команда ls не выводит специальные файлы, чьи имена начинаются с точки (.). У каждого каталога, кроме корневого, есть две специальных записи -- это сам каталог (.) и родительский каталог (..). У корневого каталога отсутствует родительский каталог.

Листинг 44 использует опции -l и -a для отображения длинного формата списка всех файлов, включая записи. и .. .

Листинг 44. Расширенный формат вывода

```
[ian@echidna lpi103]$ ls -al
total 56
drwxrwxr-x    2 ian      ian          4096 Sep 30 15:01 .
drwxr-xr-x   94 ian      ian          8192 Sep 27 12:57 ..
-rw-rw-r--    1 ian      ian             8 Sep 26 15:24 sedtab
-rw-rw-r--    1 ian      ian            24 Sep 23 12:27 text1
-rw-rw-r--    1 ian      ian            25 Sep 23 13:39 text2
-rw-rw-r--    1 ian      ian            84 Sep 25 17:47 text3
-rw-rw-r--    1 ian      ian            26 Sep 25 22:28 text4
-rw-rw-r--    1 ian      ian            24 Sep 26 12:46 text5
-rw-rw-r--    1 ian      ian            98 Sep 26 16:09 text6
-rw-rw-r--    1 ian      ian            15 Sep 23 14:11 xaa
-rw-rw-r--    1 ian      ian             9 Sep 23 14:11 xab
-rw-rw-r--    1 ian      ian            18 Sep 23 14:11 yaa
-rw-rw-r--    1 ian      ian             7 Sep 23 14:11 yab
```

В Листинге 44, первая строка показывает общее число дисковых блоков (56), занимаемых выведенным на экран файлами. Оставшиеся поля расскажут о файле.

- Первое поле (drwxrwxr-x или -rw-rw-r-- в этом случае) говорит нам о том, является ли запись обычным файлом (-) или каталогом (d). Также вы можете увидеть символьные ссылки (l), о которых мы узнаем чуть позже или другие значения для специальных файлов (как например, для файлов в файловой системе /dev). За типом файла следует три набора прав доступа (как rwx или r--) для владельца, членов группы владельца и всех остальных. Три значения соответственно указывают пользователю, группе и всем остальным есть ли у них право на чтение (r), запись (w) или исполнение (x). Другие возможности использования как setuid будут рассказаны позже.
- Следующее числовое поле говорит нам число жестких ссылок на файл. Мы говорили, что inode содержит информацию о файле. Запись в каталоге о файле содержит жесткую ссылку (или указатель) на inode для этого файла, так, что у каждой записи в каталоге есть по крайней мере одна жесткая ссылка. У записей каталога есть дополнительная ссылка на запись . и по одной записи .. для каждого подкаталога. Из приведенного выше листинга видно, что у моего домашнего каталога есть несколько подкаталогов.
- Следующие два поля представляют владельца файла и группу, к которой он принадлежит. Некоторые системы, такие как Red Hat, по умолчанию каждому пользователю предоставляют свою группу. В других системах все пользователи могут быть в одной или нескольких группах.
- Следующее поле указывает размер файла.
- Предпоследнее поле указывает время последней модификации.
- Последнее поле содержит имя файла или каталога.

Опция -i команды ls отобразит для вас номера inode. Мы поговорим о них чуть позже, а также когда будем обсуждать жесткие и символьные ссылки в руководстве для темы 104.

Множество файлов

Вы также можете определить несколько параметров команде ls, каждый из которых будет или именем файла или каталога. Если имя представляет собой каталог, то команда ls отобразит содержимое этого каталога, а не саму запись о каталоге. В нашем примере, предположим мы хотим получить информацию о каталоге lpi103. Команда ls -l ../lpi103 отобразит нам список как в предыдущем примере. Листинг 45 покажет, как использовать ls -ld и как отобразить список записей для нескольких файлов и каталогов.

Листинг 45. Использование ls -d

```
[ian@echidna lpi103]$ ls -ld ../lpi103 sedtab xaa
drwxrwxr-x    2 ian      ian          4096 Oct  2 18:49 ../lpi103
-rw-rw-r--    1 ian      ian             8 Sep 26 15:24 sedtab
-rw-rw-r--    1 ian      ian            15 Sep 23 14:11 xaa
```

Заметим, что время модификации lpi103 отличается от того, чтобы было в предыдущем листинге. Также как и в предыдущем листинге, время доступа файлов и каталогов изменено. Вы этого ожидали? Думаю, нет. Однако при разработке этого руководства я создал несколько дополнительных примеров, а затем удалил их,

поэтому дата доступа каталога отражает этот факт. Мы поговорим об этом чуть позже, когда будем обсуждать поиск файлов.

Сортировка вывода

По умолчанию команда ls выводит файлы в алфавитном порядке. Для сортировки вывода существует множество опций. Например, ls -t отсортирует по дате модификации (от самой последней до самой старой), в то время как ls -lS отсортирует список по размеру (от самых больших к маленьким). Добавление -r приводит к сортировке в обратном порядке. Например, используем ls -lrt, чтобы вывести длинный список, отсортированный в обратном порядке по дате модификации. За подробностями обратитесь к man-страницам.

Копирование, перемещение и удаление

Мы научились создавать файлы, но предположим, что хотим сделать их копию или же переименовать их, или переместить в другое место файловой системы, или же удалить совсем. Для этих целей мы будем использовать три команды.

cp
: используется для копирования одного или более файлов. Вы должны передать ей по крайней мере два аргумента, один (или более) источников, а второй имя цели. Если вы определите два имени файла, то первый будет скопирован во второй. Как источник, так и цель могут включать в себя пути. Если вы определите каталог как последнее имя, то можете определить множество файлов, которое будет в него скопировано. Все файлы будут скопированы из существующих месторасположений, а копии будут иметь те же имена, что и исходные файлы. Заметим, что здесь нет никаких предположений по умолчанию о том, является ли цель текущим каталогом как в операционных системах DOS и Windows.

mv
: используется для перемещения или переименования одного или более файлов или каталогов. В общем случае имена, которые вы можете использовать, следуют тем же правилам, что и для копирования с помощью команды cp; вы можете переименовать один файл или переместить набор файлов в новый каталог. Так как имя это только запись в каталоге, которая связана с inode, то вас не должно удивлять, что значение inode не меняется за исключением, когда файл перемещается в другую файловую систему, в этом случае перемещение похоже на копирование, за которым следует удаление оригинала.

rm
: используется для удаления одного или нескольких файлов. Мы коротко рассмотрим, как удалять каталоги.

Листинг 46 иллюстрирует применение cp и mv для создания резервных копий текстовых файлов. Мы также используем ls -i чтобы показать inode некоторых файлов.

1. Сначала скопируем text1 в text1.bkp.
2. Затем создадим подкаталог с помощью команды mkdir
3. Затем создадим вторую копию text1, на этот раз в созданном каталоге и покажем, что все три файла имеют разные inode.
4. Затем мы перемещаем text1.bkp в созданный каталог и после этого переименовываем. Хотя мы могли бы сделать с помощью одной команды, мы

для примера используем две.

5. Мы проверяем снова значения inode, чтобы подтвердить, что text1.bkp с inode 2129019 больше не находится в каталоге lpi103, но, что это inode файла text1.bkp.1 в каталоге для резервных копий.

Листинг 46. Копирование и перемещение файлов

```
[ian@echidna lpi103]$ cp text1 text1.bkp
[ian@echidna lpi103]$ mkdir backup
[ian@echidna lpi103]$ cp text1 backup/text1.bkp.2
[ian@echidna lpi103]$ ls -i text1 text1.bkp backup
2128984 text1  2129019 text1.bkp

backup:
1564497 text1.bkp.2
[ian@echidna lpi103]$ mv text1.bkp backup
[ian@echidna lpi103]$ mv backup/text1.bkp backup/text1.bkp.1
[ian@echidna lpi103]$ ls -i text1 text1.bkp backup
ls: text1.bkp: No such file or directory
2128984 text1

backup:
2129019 text1.bkp.1  1564497 text1.bkp.2
```

Обычно cp перезапишет существующий файл, если он существует и в него можно писать. С другой стороны команда mv не переместит или переименует файл, если цель существует. Существует несколько опций, позволяющих менять поведение cp и mv.

-f или --force
: заставит cp попытаться удалить существующую цель, если в нее нельзя записывать.

-i или --interactive
: попросит интерактивно подтвердить попытку замещения существующего файла

-b или --backup
: сделает резервную копию файлов, которые будут замещены.

Как обычно обращайтесь к man-страницам за более подробной информацией.

В Листинге 47 мы продемонстрируем копирование с резервированием, а затем удаление файлов.

Листинг 47. Резервирование копий и удаление файлов

```
[ian@echidna lpi103]$ cp text2 backup
[ian@echidna lpi103]$ cp --backup=t text2 backup
[ian@echidna lpi103]$ ls backup
text1.bkp.1  text1.bkp.2  text2  text2.~1~
[ian@echidna lpi103]$ rm backup/text2 backup/text2.~1~
[ian@echidna lpi103]$ ls backup
text1.bkp.1  text1.bkp.2
```

Заметим, что команда rm также принимает опции -i и -f. Как только вы удалил файл с помощью rm, у файловой системы к нему больше нет доступа. На некоторых системах по умолчанию есть псевдоним alias rm='rm -i' для пользователя root, чтобы помочь предотвратить случайное удаление файла. Это также полезная мысль, если вы боитесь случайно что-либо удалить.

Прежде чем мы закончим обсуждение, следует сказать, что команда cp по умолчанию создает новую дату создания для новых файлов. Владелец и группа нового файла такие же, как и у оригинала. Опция -p может использоваться для сохранения выбранных атрибутов. Заметим, что только root может сохранять права владения. Смотри man-страницы для получения подробной информации.

Mkdir и rmdir

Мы уже видели, как создать каталог с помощью mkdir.Теперь пойдем дальше и рассмотрим команду для удаления каталогов rmdir.

Mkdir

Положим, вы находитесь в каталоге lpi103 и хотите создать подкаталоги dir1 и dir2. Команда mkdir,как и другие рассмотренные команды, может создать несколько каталогов за один раз как показано в Листинге 48.

Листинг 48. Создание нескольких каталогов

```
[ian@echidna lpi103]$ mkdir dir1 dir2
```

Команда не выводит подтверждения об успешном выполнении, но вы можете использовать echo $?, чтобы проверить, что код выхода действительно 0.

Если вы хотите создать вложенные каталоги, как например d1/d2/d3, то ничего не выйдет, так ни d1 ни d2 не существуют. К счастью у mkdir есть опция -p, которая позволяет создать любое число родительских каталогов. Листинг 49 иллюстрирует это.

Листинг 49. Создание родительских каталогов

```
[ian@echidna lpi103]$ mkdir d1/d2/d3
mkdir: cannot create directory `d1/d2/d3': No such file or directory
[ian@echidna lpi103]$ echo $?
1
[ian@echidna lpi103]$ mkdir -p d1/d2/d3
[ian@echidna lpi103]$ echo $?
0
```

Rmdir

Удаление каталогов происходит с помощью команды rmdir. Снова, существует опция -p, позволяющая удалять родительские каталоги. С помощью rmdir вы можете удалить каталоги, только если они пустые. Мы рассмотрим другой способ сделать

это, когда будем рассматривать рекурсивное манипулирование. Один раз, выучив как использовать команду rmdir, вы вряд ли будете часто применять ее в командной строке, но знать о ней, тем не менее, стоит.

Чтобы проиллюстрировать удаление каталогов, мы скопировали файл text1 в каталог d1/d2, так что теперь он больше не пуст. Затем мы использовали rmdir, чтобы удалить все каталоги, которые мы создали с помощью mkdir. Как вы видите, d1 и d2 не удаляться, потому как d2 не пуст. Другие же каталоги были удалены. Как только мы удалим копию text1 из d2, мы сможем удалить d1 и d2 простой командой rmdir -p.

Листинг 50. Удаление каталогов

```
[ian@echidna lpi103]$ cp text1 d1/d2
[ian@echidna lpi103]$ rmdir -p d1/d2/d3 dir1 dir2
rmdir: `d1/d2': Directory not empty
[ian@echidna lpi103]$ ls . d1/d2
.:
backup  sedtab  text2  text4  text6  xab  yab
d1      text1   text3  text5  xaa    yaa

d1/d2:
text1
[ian@echidna lpi103]$ rm d1/d2/text1
[ian@echidna lpi103]$ rmdir -p d1/d2
```

Рекурсивное манипулирование

В оставшихся нескольких частях этого раздела мы рассмотрим различные операции обработки над множеством файлов, а также рекурсивном применении команд к дереву каталогов.

Рекурсивный просмотр

У команды ls есть опция -R (заглавная буква 'R') для просмотра списка каталога и его подкаталогов. Опция рекурсии применима только к именам каталогов; она не будет искать все файлы, положим 'text1', в дереве каталогов. Мы можете использовать другие опции, которые мы рассмотрели совместно с -R. Рекурсивный список каталога lpi103, включая значения inode, показан в Листинге 51.

Листинг 51. Рекурсивный просмотр каталогов

```
[ian@echidna lpi103]$ ls -iR ~/lpi103
/home/ian/lpi103:
1564496 backup  2128985 text2  2128982 text5  2128987 xab
2128991 sedtab  2128990 text3  2128995 text6  2128988 yaa
2128984 text1   2128992 text4  2128986 xaa    2128989 yab

/home/ian/lpi103/backup:
2129019 text1.bkp.1  1564497 text1.bkp.2
```

Рекурсивное копирование

Вы можете использовать опцию -r (или -R или --recursive), чтобы сообщить команде cp о прохождении по всему дереву каталога источника и рекурсивно скопировать его содержимое. Чтобы предотвратить бесконечную рекурсию, сам исходный каталог может быть не скопирован. Листинг 52 показывает, как скопировать все в нашем каталоге lpi103 в подкаталог copy1. Мы используем ls -R, чтобы показать полученное дерево каталогов.

Листинг 52. Рекурсивное копирование

```
[ian@echidna lpi103]$ cp -pR . copy1
cp: cannot copy a directory, `.', into itself, `copy1'
[ian@echidna lpi103]$ ls -R
.:
backup  sedtab  text2  text4  text6  xab  yab
copy1   text1   text3  text5  xaa    yaa

./backup:
text1.bkp.1  text1.bkp.2

./copy1:
backup  text1  text3  text5  xaa  yaa
sedtab  text2  text4  text6  xab  yab

./copy1/backup:
text1.bkp.1  text1.bkp.2
```

Рекурсивное удаление

Мы упомянули ранее, что команда rmdir удаляет только пустые каталоги. Мы можем использовать опцию -r (или -R или --recursive), чтобы заставить команду rm удалить как файлы так и каталоги, как показано в Листинге 53, где мы удаляем каталог copy1, который только что создали, а также его содержимое, включая каталог с резервными копиями и его содержимое.

Листинг 53. Рекурсивное удаление

```
[ian@echidna lpi103]$ rm -r copy1
[ian@echidna lpi103]$ ls -R
.:
backup  text1  text3  text5  xaa  yaa
sedtab  text2  text4  text6  xab  yab

./backup:
text1.bkp.1  text1.bkp.2
```

Если файлы не доступны вам для записи, то вы можете добавить опцию -f, чтобы принудительно удалить их. Так часто поступает пользователь root, при чистке системы, но будьте внимательны, так как можете потерять ценные данные, если будете невнимательны.

Шаблоны и подстановки

Часто вам необходимо произвести единую операцию нам множеством объектов файловой системы, не оперируя с деревом, как мы поступали с рекурсивными операциями. Например, вы захотите найти время модификации всех текстовых файлов, которые мы создали в каталоге lpi103, не выводя разделенных файлов. Хотя это легко сделать в нашем небольшом каталоге, это задача становится непосильной в большой файловой системе.

Чтобы решить эту проблему, используйте поддержку шаблонов, которая встроена в bash. Эта поддержка, также называемая "globbing" (потому что изначально она была реализована программой /etc/glob), позволит вам определить множество файлов с помощью шаблона.

Строка, содержащая любой из символов '?', '*' или '[', является шаблоном. Подстановка это процесс, в котором интерпретатор (или возможно другая программа) заменяет эти шаблоны на список путей, соответствующих шаблону. Соответствие осуществляется следующим образом.

?

означает один любой символ

*****

соответствует любой строке, включая пустую строку.

[

представляет класс символов. Класс символов это непустая строка, завершенная ']'. Соответствие означает совпадение с любым символом, заключенным в скобках. Есть несколько особых случаев.

- Символы '*' и '?' означают сами себя. Если вы используете их в именах файлах, то вам следует заботиться об использовании кавычек или escape-последовательностей.
- Так как строка должная быть непустой, и завершатся знаком ']', то вы должны помещать символ ']' первым в строке, если хотите найти его соответствие.
- Символ '-' между двумя другими означает диапазон, который включает как эти два символа, так и все те, что находятся между ними. Например, [0-9a-fA-F] представляет собой множество шестнадцатеричных цифр, написанных в верхнем или нижнем регистрах. Чтобы найти соответствие для '-', вы можете поместить его первым или последним в диапазоне.
- Символ '!' означает первый символ диапазона, который дополняет диапазон таким образом, что они не пересекаются. Например, [!0-9] означает любой символ кроме чисел от 0 до 9. Символ '!' соответствует себе, если он не стоит в первой позиции. Помните, что '!' также используется в истории интерпретатора, поэтому будьте внимательны.

Подстановка применяется отдельно к каждому компоненту имени пути. Вы не можете использовать '/' для совпадения или включения его в диапазон. Вы можете использовать его в любом месте, чтобы определить множество файлов или имен каталогов, например, в командах ls, cp, mv или rm. В Листинге 54, мы сначала создаем несколько файлов со странными именами, а затем используем команды ls и rm с шаблонами.

Листинг 54. Примеры использования шаблонов

```
[ian@echidna lpi103]$ echo odd1>'text[*?!1]'
[ian@echidna lpi103]$ echo odd2>'text[2*?!]'
[ian@echidna lpi103]$ ls
backup  text1       text2       text3  text5  xaa  yaa
sedtab  text[*?!1]  text[2*?!]  text4  text6  xab  yab
[ian@echidna lpi103]$ ls text[2-4]
text2  text3  text4
[ian@echidna lpi103]$ ls text[!2-4]
text1  text5  text6
[ian@echidna lpi103]$ ls text*[2-4]*
text2  text[2*?!]  text3  text4
[ian@echidna lpi103]$ ls text*[!2-4]* # Surprise!
text1  text[*?!1]  text[2*?!]  text5  text6
[ian@echidna lpi103]$ ls text*[!2-4] # More surprise!
text1  text[*?!1]  text[2*?!]  text5  text6
[ian@echidna lpi103]$ echo text*>text10
[ian@echidna lpi103]$ ls *\!*
text[*?!1]  text[2*?!]
[ian@echidna lpi103]$ ls *[x\!]*
text1       text2       text3  text5  xaa
text[*?!1]  text[2*?!]  text4  text6  xab
[ian@echidna lpi103]$ ls *[y\!]*
text[*?!1]  text[2*?!]  yaa  yab
[ian@echidna lpi103]$ ls tex?[[]*
text[*?!1]  text[2*?!]
[ian@echidna lpi103]$ rm tex?[[]*
[ian@echidna lpi103]$ ls *b*
sedtab  xab  yab

backup:
text1.bkp.1  text1.bkp.2
[ian@echidna lpi103]$ ls backup/*2
backup/text1.bkp.2
[ian@echidna lpi103]$ ls -d .*
.  ..
```

Примечания:

1. Дополнение совместно с '*' может привести к неожиданным сюрпризам. Шаблон '*[!2-4]' соответствует длиннейшей части имени, за которым не следуют символы 2, 3 или 4, что соответствует как text[*?!1] так и text[2*?!]. Теперь оба сюрприза станут понятны.
2. Что касается ранних примеров ls,то если подстановка является именем каталога, а опция -d не указана, тогда будет выдано содержимое этого каталога (как в примере выше для шаблона '*b*').
3. Если имя файла начинается с точки (.), то необходимо явно его указать для поиска соответствия. Заметим, что только последняя команда ls отобразила две специальных записи (. и ..).

Помните, что любая последовательность шаблонных символов может быть интерпретирована интерпретатором и привести к неожиданным результатам. Более того, если вы определите шаблон, который не соответствует никаким объектам в файловой системе, то POSIX требует, чтобы этот шаблон был передан в команду. Иллюстрируем это в Листинге 55. Некоторые ранние версии передавали пустой

список в команду, поэтому вы можете столкнуться со старыми сценариями, которые ведут себя необычно. Проиллюстрируем это в Листинге 55.

Листинг 55. Сюрпризы при использовании шаблонов

```
[ian@echidna lpi103]$ echo text*
text1 text2 text3 text4 text5 text6
[ian@echidna lpi103]$ echo "text*"
text*
[ian@echidna lpi103]$ echo text[[\!?]z??
text[[!?]z??
```

Более подробно смотрите в man 7 glob. Вам требуется именно этот раздел, так как есть информация о glob в разделе 3. Лучший способ понять теорию это практика, поэтому используйте при любом случае шаблоны. Не забывайте проверять с помощью ls, что шаблон делает то, что нужно, до того как применять cp, mv или rm и не получать неожиданных результатов.

Использование touch

Сейчас рассмотрим команду touch, которая может изменить время доступа и модификации или же создать пустой файл. В следующей части мы рассмотрим, как использовать эту информацию для поиска файлов и каталогов. Мы будем использовать каталог lpi103, который создали ранее в этом руководстве.

touch

Команда touch без опций принимает один или более файлов как параметры и изменяет дату модификации файлов. Обычно это то время, которое отображается при длинном формате вывода списка каталогов. В Листинге 56 мы используем старого друга echo для создания простого файла f1, а затем используем длинный формат вывода списка каталога для отображения времени модификации (или mtime). В этом случае текущее время и будет временем создания файла. Затем мы используем команду sleep, чтобы подождать 60 секунд и снова запустим ls. Заметим, что время у файла изменилось на минуту.

Листинг 56. Изменение времени модификации с помощью touch

```
[ian@echidna lpi103]$ echo xxx>f1; ls -l f1; sleep 60; touch f1; ls -l f1
-rw-rw-r--    1 ian      ian             4 Nov  4 15:57 f1
-rw-rw-r--    1 ian      ian             4 Nov  4 15:58 f1
```

Если вы определите имя файла, которого не существует, то touch просто создаст пустой файл, кроме случая, если вы не определите опцию -c или --no-create. Листинг 57 демонстрирует обе этих команды. Заметим, что создается только f2.

Листинг 57. Создание пустых файлов с помощью touch

```
[ian@echidna lpi103]$ touch f2; touch -c f3; ls -l f*
-rw-rw-r--    1 ian      ian             4 Nov  4 15:58 f1
-rw-rw-r--    1 ian      ian             0 Nov  4 16:12 f2
```

Команда touch может также изменить mtime файла на определенную дату и время с помощью опций -d или -t. Опция -d очень гибка в плане понимания различных форматов даты и времени, в то время как опция -t требует по крайней мере формата MMDDhhmm времени, а также опционального указания значений года и секунд. Листинг 58 содержит такие примеры.

Листинг 58. Установка mtime с помощью touch

```
[ian@echidna lpi103]$ touch -t 200511051510.59 f3
[ian@echidna lpi103]$ touch -d 11am f4
[ian@echidna lpi103]$ touch -d "last fortnight" f5
[ian@echidna lpi103]$ touch -d "yesterday 6am" f6
[ian@echidna lpi103]$ touch -d "2 days ago 12:00" f7
[ian@echidna lpi103]$ touch -d "tomorrow 02:00" f8
[ian@echidna lpi103]$ touch -d "5 Nov" f9
[ian@echidna lpi103]$ ls -lrt f*
-rw-rw-r--    1 ian      ian             0 Oct 24 12:32 f5
-rw-rw-r--    1 ian      ian             4 Nov  4 15:58 f1
-rw-rw-r--    1 ian      ian             0 Nov  4 16:12 f2
-rw-rw-r--    1 ian      ian             0 Nov  5 00:00 f9
-rw-rw-r--    1 ian      ian             0 Nov  5 12:00 f7
-rw-rw-r--    1 ian      ian             0 Nov  5 15:10 f3
-rw-rw-r--    1 ian      ian             0 Nov  6 06:00 f6
-rw-rw-r--    1 ian      ian             0 Nov  7 11:00 f4
-rw-rw-r--    1 ian      ian             0 Nov  8  2005 f8
```

Если вы затрудняетесь с определением даты, то можете использовать команду date. У нее также есть опция -d которая понимает те же форматы даты, что и touch.

Вы можете использовать опцию -r (или --reference) вместе с именем файла, чтобы сигнализировать, что touch (или date) следует использовать дату модификации существующего файла. В Листинге 59 содержатся некоторые примеры.

Листинг 59. Использование даты модификации существующего файла

```
[ian@echidna lpi103]$ date
Mon Nov  7 12:40:11 EST 2005
[ian@echidna lpi103]$ date -r f1
Fri Nov  4 15:58:27 EST 2005
[ian@echidna lpi103]$ touch -r f1 f1a
[ian@echidna lpi103]$ ls -l f1*
-rw-rw-r--    1 ian      ian             4 Nov  4 15:58 f1
-rw-rw-r--    1 ian      ian             0 Nov  4 15:58 f1a
```

Система Linux записывает как время модификации файла, так и время доступа. Обе отметки совпадают по значению, когда файл создается, и обе сбрасываются, когда он изменяется. Время доступа изменяется, даже если файл не модифицировался. В нашем последнем примере с touch, мы рассмотрим время доступа. Опция -a (или --time=atime, --time=access или --time=use) определяет, что время доступа должно быть изменено. Листинг 60 использует команду cat, чтобы прочесть файл f1 и отобразить его содержимое. Затем мы используем ls -l и ls -lu, чтобы отобразить

время модификации и доступа соответственно для файлов f1 и f1a, который мы создали, используя f1 как ссылку. Затем мы сбрасываем время доступа f1 до f1a, используя touch -a.

Листинг 60. Время доступа и модификации

```
[ian@echidna lpi103]$ cat f1
xxx
[ian@echidna lpi103]$ ls -lu f1*
-rw-rw-r--    1 ian      ian             4 Nov  7 14:13 f1
-rw-rw-r--    1 ian      ian             0 Nov  4 15:58 f1a
[ian@echidna lpi103]$ ls -l f1*
-rw-rw-r--    1 ian      ian             4 Nov  4 15:58 f1
-rw-rw-r--    1 ian      ian             0 Nov  4 15:58 f1a
[ian@echidna lpi103]$ cat f1
xxx
[ian@echidna lpi103]$ ls -l f1*
-rw-rw-r--    1 ian      ian             4 Nov  4 15:58 f1
-rw-rw-r--    1 ian      ian             0 Nov  4 15:58 f1a
[ian@echidna lpi103]$ ls -lu f1*
-rw-rw-r--    1 ian      ian             4 Nov  7 14:13 f1
-rw-rw-r--    1 ian      ian             0 Nov  4 15:58 f1a
[ian@echidna lpi103]$ touch -a -r f1a f1
[ian@echidna lpi103]$ ls -lu f1*
-rw-rw-r--    1 ian      ian             4 Nov  4 15:58 f1
-rw-rw-r--    1 ian      ian             0 Nov  4 15:58 f1a
```

За более полной информацией о принимаемых форматах времени и даты, смотри man или info страницы команд touch и date.

Поиск файлов

Последней темой этой части руководства будет команда find, которая используется для поиска файлов в одном или более дереве каталогов, на основе таких признаков как имя, дата модификации или размер. Снова, мы будем использовать каталог lpi103, который создали ранее в этом руководстве.

find

Команда find осуществляет поиск файлов или каталогов, используя все имя или его часть, или же другие критерии как размер, тип, владелец файла, дата создания или дата последнего доступа. Простейший поиск осуществляется по имени или его части. Листинг 61 демонстрирует каталог lpi103, в котором мы сначала ищем все файлы, имеющие в своем имени букву '1' или 'k', а затем производим поиск пути, который мы разъясним чуть ниже.

Листинг 61. Поиск файлов по имени

```
[ian@echidna lpi103]$ find . -name "*[1k]*"
./text1
./f1
./backup
./backup/text1.bkp.2
```

```
./backup/text1.bkp.1
./f1a
[ian@echidna lpi103]$ find . -ipath "*ACK*1"
./backup/text1.bkp.1
[ian@echidna lpi103]$ find . -ipath "*ACK*/*1"
./backup/text1.bkp.1
```

Замечания:

1. Шаблоны, которые вы можете использовать такие же, как мы видели в разделе ранее Шаблоны и подстановки.
2. Вы можете использовать -path вместо -name, чтобы находить полные пути, а не просто имена файлов. В этом случае шаблон может изменять компоненты пути.
3. Если вам нужен регистронезависимый поиск, как в примере с использованием ipath, предваряйте опции find, которые осуществляют по строке или шаблону, символом 'i'
4. Если вы хотите найти файл или каталог, чье имя начинается с точки, как например .bashrc или текущий каталог (.), то тогда вы должны определить ведущей точку, как часть шаблона. В противном случае поиск будет игнорировать эти файлы и каталоги.

В первом примере выше мы искали как файлы, так и каталоги (./backup). Используйте параметр -type, а также однобуквенный тип, чтобы ограничить поиск. Используйте 'f' для регулярных файлов, 'd' для каталогов и 'l' для символьных ссылок. Смотри man-страницу find, чтобы узнать о других типах. На Листинге 62 представлен результат поиска в каталогах (-type d).

Листинг 62. Поиск файлов по типу

```
[ian@echidna lpi103]$ find . -type d
.
./backup
[ian@echidna lpi103]$ find . -type d -name "*"
./backup
```

Заметим, что -type d. без указания какой-либо спецификации отобразит список каталогов, у которых в имени ведущая точка (только текущий каталог в этом случае).

Мы также можем искать файлы по размеру, как определенному (n) так и файлы, размер которых больше (+n) или меньше (-n) определенного значения. Используя как верхнюю, так и нижнюю границы, можно найти файлы, чьи размеры попадают в этот диапазон. По умолчанию опция -size команды find предполагает файл из 'b' 512-байт блоков. Среди других, можно выбрать 'c' для байт, 'k' для килобайт. В Листинге 63 мы ищем сначала все файлы размера 0, а затем все файлы, чей размер от 24 до 25 байт. Заметим, что определение -empty вместо -size 0 также заставляет искать пустые файлы.

Листинг 63. Поиск файлов по размеру

```
[ian@echidna lpi103]$ find . -size 0
./f2
./f3
./f4
./f5
./f6
./f7
./f8
./f9
./f1a
[ian@echidna lpi103]$ find . -size -26c -size +23c -print
./text1
./text2
./text5
./backup/text1.bkp.2
./backup/text1.bkp.1
```

В Листинге 63 рассматривается опция -print, которая является примером действия, которое может быть выполнено на результатом поиска. В bash shell это действие выполняется по умолчанию, если не определено иное. В некоторых системах и интерпретаторах действие обязательно, в противном случае вывода не будет.

Другие действия включают в себя -ls, которое печатает информацию о файле, аналогичную команде ls -lids или -exec, которое выполняет команду для каждого файла. Действие -exec должно заканчиваться точкой с запятой в виде escape-последовательности. Также используйте {}, если вы хотите, чтобы возвращаемый файл использовался в команде. Как мы видели раньше, фигурные скобки также интерпретируются, поэтому их надо либо заключать в кавычки, либо писать как escape-последовательность. Листинг 64 показывает как опции -ls и -exec могут использоваться для выдачи информации о файлах.

Листинг 64. Поиск файлов и действия над ними

```
[ian@echidna lpi103]$ find . -size -26c -size +23c -ls
2128984    4 -rw-rw-r--   1 ian    ian      24 Sep 23 12:27 ./text1
2128985    4 -rw-rw-r--   1 ian    ian      25 Sep 23 13:39 ./text2
2128982    4 -rw-rw-r--   1 ian    ian      24 Sep 26 12:46 ./text5
1564497    4 -rw-rw-r--   1 ian    ian      24 Oct  4 09:45
./backup/text1.bkp.2
2129019    4 -rw-rw-r--   1 ian    ian      24 Oct  4 09:43
./backup/text1.bkp.1
[ian@echidna lpi103]$ find . -size -26c -size +23c -exec ls -l '{}' \;
-rw-rw-r--    1 ian    ian      24 Sep 23 12:27 ./text1
-rw-rw-r--    1 ian    ian      25 Sep 23 13:39 ./text2
-rw-rw-r--    1 ian    ian      24 Sep 26 12:46 ./text5
-rw-rw-r--    1 ian    ian      24 Oct  4 09:45 ./backup/text1.bkp.2
-rw-rw-r--    1 ian    ian      24 Oct  4 09:43 ./backup/text1.bkp.1
```

Опция -exec используется для самых различных задач, ограниченных вашей фантазией. Например:

```
find . -empty -exec rm '{}' \;
```

удаляет все пустые файлы в дереве каталогов, в то время как

```
find . -name "*.htm" -exec mv '{}' '{}l' \;
```

переименует все файлы .htm в .html.

В наших последних примерах мы используем отметки времени модификации, описанные в команде touch, чтобы найти все файлы с определенной датой модификации. В Листинге 65 показаны три примера:

1. При использовании -mtime -2 команда find ищет все файлы, которые были модифицированы в течение последних двух дней. День в данном случае это 24 часовой период, отсчитывающийся от текущей даты и времени. Заметим, что вам следует использовать -atime, если вы хотите осуществить поиск файлов на основе времени доступа, а не времени модификации.
2. Указание опции -daystart гарантирует, что мы хотим рассматривать календарные дни, начало которых отсчитывается от полуночи. В этом случае файл f3 в результат не попадает.
3. Наконец, мы покажем, как использовать диапазон в минутах, а не днях, чтобы найти файлы, модифицированные межу одним часом (60 минут) и 10 часами (600 минут) ранее.

Листинг 65. Поиск файлов по временным отметкам

```
[ian@echidna lpi103]$ find . -mtime -2 -type f -exec ls -l '{}' \;
-rw-rw-r--    1 ian      ian             0 Nov  5 15:10 ./f3
-rw-rw-r--    1 ian      ian             0 Nov  7 11:00 ./f4
-rw-rw-r--    1 ian      ian             0 Nov  6 06:00 ./f6
-rw-rw-r--    1 ian      ian             0 Nov  8  2005 ./f8
[ian@echidna lpi103]$ find . -daystart -mtime -2 -type f -exec ls -l '{}'
\;
-rw-rw-r--    1 ian      ian             0 Nov  7 11:00 ./f4
-rw-rw-r--    1 ian      ian             0 Nov  6 06:00 ./f6
-rw-rw-r--    1 ian      ian             0 Nov  8  2005 ./f8
[ian@echidna lpi103]$ find . -mmin -600 -mmin +60 -type f -exec ls -l
'{}' \;
-rw-rw-r--    1 ian      ian             0 Nov  7 11:00 ./f4
```

Man-страницы команды find помогут вам изучить расширенный набор опций, который мы не можем рассмотреть в этом кратком введении.

Потоки, программные каналы и перенаправления

Перенаправление старндартного ввода/вывода

Напомним, что интерпретатор работает с тремя стандартными потоками.

1. stdout это стандартный поток вывода, который обеспечивает вывод команды. Его дескриптор равен 1.
2. stderr это стандартный поток ошибок, который выводит ошибки команд. Его

дескриптор равен 2.
3. stdin это стандартный поток ввода, который обеспечивает ввод командам. Его дескриптор равен 0.

Входные потоки обеспечивают ввод командам, который обычно поступает от клавиш терминала. Потоки вывода печатают символы текста, обычно на терминал. Изначально терминал представлял собой печатную машинку ASCII или же дисплейный терминал, сейчас в основном это окно на рабочем столе графической среды.

В разделе Текстовые потоки и фильтры мы видели, как перенаправлять стандартный вывод в файл или на стандартный ввод другой команды, а также можем перенаправить стандартный ввод из файла или из вывода другой команды.

Перенаправление вывода

Существует два способа перенаправить вывод:

n>

перенапрвляет вывод из файлового дескриптора nв файл. У вас должно быть право записи в файл. Если файла не существует, то он будет создан. Если файл существует, его содержимое будет утеряно без предупреждения.

n>>

также перенаправляет вывод из файлового дескриптора n в файл. Снова, у вас должно быть право на запись в файл. Если файл не существует, то будет создан. Если он существует, то вывод команды добавится к существующему файлу.

Символы nв n> или n>> являются дескрипторами файла. Если его не написать, то по умолчанию предполагается стандартный вывод. В Листинге 66 приведено перенаправление стандартного вывода и стандартного потока ошибок команды ls, используя файлы, созданные ранее в каталоге lpi103. Мы также продемонстрируем добавление вывода в существующий файл.

Листинг 66. Перенаправление вывода

```
[ian@echidna lpi103]$  ls x* z*
ls: z*: No such file or directory
xaa  xab
[ian@echidna lpi103]$ ls x* z* >stdout.txt 2>stderr.txt
[ian@echidna lpi103]$  ls w* y*
ls: w*: No such file or directory
yaa  yab
[ian@echidna lpi103]$ ls w* y* >>stdout.txt 2>>stderr.txt
[ian@echidna lpi103]$ cat stdout.txt
xaa
xab
yaa
yab
[ian@echidna lpi103]$ cat stderr.txt
ls: z*: No such file or directory
ls: w*: No such file or directory
```

Мы сказали, что перенаправление с помощью n> обычно переписывает существующий файл. Вы можете контролировать это с помощью опции noclobber

встроенной команды set. Если она определена, то вы моежете ее подавить с помощью опции n>| как показано в Листинге 67.

Листинг 67. Перенаправление вывода с помощью noclobber

```
[ian@echidna lpi103]$ set -o noclobber
[ian@echidna lpi103]$ ls x* z* >stdout.txt 2>stderr.txt
-bash: stdout.txt: cannot overwrite existing file
[ian@echidna lpi103]$ ls x* z* >|stdout.txt 2>|stderr.txt
[ian@echidna lpi103]$ cat stdout.txt
xaa
xab
[ian@echidna lpi103]$ cat stderr.txt
ls: z*: No such file or directory
[ian@echidna lpi103]$ set +o noclobber #восстанавливанем изначальное
значение параметра noclobber
```

Иногда вам требуется перенаправить как стандартный вывод так поток ошибок в файл. Это часто делается для автоматизированных процессов или фоновых задач, так что вы можете рассмотреть вывод позже. Используйте &> или &>>, чтобы перенаправить стандартный вывод и поток ошибок в одно и тоже место. Другой способ состоит в перенаправлении файлового дескриптора n, а затем перенаправлении файлового дескриптора m в одно и тоже место с помощью конструкции m>&n или m>>&n. Важен порядок, в котором осуществляется перенаправление вывода. Например,

```
command 2>&1 >output.txt
```

не тоже самое, что

```
command >output.txt 2>&1
```

Проиллюстрируем эти концепции в Листинге 68. Заметим, что в последней команде стандартный вывод был перенаправлен после потока ошибок, таким образом стандартный вывод все еще выводится на терминал.

Листинг 68. Перенаправление двух потоков в один файл

```
[ian@echidna lpi103]$ ls x* z* &>output.txt
[ian@echidna lpi103]$ cat output.txt
ls: z*: No such file or directory
xaa
xab
[ian@echidna lpi103]$ ls x* z* >output.txt 2>&1
[ian@echidna lpi103]$ cat output.txt
ls: z*: No such file or directory
xaa
xab
[ian@echidna lpi103]$ ls x* z* 2>&1 >output.txt
ls: z*: No such file or directory
[ian@echidna lpi103]$ cat output.txt
xaa
xab
```

В другие разы вам потребуется проигнорировать полностью стандартный вывод или поток ошибок. В этом случае перенаправьте нужный поток в /dev/null. В Листинге 69 мы показжем как игнорировать поток ошибок команды ls.

Листинг 69. Игнорирование вывода с помощью /dev/null

```
[ian@echidna lpi103]$ ls x* z* 2>/dev/null
xaa  xab
[ian@echidna lpi103]$ cat /dev/null
```

Перенаправление ввода

Также как мы перенаправляли потоки stdout и stderr, вы тоже можете перенаправлять stdin из файла, используя оператор <. Если вы вспомните в обсуждении sort и uniq мы использовали команду tr для замены пробелов в файле text1 на символы табуляции. В том примере мы использовали вывод команды cat, чтобы создать стандартный ввод для команды tr. Вместо бессмысленного вызова cat,мы можем использовать перенаправление ввода для преобразования пробелов символы табуляции, как показано в Листинге 70.

Листинг 70. Перенаправление ввода

```
[ian@echidna lpi103]$ tr ' ' '\t'<text1
1       apple
2       pear
3       banana
```

У интерпретаторов, включая bash, также есть концепция документа, которая является другой формой перенаправления ввода. Она включает в себя использование << и слова, такого как END, в качестве маркера или сигнала конца ввода. Проиллюстрируем это на примере Листинга 71.

Листинг 71. Перенаправление ввода, используя концепцию документа

```
[ian@echidna lpi103]$ sort -k2 <<END
> 1 apple
> 2 pear
> 3 banana
> END
1 apple
3 banana
2 pear
```

Помните, как мы создали файл text2 в Листинге 23? Вы можете удивиться, почему бы просто не набрать sort -k2, ввести свои данные, а затем нажать **Ctrl-d**, чтобы сигнализировать окончание ввода. Короткий ответ состоит в том, что вы можете, но в таком случае вы не узнали бы о концепции документа. В действительности, документы очень часто используются в сценариях (которые рассмотрены в

руководстве по теме 109 об интерпретаторах, сценариях, программировании и компиляции). В сценарии нет другого способа сигнализировать о том, какие строки необходимо воспринимать как ввод. Так как сценарии интенсивно используют табуляцию для выравнивания информации, то существует другой прием работы с документами. Если вы используете <<- вместо <<, тогда ведущие символы табуляции будут удалены. В Листинге 72 мы использовали подобную технику для создания символа табуляции, который затем использовали в Листинге 42. Затем мы создадим очень маленький сценарий, содержащий две команды cat каждый из которых будет считывать из документа. Наконец, мы используем команду . (точку), чтобы обнаружить сценарий в текущем каталоге и запустить его.

Листинг 72. Перенаправление ввода с помощью документа

```
[ian@echidna lpi103]$ ht=$(echo -en "\t")
[ian@echidna lpi103]$ cat<<END>ex-here.sh
> cat <<-EOF
> apple
> EOF
> ${ht}cat <<-EOF
> ${ht}pear
> ${ht}EOF
> END
[ian@echidna lpi103]$ cat ex-here.sh
cat <<-EOF
apple
EOF
        cat <<-EOF
        pear
        EOF
[ian@echidna lpi103]$ . ex-here.sh
apple
pear
```

Конвейеры

В разделе Текстовые потоки и фильтры мы обсуждали фильтрацию текста, как процесс взятия входного потока, совершения преобразования над текстом и отсылки его в выходной поток. Мы также сказали, что фильтрация часто осуществляется с помощью конвейера команд, в котором выход одной команды соединяется или перенаправляется на вход другой команды. Подобное использование конвейров не ограничивается только текстовыми потоками, хотя именно для они используются чаще всего.

Соединение stdout с stdin

Как мы уже видели, мы используем оператор | (конвейера) между двумя командами для перенаправления stdout первой команды на stdin второй команды. Мы конструируем длинные конвейеры из нескольких команд в Листинге 73.

Листинг 73. Конвейер из нескольких команд

```
command1 | command2 | command3
```

Следует заметить, что конвейеры перанаправляют только stdout на stdin. Вы не можете использовать 2|, чтобы перенаправить stderr, по крайней мере, используя наши сейчас знания. Если stderr был перенаправлен на stdout, тогда оба потока будут соеденены. Иллюстрируем этот подход в Листинге 74, в котором используем конвейер для сортировки сообщений об ошибках и нормальных сообщений команды ls с четырьмя шаблонами, расположенными не по алфавиту.

Листинг 74. Конвейер из двух выходных потоков

```
[ian@echidna lpi103]$ ls y* x* z* u* q*  2>&1 |sort
ls: q*: No such file or directory
ls: u*: No such file or directory
ls: z*: No such file or directory
xaa
xab
yaa
yab
```

У любой из команд могут быть опции или аргументы. Многие команды используют дефис (-) вместо имени файла как аргумент, чтобы сообщить, что ввод будет из stdin, а не из файла. Подробнее смотрите в man-страницах. Конструирование конвейера из команд, каждая из которых решает свою задачу, составляет философию решения задач в Linux UNIX.

Одним из преимуществ конвейеров в системах Linux и UNIX является то, что в отличие от других популярных операционных систем, конвейеры не используют промежуточных файлов. Stdout первой команды не пишется в файл, который затем считывается второй командой. Если ваша конкретная версия tar не поддерживает разжатие файлов с помощью bzip2,не беспокойтесь. Как мы видели в руководстве по Теме 102, вы можете просто использовать конвейер как, например

```
bunzip2 -c drgeo-1.1.0.tar.bz2 | tar -xvf -
```

Вывод в качестве аргументов

В разделе Использование командной строки мы изучили подстановку команд и как использовать вывод одной команды как часть другой. В предыдущем разделе Управление файлами мы узнали как использовать опцию -i команды find, чтобы использовать вывод команды find как ввод для другой команды. В Листинге 75 представлено три способа отображения содержимого файлов text1 и text2.

Листинг 75. Использование вывода как аргументов с помощью подстановки команды и find -exec

```
[ian@echidna lpi103]$ cat `ls text[12]`
1 apple
2 pear
3 banana
9       plum
3       banana
10      apple
```

```
[ian@echidna lpi103]$ cat $(find . -name "text[12]")
1 apple
2 pear
3 banana
9       plum
3       banana
10      apple
[ian@echidna lpi103]$ find . -name "text[12]" -exec cat '{}' \;
1 apple
2 pear
3 banana
9       plum
3       banana
10      apple
```

Приведенные примеры работают, но с ограничениями. Положим, что у вас есть файл, содержащий разделитель (пробел в этом случае). Посмотрите Листинг 76 и попробуйте понять, что делает каждая команда, прежде чем читать дальше.

Листинг 76. Использование вывода в качестве аргументов с помощью подстановки команды и find -exec

```
[ian@echidna lpi103]$ echo grapes>"text sample2"
[ian@echidna lpi103]$ cat `ls text*le2`
cat: text: No such file or directory
cat: sample2: No such file or directory
[ian@echidna lpi103]$ cat "`ls text*le2`"
grapes
[ian@echidna lpi103]$ cat "`ls text*2`"
cat: text2
text sample2: No such file or directory
```

Вот, что мы делали.
Мы создали файл "text sample2", содержащий одну строку со словом "grapes"

- Мы попытались использовать подстановку команды, чтобы отобразить содержимое "text sample2". Нас постигла неудача, так как bash передал два параметра команде cat, а именно text и sample2.
- Будучи умнее bash, мы заключили значения команды подстановки в кавычки. Это сработало
- Наконец, мы заменили шаблон, и вывод представляет собой странную ошибку. Здесь получилось так, что bash передал команде cat один параметр, который эквивалентен строке, являющей результатом

  ```
  echo -e "text2\ntext sample2"
  ```

  Если это кажется странным, попробуйте сами!

Что нам требуется так это способ выделения имен файлов вне зависимости от количества в них слов. Мы не упомянули ранее, что когда вывод команды как, например ls, используется в конвейере или команде подстановке, то он считывается построчно. Один способ состоит в использовании встроенной команды read в цикле

со встроенной командой while. Хотя это и находится за рамками этого руководства, мы покажем данное решение.

Листинг 77. Использование while и read в цикле

```
[ian@echidna lpi103]$ ls text*2 | while read l; do cat "$l";done
9       plum
3       banana
10      apple
grapes
```

xargs

Большую часть времени мы будем обрабатывать списки файлов, поэтому нам требуется средства их создания и обработки. К счастью, у команды find есть опция -print0, которая разделяет строки своего вывода символом конца строки, а не символом новой строки. Такие команды как tar и xargs содержат опцию -0 (или --null), которая позволяет им понимать такой тип параметров. Мы уже встречались с tar. Команда xargs работает почти как опция -exec команды find, однако существует большая разница между ними, которую мы увидим. Давайте посмотрим пример.

Листинг 78. Использование xargs с -0

```
[ian@echidna lpi103]$ find . -name "text*2" -print0 |xargs -0 cat
9       plum
3       banana
10      apple
1 apple
2 pear
3 banana
grapes
```

Заметим, что теперь мы перенаправили вывод из find в xargs. Вам не надо использовать точку с запятой в конце команды и, по умолчанию, xargs добавляет аргументы к командной строке. Однако система выдала 7 строк вместо ожидаемых четырех. Что пошло не так?

снова о find

Мы можем использовать команду wc для проверки того, что всего в двух файлах, вывод которых мы ожидали, четыре строки . Корень проблемы лежит в том, что find ведет поиск в каталоге с резервными копиями, в котором она находит backup/text1.bkp.2, соответствующий нашему шаблону. Чтобы решить проблему, воспользуемся опцией -maxdepth команды find, чтобы ограничить глубину поиска до текущего каталога. Есть также опция -mindepth, которая позволит еще более уточнить поиск. В Листинге 79 приведено полное решение.

Листинг 79. Ограничение find

```
[ian@echidna lpi103]$ ls text*2
text2  text sample2
[ian@echidna lpi103]$ wc text*2
      3       6      25 text2
      1       1       7 text sample2
      4       7      32 total
[ian@echidna lpi103]$ find . -name "text*2" -maxdepth 1 -print0 |xargs -0
cat
9       plum
3       banana
10      apple
grapes
```

Подробно о xargs

Существует разница между xargs и find -exec.
Команда xargs по умолчанию передает так много аргументов команде, насколько это возможно. Вы можете ограничить число входных строк с помощью -l или --max-lines за которой следует число. Кроме того, можете использовать -n или --max-args для ограничения количества передаваемых аргументов или -s или --max-chars для ограничения максимального числа символов в строке аргументов. Если ваша команда способна обработать несколько аргументов, то более эффективной будет передача ей как можно большего числа параметров за раз.
Вы можете использовать '{}' как делали для find -exec если определите опцию -i или --replace. Вы можете изменить поведение '{}' по умолчанию для строки, которые сигнализируют место подстановки входного параметра, определив значение для -i. То есть подразумевается -I 1.
Наш последний пример с xargs показан в Листинге 80.

Листинг 80. Примеры xargs

```
[ian@echidna lpi103]$ # передача всех аргументов за раз
[ian@echidna lpi103]$ find . -name "text*2"  |xargs echo
./text2 ./backup/text1.bkp.2 ./text sample2
[ian@echidna lpi103]$ # покажем файлы, которые мы создали раньше с
помощью команды touch
[ian@echidna lpi103]$ ls f[0-n]*|xargs echo
f1 f1a f2 f3 f4 f5 f6 f7 f8 f9
[ian@echidna lpi103]$ # удалим их всех одной строкой
[ian@echidna lpi103]$ ls f[0-n]*|xargs rm
[ian@echidna lpi103]$ # используем строку подстановки
[ian@echidna lpi103]$ find . -name "text*2"  |xargs -i echo - '{}' -
- ./text2 -
- ./backup/text1.bkp.2 -
- ./text sample2 -
[ian@echidna lpi103]$ # Ограничимся одной строкой вывода на вызов
[ian@echidna lpi103]$ find . -name "text*2"  |xargs -l1 echo
./text2
./backup/text1.bkp.2
./text sample2
[ian@echidna lpi103]$ # Ограничимся одним аргументом на вызов
[ian@echidna lpi103]$ find . -name "text*2"  |xargs -n1 echo
./text2
./backup/text1.bkp.2
./text
sample2
```

Заметим, что мы не использовали здесь -print0. Объясняет ли это последний пример в Листинге 80?

Разделение вывода

В этом разделе рассмотрим еще одну команду. Иногда вам понадобится просмотреть вывод на экране и сохранить его в файл для последующего использования. Хотя вы можете это сделать, перенаправив вывод в файл в одном окне, а затем использовать tail -fn1, чтобы увидеть текст на другом экране, использования команды tee гораздо проще.

Вы можете использовать tee в конвейере. Ее аргументы это файл (или файлы), в которые необходимо скопировать стандартный вывод. Опция -a добавляет, а не переписывает файлы. Как мы видели ранее в этом разделе при обсуждении конвейеров, вам понадобиться перенаправить stderr в stdout, до того как соединить его с tee в случае, если вам требуется оба потока. В Листинге 81 показано использование tee для сохранения вывода двух файлов f1 и f2.

Листинг 81. Разделение stdout с помощью tee

```
[ian@echidna lpi103]$ ls text[1-3]|tee f1 f2
text1
text2
text3
[ian@echidna lpi103]$ cat f1
text1
text2
text3
[ian@echidna lpi103]$ cat f2
text1
text2
text3
```

Создание, отслеживание и уничтожение процессов

Приоритетные и фоновые задачи

Когда вы выполняете команду в терминальном окне, как мы делали до этого, то вы запускали ее в приоритетном режиме. Наши команды работали довольно быстро, но предположим, что вы в графической среде и хотите запустить на рабочем столе цифровые часы. Не будем учитывать тот факт, что в большинстве сред они уже есть; мы просто рассматриваем как пример.

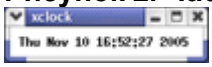
Если вы работаете в системе X Window, то возможно у вас есть такие утилиты как xclock или xeyes. Мы будем использовать xclock. В man-странице сказано, что вы можете запустить цифровые часы с помощью команды

```
xclock -d -update 1
```

Часть -update 1 является запросом на обновление часов раз в секунду, иначе стрелки часов обновлялись бы раз в минуту. Давайте запустим ее в терминальном

окне. Мы увидим картину как на Рисунке Figure 2, а терминал будет выглядеть как в Листинге 82. Если у вас нет xclock или системы X Window, мы покажем, как создать в терминале простые часы, чтобы вы могли выполнить упражнения.

Рисунок 2. Часы xclock



Листинг 82. Работающие часы xclock

```
[ian@echidna ian]$ xclock -d -update 1
```

К сожалению, терминальное окно больше не отображает приглашение, но нам надо как-то вернуться. К счастью, в Bash есть клавиша приостановки Ctrl-z. Нажав эту комбинацию, вы снова видите приглашение как показано в Листинге 83.

Листинг 83. Приостановление xclock с помощью Ctrl-z

```
[ian@echidna ian]$ xclock -d -update 1

[1]+  Stopped                 xclock -d -update 1
[ian@echidna ian]$
```

Часы все еще на рабочем столе, но они не работают. Произошло приостановление. На самом деле, если вы перетащите на него другое окно, но оно даже не перерисуется. Вы также видите, что на терминале отобразилось сообщение "[1]+ Stopped". В сообщении 1 это номер задачи. Вы можете перезапустить часы, набрав fg %1. Вы также можете использовать имя команды или часть ее, набрав fg %xclock или fg %?clo. Наконец, вы можете просто использовать fg без параметров для перезапуска самой последней задачи, job 1 в нашем случае. Перезапуск с помощью fg также влечет возврат задачи в приоритетный режим, поэтому вы не видите приглашения. Что вам необходимо сделать, так это поместить задачу в фон; команда bg принимает те же параметры, что и команда fg и делает тоже самое. В Листинге 84 показано, как вернуть в приоритетный режим xclock и приостановить ее, используя две формы команды fg.. Вы можете снова ее приостановить и поместить в фон; часы будут работать, а вы сможете продолжать работу в терминале.

Листинг 84. Помещение xclock в фоновый режим работы

```
[ian@echidna ian]$ fg %1
xclock -d -update 1

[1]+  Stopped                 xclock -d -update 1
[ian@echidna ian]$ fg %?clo
xclock -d -update 1

[1]+  Stopped                 xclock -d -update 1
[ian@echidna ian]$ bg
[1]+ xclock -d -update 1 &
[ian@echidna ian]$
```

Использование "&"

Вы заметили, что, когда поместили xclock в фон, сообщение больше не гласило "Stopped" и что оно было завершено символом амперсанда (&). На самом деле, вам не надо приостанавливать процесс, чтобы помещать его в фон. Вы можете просто добавить в команду знак амперсанда, и shell интерпретатор запустит команду (или список команд) в фоном режиме. Давайте запустим аналоговые часы таким способом. Вы увидите часы как на Рисунке 3, а терминал будет выглядеть как в Листинге 85.

Рисунок 3. Аналоговые часы xclock

Листинг 85. Запуск xclock в фоне с помощью &

```
[ian@echidna ian]$ xclock -bg wheat -hd red -update 1&
[2] 5659
```

Заметим, что на этот раз сообщение немного отличается. Оно содержит номер задачи и идентификатор процесса (PID). Мы рассмотрим PID и его статус немного позже. Сейчас рассмотрим команду jobs для поиска запущенных задач. Добавим опцию -l, чтобы посмотреть PID процессов, и увидим, что задача 2 имеет PID 5659 как показано в Листинге 86. Заметим также, что у задачи 2 стоит знак плюс (+) перед номером, сигнализирующий, что это текущая задача. Эта задача также перейдет в приоритетный режим, если не будет задана никакая работа команде fg.

Листинг 86. Отображение работы и информации о процессе

```
[ian@echidna ian]$ jobs -l
[1]-  4234 Running                 xclock -d -update 1 &
[2]+  5659 Running                 xclock -bg wheat -hd red -update 1 &
```

Прежде, чем будем рассматривать другие вопросы, связанные с задачами, реализуем сами цифровые часы. Мы будем использовать команду sleep для создания задержки на две секунды, а затем используем команду date для выдачи текущей даты и времени. Мы вставим эти команды в цикл while, используя блок do/done для создания бесконечного цикла. Наконец, мы используем круглые скобки для создания списка команд и переведем весь список в фоновой режим с помощью амперсанда.

Листинг 87. Цифровые часы

```
[ian@echidna ian]$ (while sleep 2; do date;done) &
[1] 16291
[ian@echidna ian]$ Thu Nov 10 22:58:02 EST 2005
Thu Nov 10 22:58:04 EST 2005
Thu Nov 10 22:58:06 EST 2005
Thu Nov 10 22:58:08 EST 2005
fThu Nov 10 22:58:10 EST 2005
Thu Nov 10 22:58:12 EST 2005
gThu Nov 10 22:58:14 EST 2005

( while sleep 2; do
    date;
done )
Thu Nov 10 22:58:16 EST 2005
Thu Nov 10 22:58:18 EST 2005
```

Как и ожидали, наш список выполняется как задача 1 с PID 16291. Каждые две секунды команда date выдает на терминал время и текущую дату. Ваш ввод выделен шрифтом. При медленном наборе символы будут рассеяны по экрану, прежде чем будет набрана вся команда. На самом деле заметим, что 'f' 'g', которые мы напечатали для выдачи списка в приоритетный режим, находятся на двух строках. После того, как мы ввели команду fg, bash отобразит команду, которая сейчас работает в интерпретаторе, а именно, список команд, который продолжает работать каждые две секунды.

Как только мы перевели задачу в приоритетный режим, то можем как завершить задачу (или убить), так и предпринять какое-либо другое действие. В этом случае мы используем Ctrl-c для завершения работы наших часов.

Стандартный ввод/вывод и фоновые процессы

Вывод команды date в предыдущем примере рассеян вместе с символами команды fg, которую мы пытались набрать. Встает интересный вопрос. Что случится с процессом, если ему требуется ввод из stdin?

Терминальный процесс, в котором мы запустили фоновое приложение, называется контролирующим терминалом. В случае не использования перенаправления, потоки stdout и stderr фонового процесса направляются на контролирующий терминал. Также фоновая задача ожидает ввод от контролирующего терминала, но у контролирующего терминала нет способа передать ваши символы в stdin фонового процесса. В таком случае Bash приостанавливает процесс, так что он больше не исполняется. Вы можете вывести его в приоритетный режим и предоставить необходимый ввод. Листинг 88 иллюстрирует простые примеры, в которых вы переключаете список команд в фоновый режим. Через некоторое время нажимаете Enter и процесс останавливается. Переводите задачу в приоритетный режим, предоставляете строку ввода, за которой следует сигнал окончания ввода Ctrl-d файла. Список команд завершается, и мы отображаем созданный файл.

Листинг 88. Ожидание stdin

```
[ian@echidna ian]$ (date; cat - >bginput.txt; date)&
[1] 18648
[ian@echidna ian]$ Fri Nov 11 00:03:28 EST 2005


[1]+  Stopped                 ( date; cat - >bginput.txt; date )
[ian@echidna ian]$ fg
( date; cat - >ginput.txt; date )
input data
Fri Nov 11 00:03:53 EST 2005
[ian@echidna ian]$ cat bginput.txt
input data
```

Задачи без терминалов

На практике вы возможно захотите осуществлять ввод\вывод для фоновых процессов из или в файлы Встает другой вопрос; что случится с процессом, если контролирующий терминал закроется или пользователь выйдет из системы? Ответ зависит от используемого интерпретатора. Если он посылает сигнал SIGHUP (или зависание), то приложение закроется. Мы коротко рассмотрим сигналы, а сейчас рассмотрим другой способ решения этой проблемы.

nohup

Команда nohup используется для запуска команды, которая будет игнорировать сигналы зависания и добавлять stdout и stderr в файл. По умолчанию это файл nohup.out или $HOME/nohup.out. Если писать в файл нельзя, то команда не запустится. Если вы хотите, то можете перенаправить куда угодно stdout или stderr как мы уже узнали из предыдущего раздела этого руководства.

Еще одной особенностью nohup является то, что она не будет выполнять конвейер или список команд. В теме Перенаправление стандартного ввода\вывода Мы рассмотрели, как использовать сценарии. Вы можете сохранить конвейер или список команд в файл и запустить его с помощью sh (интерпретатором по умолчанию) или командой bash, хотя вы не можете использовать команду . или source как мы делали в примере ранее. В следующем руководстве этой серии (по теме 104, рассказывающем об Устройствах, файловых системах Linux, структуре файловой системы) мы покажем, как сделать сценарий исполняемым, но сейчас мы будем использовать команду sh или bash. Листинг 89 показывает, как это можно сделать с нашими написанными часами. Не стоит и говорить, что запись времени в файл не особенно полезна, кроме того, файл будет расти в размере, поэтому мы установим интервал обновления каждые 30 секунд.

Листинг 89. Использование nohup и сценария

```
[ian@echidna ian]$ echo "while sleep 30; do date;done">pmc.sh
[ian@echidna ian]$ nohup . pmc.sh&
[1] 21700
[ian@echidna ian]$ nohup: appending output to `nohup.out'

[1]+  Exit 126                nohup . pmc.sh
[ian@echidna ian]$ nohup sh pmc.sh&
```

```
[1] 21709
[ian@echidna ian]$ nohup: appending output to `nohup.out'

[ian@echidna ian]$ nohup bash pmc.sh&
[2] 21719
[ian@echidna ian]$ nohup: appending output to `nohup.out'
```

Если отобразим содержимое nohup.out, то увидим, что первая строка отображает причину получения кода выхода 126 в нашей первой попытке выше. Последующие строки являются выводом двух версий pmc.sh, которые сейчас работают в фоне. Это проиллюстрировано в Листинге 90.

Листинг 90. Вывод не прерванных процессов

```
[ian@echidna ian]$ cat nohup.out
/bin/nice: .: Permission denied
Fri Nov 11 15:30:03 EST 2005
Fri Nov 11 15:30:15 EST 2005
Fri Nov 11 15:30:33 EST 2005
Fri Nov 11 15:30:45 EST 2005
Fri Nov 11 15:31:03 EST 2005
```

Сейчас обратим внимание на статус процесса. Если вы планируете передохнуть, то не спешите, так как вам предстоит создать еще две задачи, которые создадут еще большие файлы в вашей системе. Вы можете использовать команду fg, чтобы перевести каждый процесс в приоритетный режим, а затем использовать Ctrl-c для его завершения, и если вы позволите им поработать чуть подольше, то увидим другие способы мониторинга и взаимодействия с ними.

Статус процесса

В предыдущих частях этого раздела мы познакомились с командой jobs и увидели, как использовать ее для просмотра Process ID (или PID) наших задач.

ps

Есть другая команда, команда ps, которая отображает различную информацию о статусе процесса. Помните, что "ps" это акроним от "process status". Команда ps принимает ноль или более номеров PID в качестве аргументов и отображает статус соответствующих процессов. Если использовать команду jobs с опцией -p, то вывод будет представлять собой PID лидера группы процессов каждой задачи. Мы будем использовать вывод этой команды как аргументы команды ps, что и показано в Листинге 91.

Листинг 91. Статус фоновых процессов

```
[ian@echidna ian]$ jobs
[1]-  Running                 nohup sh pmc.sh &
[2]+  Running                 nohup bash pmc.sh &
[ian@echidna ian]$ jobs -p
21709
21719
[ian@echidna ian]$ ps `jobs -p`
  PID TTY      STAT   TIME COMMAND
21709 pts/3    SN     0:00 sh pmc.sh
21719 pts/3    SN     0:00 bash pmc.sh
```

Если используем команду ps без всяких опций, то увидим список процессов, которые контролируются текущим терминалом как показано в Листинге 92.

Листинг 92. Отображение статуса с помощью ps

```
[ian@echidna ian]$ ps
  PID TTY          TIME CMD
20475 pts/3    00:00:00 bash
21709 pts/3    00:00:00 sh
21719 pts/3    00:00:00 bash
21922 pts/3    00:00:00 sleep
21930 pts/3    00:00:00 sleep
21937 pts/3    00:00:00 ps
```

Некоторые опции как -f (full), -j (jobs), и -l (long) позволяет отобразить информацию с нужной точностью. Если мы не определим никаких номеров PID, то существует другая полезная опция --forest, которая отображает команды в виде дерева, показывая порождение процессов. В частности, мы видим, что команды sleep предыдущего листинга являются детьми сценариев, которые мы запустили в фоновом режиме. Если мы запустим эту команду в другой момент времени, то можем увидеть команду date в списке процессов, однако на этом примере трудно увидеть другие расхождения. Проиллюстрируем на примере Листинга 93.

Листинг 93. Расширенная информация о статусе процессов

```
[ian@echidna ian]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
ian      20475 20474  0 15:02 pts/3    00:00:00 -bash
ian      21709 20475  0 15:29 pts/3    00:00:00 sh pmc.sh
ian      21719 20475  0 15:29 pts/3    00:00:00 bash pmc.sh
ian      21945 21709  0 15:34 pts/3    00:00:00 sleep 30
ian      21953 21719  0 15:34 pts/3    00:00:00 sleep 30
ian      21954 20475  0 15:34 pts/3    00:00:00 ps -f
[ian@echidna ian]$ ps -j --forest
  PID  PGID   SID TTY          TIME CMD
20475 20475 20475 pts/3    00:00:00 bash
21709 21709 20475 pts/3    00:00:00 sh
21945 21709 20475 pts/3    00:00:00  \_ sleep
21719 21719 20475 pts/3    00:00:00 bash
21953 21719 20475 pts/3    00:00:00  \_ sleep
21961 21961 20475 pts/3    00:00:00 ps
```

Список других процессов

Команды ps, которые мы использовали в примерах, выдавали список только тех процессов, которые запущены через терминал (обратим внимание на столбец SID во втором примере Листинга 93). Чтобы увидеть все процессы, контролируемые терминалами, используйте опцию -a. Опция -x отобразит процессы без контролирующего терминала, а опция -e отобразит информацию для каждого процесса. В Листинге 94 приведен полный формат всех процессов, контролируемых терминалом.

Листинг 94. Отображение остальных процессов

```
[ian@echidna ian]$ ps -af
UID        PID  PPID  C STIME TTY          TIME CMD
ian       4234 32537  0 Nov10 pts/0    00:00:00 xclock -d -update 1
ian       5659 32537  0 Nov10 pts/0    00:00:00 xclock -bg wheat -hd red
-update
ian      21709 20475  0 15:29 pts/3    00:00:00 sh pmc.sh
ian      21719 20475  0 15:29 pts/3    00:00:00 bash pmc.sh
ian      21969 21709  0 15:35 pts/3    00:00:00 sleep 30
ian      21977 21719  0 15:35 pts/3    00:00:00 sleep 30
ian      21978 20475  0 15:35 pts/3    00:00:00 ps -af
```

Заметим, что этот список включает два процесса xclock, которые мы запустили ранее в графическом режиме этой системы (о чем свидетельствует pts/0), в то время как оставшиеся процессы ассоциированы с ssh (Secure Shell) соединением (pts/3 в этом случае).

Существует множество опций у команды ps, которые позволяют отобразить нужную информацию. Например, опции, позволяющие отобразить процессы конкретного пользователя. За подробностями обращайтесь к man-страницам команды ps или же можете получить краткое описание, используя команду ps --help.

top

Если вы используете команду ps несколько раз к ряду, чтобы увидеть различные изменения, то возможно вам стоит вместо нее использовать команду top. Она выводит постоянно обновляющийся список процессов и некоторую полезную информацию. Смотри man-страницы top, чтобы узнать о списке опций, а также о том, как сортировать процессы по объему используемой памяти или другим критериям. В Листинге 95 показано несколько первых строк вывода команды top.

Листинг 95. Вывод других процессов

```
  3:37pm  up 46 days,  5:11,  2 users,  load average: 0.01, 0.17, 0.19
96 processes: 94 sleeping, 1 running, 0 zombie, 1 stopped
CPU states:  0.1% user,  1.0% system,  0.0% nice,  0.9% idle
Mem:  1030268K av,  933956K used,   96312K free,       0K shrd,  119428K
buff
Swap: 1052216K av,    1176K used, 1051040K free                 355156K
cached
```

```
  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
22069 ian       17   0  1104 1104   848 R     0.9  0.1   0:00 top
    1 root       8   0   500  480   444 S     0.0  0.0   0:04 init
    2 root       9   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root       9   0     0    0     0 SW    0.0  0.0   0:00 kapmd
    4 root      19  19     0    0     0 SWN   0.0  0.0   0:00
ksoftirqd_CPU0
    5 root       9   0     0    0     0 SW    0.0  0.0   0:00 kswapd
```

Сигналы

Рассмотрим теперь сигналы в Linux. Они являются асинхронным средством взаимодействия с процессами. Мы уже упомянули сигнал SIGHUP, а также использовали комбинации Ctrl-c и Ctrl-z для посылки сигнала процессам. Главный способ посылки сигнала состоит в использовании команды kill.

Посылка сигналов с помощью kill

Команда kill посылает сигнал определенной задаче или процессу. Листинг 96 содержит пример использования сигналов SIGTSTP и SIGCONT для остановки и возобновления фоновой задачи. Использование сигнала SIGTSTP эквивалентно использованию команды fg для перевода задачи в приоритетный режим, а затем Ctrl-z для ее приостановки. Действие сигнала SIGCONT похоже на работу команды bg.

Листинг 96. Остановка и запуск фоновых задач

```
[ian@echidna ian]$ kill -s SIGTSTP %1
[ian@echidna ian]$ jobs -l
[1]+ 21709 Stopped                 nohup sh pmc.sh
[2]- 21719 Running                 nohup bash pmc.sh &
[ian@echidna ian]$ kill -s SIGCONT %1
[ian@echidna ian]$ jobs -l
[1]+ 21709 Running                 nohup sh pmc.sh &
[2]- 21719 Running                 nohup bash pmc.sh &
```

В этом примере мы использовали номер задачи (%1), но вы также можете посылать сигналы по идентификатору процесса (то есть 21709 является PID задачи %1). Если вы используете команду tail в то время как задача %1 остановлена, то только один процесс изменит файл nohup.out.

Существует всевозможное множество сигналов в вашей системе, которые вы можете отобразить с помощью команды kill -l. Некоторые используются для сообщения об ошибках, например для сообщения о неверных кодах операции, исключения при работе с плавающей точкой или попытке обратится к памяти другого процесса. Заметим, что у сигналов есть как номер, например, 20, так и имя, например, SIGTSTP. Вы можете использовать как номер, так и имя с помощью опции -s. Следует всегда проверять номера сигналов на системе, прежде чем делать какие-либо предположения о сигналах.

Обработчики сигналов и завершение процесса

Мы уже видели, что Ctrl-c завершает процесс. На самом деле она посылает сигнал SIGINT (или прерывание) процессу. Если вы используете kill без указания сигнала, то система пошлет сигнал SIGTERM. Для большинства применений эти два сигнала эквивалентны.

Мы сказали, что команда nohup иммунизирует процесс от восприятия сигнала SIGHUP. В общем случае процесс может реализовать обработчик сигнала для перехвата сигналов. Поэтому процесс может реализовать обработчик сигнала для перехвата как SIGINT так и SIGTERM. Так как обработчик сигнала знает, какой сигнал был послан, он может, например, проигнорировать сигнал SIGINT и завершить работу только при получении сигнала SIGTERM. В Листинге 97 показано, как послать сигнал SIGTERM задаче %1. Заметим, что статус процесса изменился на "Завершен" сразу после того, как мы послали сигнал. Статус изменится на "Прерван", если мы пошлем сигнал SIGINT. Через несколько мгновений произойдет очистка процессов, и теперь задача больше на находится в списке задач.

Листинг 97. Завершение процесса с помощью SIGTERM

```
[ian@echidna ian]$ kill -s SIGTERM %1
[ian@echidna ian]$ jobs -l
[1]  21709 Terminated              nohup sh pmc.sh
[2]- 21719 Running                 nohup bash pmc.sh &
[ian@echidna ian]$ jobs -l
[2]+ 21719 Running                 nohup bash pmc.sh &
```

Обработчики сигналов предоставляют процессу гибкость в том, что он может выполнять свою обычную работу и прерываться по сигналу только для особых целей. Кроме того, что процесс может перехватить запросы на окончание работы и возможно предпринять определенные действия, как например закрытие файлов или проверить работу текущей транзакции, сигналы часто используются, чтобы сообщить демону о его перезапуске и повторном прочтении файла конфигурации. Вы можете сделать это с процессом inetd, чтобы ваши изменения параметров сети вступили в силу или послать сигнал демону печати (lpd), когда добавляете новый принтер.

Безаппеляционное завершение процессов

Некоторые сигналы не могут быть перехвачены, как например некоторые аппаратные исключения. SIGKILL, который скорее всего вы будете использовать, нельзя отловить обработчиком сигналов, и он используется для завершения процесса. В общем случае его следует использовать только, когда другие средства завершения работы процесса не помогают.

Logout и nohup

Помните, мы говорили, что команда nohup позволит всем нашим процессам продолжать работу после нашего выхода из системы. Давайте сделаем это, а затем снова войдем в систему. После нашего возвращения проверим статус процесса, исполняющего наши написанные часы с помощью jobs и ps как мы уже делали до

этого. Вывод представлен в Листинге 98.

Листинг 98. Повторный заход в систему

```
[ian@echidna ian]$ jobs
[ian@echidna ian]$ ps -a
  PID TTY          TIME CMD
 4234 pts/0    00:00:00 xclock
 5659 pts/0    00:00:00 xclock
27217 pts/4    00:00:00 ps
```

Мы видим, что работаем в этот раз на pts/4, но наших задач нет и следа, как будто мы только запустили команду ps и два процесса xclock в графическом режиме с терминала (pts/0). Не совсем то, что мы ожидали увидеть. Однако не все потеряно. В Листинге 99 мы покажем один способ как найти потерянные задачи с помощью опции -s означающей идентификатор сессии, а также идентификатор сессии 20475, который мы видели в Листинге 93. Подумайте о других способах обнаружения задач для случая, если вы не знаете идентификатора сессии.

Листинг 99. Повторный заход в систему

```
[ian@echidna ian]$ ps -js 20475
  PID  PGID   SID TTY          TIME CMD
21719 21719 20475 ?        00:00:00 bash
27335 21719 20475 ?        00:00:00 sleep
```

Зная о том, как убивать процессы, вы можете убить их, используя PID и команду kill.

Приоритеты исполнения процесса

Приоритеты

Как мы уже видели в предыдущем разделе, Linux, как и большинство современных операционных систем выполняет множество процессов. Это достигается путем разделения CPU и других ресурсов всеми процессами. Если некоторый процесс может использовать 100% ресурсов CPU, то другие процессы могут перестать отвечать на запросы и вообще что-то делать. Когда мы рассматривали Статус процесса в предыдущем разделе, то видели, что вывод по умолчанию команды top выдает список процессов, расположенных в порядке убывания потребления ресурсов CPU. Если мы запустим наши часы и команду top, то вряд ли увидим этот процесс в списке, потому как большую часть времени он не использует ресурсы CPU.

В вашей системе могут быть команды, которые могут использовать несколько CPU. Это такие программы как видео-редакторы, программы преобразования изображений или же кодирования звука, как например mp3 в ogg.

Мы создадим небольшой сценарий, который использует CPU и делает немного больше. Он принимает два параметра, счетчик и метку. Он выводит метку, текущую

дату и время, затем уменьшает счетчик до тех пор, пока не достигнет 0, затем снова печатает метку и дату. Этот сценарий не проверяет ошибки, но зато он подходит для иллюстрации.

Листинг 100. Скрипт, интенсивно использующий CPU

```
[ian@echidna ian]$ echo 'x="$1"'>count1.sh
[ian@echidna ian]$ echo 'echo "$2" $(date)'>>count1.sh
[ian@echidna ian]$ echo 'while [ $x -gt 0 ]; do let x=$x-
1;done'>>count1.sh
[ian@echidna ian]$ echo 'echo "$2" $(date)'>>count1.sh
[ian@echidna ian]$ cat count1.sh
x="$1"
echo "$2" $(date)
while [ $x -gt 0 ]; do let x=$x-1;done
echo "$2" $(date)
```

Если вы запустили этот сценарий на своей системе, то можете увидеть результат как в Листинге 101. Этот сценарий интенсивно использует CPU, как мы скоро увидим. Если вы используете не свою рабочую станцию, убедить что вам можно использовать ресурсы CPU .

Листинг 101. Запуск count1.sh

```
[ian@echidna ian]$ sh count1.sh 10000 A
A Mon Nov 14 07:14:04 EST 2005
A Mon Nov 14 07:14:05 EST 2005
[ian@echidna ian]$ sh count1.sh 99000 A
A Mon Nov 14 07:14:26 EST 2005
A Mon Nov 14 07:14:32 EST 2005
```

Пока все хорошо. Давайте теперь используем полученные ранее знания и создадим список команд, давайте запустим сценарий в фоновом режиме работы и используем команду top, чтобы увидеть количество ресурсов CPU, потребляемое сценарием. Список команд показан в Листинге 102, а вывод команды top в Листинге 103.

Листинг 102. Запуск count1.sh и top

```
[ian@echidna ian]$ (sh count1.sh 99000 A&);top
```

Листинг 103. Интенсивное использование CPU

```
  7:20am  up 48 days, 20:54,  2 users,  load average: 0.05, 0.05, 0.00
91 processes: 88 sleeping, 3 running, 0 zombie, 0 stopped
CPU states:  0.1% user,  0.0% system,  0.0% nice,  0.9% idle
Mem:  1030268K av, 1002864K used,   27404K free,       0K shrd,  240336K
buff
Swap: 1052216K av,  118500K used,  933716K free                  605152K
cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 8684 ian       20   0  1044 1044   932 R    98.4  0.1   0:01 sh
```

Неплохо. С помощью простого сценария мы заняли 98.4% ресурсов CPU.

Отображение и установка приоритетов

Если мы выполняем большую задачу, то можем обнаружить, что она влияет на нашу возможность (или же возможности других пользователей) выполнения других задач в системе. Системы Linux и UNIX используют систему приоритетов из 40 значений, начиная от -20 (наивысший приоритет) и заканчивая 19 (низший приоритет).

nice

Процессы обычных пользователей обычно имеют нулевой приоритет. Команда nice отобразит наш приоритет по умолчанию. Команда ps также может отображать приоритет (nice, или NI, уровень), например с помощью опции -l. Проиллюстрируем на примере Листинга 104, в которым выделен наш приоритет в виде значения 0.

Листинг 104. Отображение информации о приоритетах

```
[ian@echidna ian]$ nice
0
[ian@echidna ian]$ ps -l
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME
CMD
000 S   500  7283  7282  0  70   0    -  1103 wait4  pts/2    00:00:00
bash
000 R   500  9578  7283  0  72   0    -   784 -      pts/2    00:00:00 ps
```

Команда nice также может использоваться для запуска процесса с другим приоритетом. Вы можете использовать опцию -n или (--adjustment) вместе с положительным числом, чтобы увеличить приоритет или отрицательное число, чтобы уменьшить его. Помните, что процесс с наименьшим значением приоритета работает чаще всех, поэтому считайте, что увеличение значения приоритета означает для процесса быть более дружелюбным по отношению к другим процессам. Заметим, что вам обычно требуются права суперпользователя (root), чтобы применять отрицательные значения. Другими словами, обычные пользователи могут только сделать свои процессы более дружелюбными. В Листинге 105, мы запустим две копии сценария count1.sh в фоновом режиме с разными приоритетами исполнения. Заметим, что между окончанием их работы появилась задержка в 5 секунд. Попытайтесь поэкспериментировать с разными значениями nice, или же запустить с другим значением приоритета, чтобы увидеть возможную разницу.

Листинг 105. Использование nice для установки приоритетов

```
[ian@echidna ian]$ (sh count1.sh 99000 A&);\
> (nice -n 19 sh count1.sh 99000 B&);\
> sleep 2;ps -l;sleep 20
B Mon Nov 14 08:17:36 EST 2005
A Mon Nov 14 08:17:36 EST 2005
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME
CMD
000 S   500  7283  7282  0  70   0    -  1104 wait4  pts/2    00:00:00
bash
```

```
000 R   500 10765     1 84  80   0    -  1033 -      pts/2    00:00:01 sh
000 R   500 10767     1 14  79  19    -  1033 -      pts/2    00:00:00 sh
000 R   500 10771  7283  0  72   0    -   784 -      pts/2    00:00:00 ps
A Mon Nov 14 08:17:43 EST 2005
B Mon Nov 14 08:17:48 EST 2005
```

Заметим также, что, как и в команде nohup вы не можете использовать список команд или конвейер как аргумент nice.

Изменение приоритетов

renice

Если вы запустили процесс и поняли, что он должен работать с другим приоритетом, то существует способ изменить приоритет работающего процесса с помощью команды renice. Вы указываете абсолютный приоритет (а не величину изменения) процесса или процессов, приоритет которых хотите изменить. Смотри Листинг 106.

Листинг 106. Использование renice для изменения приоритетов

```
[ian@echidna ian]$ sh count1.sh 299000 A&
[1] 11322
[ian@echidna ian]$ A Mon Nov 14 08:30:29 EST 2005

[ian@echidna ian]$ renice +1  11322;ps -l
11322: old priority 0, new priority 1
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME
CMD
000 S   500  7283  7282  0  75   0    -  1104 wait4  pts/2    00:00:00
bash
000 R   500 11322  7283 96  77   1    -  1032 -      pts/2    00:00:11 sh
000 R   500 11331  7283  0  76   0    -   786 -      pts/2    00:00:00 ps
[ian@echidna ian]$ renice +3  11322;ps -l
11322: old priority 1, new priority 3
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME
CMD
000 S   500  7283  7282  0  75   0    -  1104 wait4  pts/2    00:00:00
bash
000 R   500 11322  7283 93  76   3    -  1032 -      pts/2    00:00:16 sh
000 R   500 11339  7283  0  76   0    -   785 -      pts/2    00:00:00 ps
```

Больше информации о командах nice и renice вы можете получить из man-страниц.

Поиск с помощью регулярных выражений

Регулярные выражения

Регулярные выражения впервые появились в теории компьютерных языков. Большинство студентов по computer science учат, что язык, описываемый регулярными выражениями, в точности такой, какой принимает конечный автомат. Регулярные выражения в этом разделе могут нести более сложный смысл, поэтому они не в точности такие же, какие вы изучали на занятиях по информатике, хотя у них одинаковое родство.

Регулярные выражения (также называемые как "regex" или "regexp") представляют способ описания текстовой строки или шаблона таким образом, что программа может осуществлять соответствие шаблона в произвольных текстовых строках, обеспечивая мощные инструменты поиска информации. Утилита grep (от generalized regular expression processor) является стандартной частью инструментария программиста или администратора Linux или UNIX, позволяя использовать регулярные выражения для поиска файлов или вывода команды. В разделе о текстовых потоках и фильтрах мы познакомились с sed или stream editor, который является еще одним стандартным инструментом, использующим регулярные выражения для поиска и замены текста в файлах или текстовых потоках. Этот раздел поможет лучше понять использование регулярных выражений в grep и sed. Другой программой, использующей регулярные выражения, является awk, которая входит в материал экзамена 201 на сертификацию LPIC-2.

Как и по остальным темам этого руководства, по регулярным выражениям и теории языков написано много книг. Смотри разделРесурсы, чтобы узнать о дополнительных источниках информации.

Как только вы узнаете о регулярных выражениях, то увидите сходство между синтаксисом регулярных выражений и шаблонами (или подстановкой), описанными в разделе Шаблоны и подстановки. Сходство это только поверхностно.

Основные строительные блоки

В программе GNU grep, которую можно найти на большинстве Linux систем, используется два синтаксиса регулярных выражений: основной и расширенный. У программы GNU grep нет никаких функциональных отличий. Здесь описан основной синтаксис, а также отличие его от расширенного синтаксиса.

Регулярные выражения состоят из символов и операторов, дополненных метасимволами. Большинство символов соответствует своим значениям, а большинство метасимволов необходимо предварять обратным слешем (\). Основными операторами являются

Конкатенация

Конкатенация соединяет два регулярных выражения. Например, регулярное выражение анайдет соответствие в строке abcdcba дважды (первый и последний символ a) точно также и с регулярным выражением b. Однако ab соответствует только abcdcba, в то время как ba соответствует только abcdcba.

Повторение

Оператор Клини * или повторения соответствует нулю или более появлений

предшествующего регулярного выражения. Поэтому выражение a*b соответствует любой строке из a, завершенной символом b включая самой строке из символа b. Оператор Клини * не надо записывать как escape-последовательность, но в выражении, если вы хотите найти сам символ (*) необходимо писать escape-последовательность.

Выбор

Оператор выбора (|) соответствует либо предшествующему, либо последующему выражению. Его надо писать, как escape-последовательность, если используется основной синтаксис. Например, выражение a*\|b*c найдет строку из любого числа a или любого числа букв b (но не оба), завершенную символом c. Снова простой символ c соответствует такому выражению.

Вам часто потребуется заключать регулярные выражения в кавычки, чтобы избежать подстановки интерпретатором.

В качестве примеров мы будем использовать текстовые файлы, созданные ранее в каталоге lpi103. Рассмотрите простые примеры в Листинге 107. Заметим, что grep принимает регулярное выражение как обязательный параметр и ноль и более файлов, в которых надо осуществить поиск. Если никаких файлов не указано, то grep будет искать в stdin, что делает ее похожей на фильтр, который можно использовать в конвейере. Если соответствия не найдено, то вывода grep не будет, хотя можно проверить ее код выхода.

Листинг 107. Простые регулярные выражения

```
[ian@echidna lpi103]$ grep p text1
1 apple
2 pear
[ian@echidna lpi103]$ grep pea text1
2 pear
[ian@echidna lpi103]$ grep "p*" text1
1 apple
2 pear
3 banana
[ian@echidna lpi103]$ grep "pp*" text1
1 apple
2 pear
[ian@echidna lpi103]$ grep "x" text1
[ian@echidna lpi103]$ grep "x*" text1
1 apple
2 pear
3 banana
[ian@echidna lpi103]$ cat text1 | grep "l\|n"
1 apple
3 banana
[ian@echidna lpi103]$ echo -e "find an\n* here" | grep "\*"
* here
```

Глядя на приведенные примеры, вы иногда можете удивиться полученным результатам, особенно при использовании повторения. Вы, возможно, ожидали, что p* или, по крайней мере pp* соответствует паре символов p, но p* и x* тоже соответствует каждой строке файла, так как оператор * соответствует нулю или большему числу раз предшествующего регулярного выражения.

Первые ярлыки

Вы знаете основные строительные блоки регулярных выражений, давайте рассмотрим некоторые удобные сокращения.

+

Оператор + похож на оператор *, за исключением того, что он соответствует одному или более вхождению предыдущего регулярного выражения. В основном синтаксисе его необходимо писать как escape-последовательность.

?

Символ ? означает ноль или более вхождений предыдущего выражения. Это не тот же знак ?, используемый при подстановках.

.

Метасимвол . (точка) означает любой символ. Одним из наиболее часто используемых шаблонов является .*, который соответствует строке произвольной длины из любых символов (или не содержащей символов совсем). Сравните точку со знаком ?, используемым в подстановке и .* с *, используемым в подстановке.

Листинг 108. Регулярные выражения

```
[ian@echidna lpi103]$ grep "pp\+" text1 # at least to p's
1 apple
[ian@echidna lpi103]$ grep "pl\?e" text1
1 apple
2 pear
[ian@echidna lpi103]$ grep "pl\?e" text1 # pe with optional l between
1 apple
2 pear
[ian@echidna lpi103]$ grep "p.*r" text1 # p, some string then r
2 pear
[ian@echidna lpi103]$ grep "a.." text1 # a followed by two other letters
1 apple
3 banana
```

Соответствие начала или конца строки

Знак ^ (каретки) означает начало строки, в то время как $ (знак доллара) означает конец строки. Поэтому ^..b соответствует любым двум символам в начале строки, за которыми следует b, в то время как ar$ соответствует любой строке, заканчивающейся на ar. Регулярное выражение ^$ соответствует пустой строке.

Более сложные выражения

До сих пор мы рассматривали применение повторения к одному символу. Если вы хотите найти одно или более вхождений строки из нескольких символов как an, которая встречается дважды в banana, используйте круглые скобки, которые надо записывать как escape-последовательности в основном синтаксисе. Также вы можете захотеть осуществить поиск нескольких символов, не используя такой обобщенный оператор как . или длинную последовательность альтернатив. Это можно сделать, заключив альтернативы в квадратные скобки ([]), которые не надо писать как escape-последовательность при использовании основного синтаксиса. Выражения в квадратных скобках составляют класс символов.Кроме нескольких

исключений, о которых мы поговорим позже, использование квадратных скобок позволяет обойтись без использования escape-последовательностей при использовании специальных символов, таких как as . и *.

Листинг 109. Круглые скобки и классы символов

```
[ian@echidna lpi103]$ grep "\(an\)\+" text1 # find at least 1 an
3 banana
[ian@echidna lpi103]$ grep "an\(an\)\+" text1 # find at least 2 an's
3 banana
[ian@echidna lpi103]$ grep "[3p]" text1 # find p or 3
1 apple
2 pear
3 banana
[ian@echidna lpi103]$ echo -e "find an\n* here\nsomewhere." | grep "[.*]"
* here
somewhere.
```

Существует несколько дополнительных возможностей при работе с классами символов.

Диапазон выражения

Диапазон выражения это два символа, разделенных знаком - (дефис), как например 0-9 для цифр или 0-9a-fA-F для шестнадцатеричных чисел. Заметим, что диапазон зависит от локали.

Именованные классы

Несколько именованных классов обеспечивают удобное обозначение для часто используемых классов. Именованные классы начинаются с [: и заканчиваются :]. Некоторые примеры:

[:alnum:]

Цифровые и буквенные символы

[:blank:]

Пробелы и символы табуляции

[:digit:]

Цифры от 0 до 9 (эквивалентно от 0-9)

`[:upper:] и [:lower:]

Соответственно буквы верхнего и нижнего регистра.

^ (отрицание)

Будучи использован на первом месте в квадратных скобках, знак ^ (каретка) дополняет значение остальных символов, так что соответствие происходит, только если (кроме ведущего ^) подстрока не принадлежит классу.

Зная особые значения символов выше, делаем вывод, что если вы хотите обнаружить символ - (дефис) в классе символа, то должны помещать его первым или последним. Если вы хотите обнаружить символ ^ (каретка), то не ставьте его первым. Знак] (правая квадратная скобка) закрывает класс, кроме случая, когда он стоит первым.

Классы символов -- это та область, в которой регулярные выражения и подстановка ведут себя одинаково, хотя отрицание отличается (^ против !). В Листинге 108 приведены примеры классов символов.

Листинг 110. Классы символов

```
[ian@echidna lpi103]$ # Ищет символы от 3 до 7
[ian@echidna lpi103]$ echo -e "123\n456\n789\n0" | grep "[3-7]"
123
456
789
[ian@echidna lpi103]$ # Ищем цифру, за которой до конца строки нет букв n
и r
[ian@echidna lpi103]$ grep "[[:digit:]][^nr]*$" text1
1 apple
```

Использование регулярных выражений совместно с sed

В кратком введении в Sed упоминалось о том, что sed использует регулярные выражения. Regexp могут использоваться как в выражениях адресации, так и в выражениях подстановки. Так, выражение /abc/s/xyz/XYZ/g означает: применить подстановку команды, которая заменит на XYZ все вхождения xyz только в строках, содержащих abc. В Листинге 111 приведено два примера с файлом text1, а в другом мы заменяем последнее слово перед точкой (.) на строку LAST WORD. Заметим, что строка First не изменилась, так как не была предварена пробелом.

Листинг 111. Регулярные выражения в sed

```
[ian@echidna lpi103]$ sed -e '/\(a.*a\)\|\(p.*p\)/s/a/A/g' text1
1 Apple
2 pear
3 bAnAnA
[ian@echidna lpi103]$ sed -e '/^[^lmnXYZ]*$/s/ear/each/g' text1
1 apple
2 peach
3 banana
[ian@echidna lpi103]$ echo "First. A phrase. This is a sentence." |\
> sed -e 's/ [^ ]*\./ LAST WORD./g'
First. A LAST WORD. This is a LAST WORD.
```

Расширенные регулярные выражения

Расширенные регулярные выражения позволяют не писать escape-последовательности нескольких символов, которые приходилось предварять знаком \ в основном синтаксисе, включая круглые скобки, '?', '+', '|', и '{'. Это значит, что они должны быть записаны как escape-последовательность, только если вы хотите, чтобы они интерпретировались как символы. Вы можете использовать опцию -E (или --extended-regexp) команды grep, чтобы сигнализировать об использовании расширенного синтаксиса регулярных выражений. Можно использовать альтернативу в виде egrep.Некоторые старые версии sed не поддерживают расширенные регулярные выражения. Если ваша версия sed поддерживает расширенные regexps, используйте опцию -r, чтобы сообщить sed, что вы будете использовать расширенный синтаксис. В Листинге 112 показан пример, рассмотренный ранее, с использованием расширенной версии egrep.

Листинг 112. Расширенные регулярные выражения

```
[ian@echidna lpi103]$ grep "an\(an\)\+" text1 # find at least 2 an's
3 banana
[ian@echidna lpi103]$ egrep "an(an)+" text1 # find at least 2 an's
3 banana
```

Поиск информации в файлах

Этот раздел завершает некоторые примеры мощных команд grep и find которые позволяют искать информацию в файловой системе. Снова, примеры довольно простые; мы используем файлы, созданные в каталоге lpi103 и его детях.

Для начала grep может искать сразу в нескольких файлах. Если вы добавите опцию -n, то она скажет номера найденных строк. Если вы просто хотите знать количество найденных строк, используйте опцию -c, а если вам нужен список файлов, в которых есть совпадения, используйте опцию -l. В Листинге 113 приведены некоторые примеры.

Листинг 113. Поиск в нескольких файлах

```
[ian@echidna lpi103]$ grep plum *
text2:9 plum
text6:9 plum
text6:9 plum
yaa:9   plum
[ian@echidna lpi103]$ grep -n banana text[1-4]
text1:3:3 banana
text2:2:3       banana
[ian@echidna lpi103]$ grep -c banana text[1-4]
text1:1
text2:1
text3:0
text4:0
[ian@echidna lpi103]$ grep -l pear *
ex-here.sh
nohup.out
text1
text5
text6
xaa
```

Наш последний пример использует команду find, чтобы найти все обычные файлы в текущем каталоге и его детях, а затем использовании xargs для передачи списка файлов команде grep, которая определит число появлений строки banana в каждом файле. Наконец, вывод фильтруется через другой экземпляр grep, на этот раз с опцией -v для поиска всех файлов, которые не содержат ни одного появления искомой строки.

Листинг 114. Поиск файлов, в которых есть хотя бы одно вхождение banana

```
[ian@echidna lpi103]$  find . -type f -print0| xargs -0 grep -c banana|
grep -v ":0$"
./text1:1
./text2:1
./xab:1
./yaa:1
./text5:1
./text6:4
./backup/text1.bkp.2:1
./backup/text1.bkp.1:1
```

В этом разделе мы затронули лишь малую часть того, что вы можете делать с помощью командной строки Linux и регулярных выражений. Более подробную информацию вы можете прочесть в man-страницах.

Редактирование файлов в vi

Использование vi

Редактор vi есть почти в каждой системе Linux и UNIX. На самом деле, если в системе есть только один текстовый редактор, то это наверняка vi, поэтому следует знать как им пользоваться. В этом разделе представлены основные команды vi, а для полного руководства по vi, обратитесь к нашему "введению в vi -- метод шпаргалки" (смотриРесурсы), или же обратитесь к man-страницам или многочисленным книгам.

Запуск vi

Большинство дистрибутивов Linux сейчас поставляется с vim (от VIIMproved) редактором, а не классическим vi. Vim обратно совместим с vi, для которого также доступна графическая оболочка (gvim), а также обычный текстовый режим. Команда vi обычно является псевдонимом или символьной ссылкой на программу. Просмотрите тему Откуда интерпретатор берет команды?, чтобы узнать какая точно команда используется.

Вы можете вспомнить изменение приоритетов, в котором мы пытались изменить приоритет работающего сценария count1.sh. Возможно, вы пытались сделать это сами, но команда выполнялась так быстро, что вы не успевали изменить приоритет с помощью renice. Давайте запустим редактор vi и добавим строку в начало файла, чтобы заснуть на 20 секунд, и у нас появилось время, чтобы изменить приоритеты.

Чтобы запустить редактор vi, используйте команду vi, а также имя файла в качестве параметра. Редактор имеет много опций. За более полной информацией обратитесь к man-страницам. Наберите команду

```
vi count1.sh
```

Вы увидите вывод как в Листинге 115. Если вы используете vim, некоторые слова могут быть подсвечены другим цветом. Vim поддерживает подсветку синтаксиса (она не являлась частью редактора vi), и по умолчанию она может быть включена.

Листинг 115. Редактирование count1.sh в vi

```
x="$1"
echo "$2" $(date)
while [ $x -gt 0 ]; do let x=$x-1;done
echo "$2" $(date)
~
~
~
~
~
~
"count1.sh" 4L, 82C
```

Режимы vi

Редактор vi может работать в двух режимах:

Режим команд
: В режиме команд, вы перемещаетесь по файлу и выполняете такие действия как поиск текста, удаление текста, изменение текста и так далее. Обычно запуск редактора происходит в режиме команд.

Режим вставки
: В режиме вставки вы набираете текст согласно позиции курсора. Чтобы вернуться в режим команд, нажмите Esc (Escape) клавишу.

Эти два режима определяют поведение редактора. Vi датирован временем, когда не все терминальные клавиатуры содержали клавиши перемещения курсора, поэтому вся работа может быть выполнена в vi с помощью обычных клавиш печатной машинки, а также паре специальных клавиш как **Esc** и **Insert**. Однако вы можете настроить vi на использование дополнительных клавиш, если они доступны; большинство клавиш клавиатуры выполняют какую-либо работу в vi. Из-за своего прошлого и медленной работы ранних терминальных соединений, vi заслужил хорошую репутацию за счет использования коротких и непонятных команд.

Выход из vi

Одну из первых вещей, которой я хотел бы выучить в новом редакторе, это как осуществлять выход из программы до того, как начать работать. Следующие способы выхода vi включают сохранение или отмену изменений или перезапуск редактирования файла. Если команды не работают, то возможно, вы находитесь в режиме вставки, поэтому нажмите Esc, чтобы покинуть режим вставки и перейти в режим команд.

:q!
: Выход с отменой всех изменений в файле. Это часто используемая команда, чтобы вернуть все в первоначальный вид.

:w!
: Записать файл (в независимости от того было ли модифицировано

содержимое или нет). Попытка перезаписать существующие файлы или файлы только для чтения или другие не записываемые файлы. Вы можете определить имя файла в качестве параметра, и этот файл будет записан, а не тот с которым вы начали работу. В общем безопаснее пропускать !, кроме случаев, когда вы знаете, что делаете.

ZZ

Записать файл, если он был изменен. Затем произвести выход. Эта команда часто применяется для нормального выхода из vi.

:e!

Редактировать текущую копию файла на диске. Команда перезагрузит файл, отменив созданные вами изменения. Вы также можете использовать команду, если копия на диске была изменена по какой-либо причине и вам требуется последняя ее версия.

:!

Запустить команду интерпретатора. Наберите команду и нажмите Enter. Когда команда завершится, вы увидите ее вывод и приглашение вернуться в редактор vi.

Замечания:

1. Когда вы наберете двоеточие (:), то курсор переместится вниз экрана, где вы можете набирать команду и параметры.
2. Если вы пропустите восклицательный знак в описанных выше командах, то можете получить сообщение об ошибке, как например, о том, что изменения не были сохранены или файл не может быть записан (например, вы редактируете файл только для чтения).
3. У команды : есть длинные формы (:quit, :write, :edit), но они используются редко.

Перемещение

Следующие команды используются для перемещения по файлу:

h

Перейти на один символ влево на текущей строке

j

Перейти на следующую строку

k

Перейти на предыдущую строку

l

Сдвинуться на один знак вправо в текущей строке

w

Перейти к следующему слову на текущей строке

e

Перейти на предыдущее слово в текущей строке

b

Перейти в начало предыдущего слова на текущей строке

Ctrl-f

Пролистнуть страницу вперед

Ctrl-b

Пролистнуть страницу назад

Если вы наберете число перед этими командами, то команда будет исполнена

определенное число раз. Это число называется счетчиком повторений или просто счетчиком. Например, 5h осуществит переход влево на пять символов. Вы можете использовать счетчики повторений со многими командами vi.

Переход по строкам

Следующие команды используются для перехода к определенным строкам вашего файла:

G
: Перейти к определенной строке вашего файла. Например, 3G переходит к строке 3. Без параметров, G переходит к последней строке файла.

H
: Переходит к строке, отстоящей вниз относительно верхнего края экрана. Например, 3H осуществляет переход к третьей строке сверху относительно текущего экрана.

L
: Аналог H, но переход осуществляется относительно нижней части экрана. Так осуществляет на вторую строку относительно нижней части экрана.

Поиск

Вы можете осуществлять поиск в файле с помощью регулярных выражений:

/
: Используйте / и регулярное выражение для поиска вперед по файлу.

?
: Используйте ? и регулярное выражение для поиска по файлу назад.

n
: Используйте n, чтобы повторить последний поиск в любом из направлений.

Вы можете предварять все вышеперечисленные команды числом, означающим счетчик повторений. Так 3/x найдет третье вхождение x относительно текущей позиции, так как и /x за которой следует команда 2n.

Модификация текста

Используйте следующие команды, если вам надо вставить, удалить или изменить текст:

i
: Перейти в режим вставки в текущей позиции. Наберите свой текст и нажмите Esc, чтобы вернуться в режим команд. Используйте I, чтобы начать вставку в начале текущей строки.

a
: Войти в режим вставки после символа в текущей позиции. Наберите свой текст и нажмите Esc, чтобы вернуться в режим команд. Используйте A, чтобы осуществить вставку в конец текущей строки.

c
: Используйте c, чтобы изменить текущий символ и перейти в режим вставки, чтобы набрать замещаемые символы.

o
: Вставить новую строку сразу за текущей строкой. Используйте O, чтобы вставить новую строку сразу над текущей строкой.

cw
: Удалить остаток текущего слова, войти в режим вставки и заменить его. Используйте счетчик повторений, чтобы заменить несколько слов. Используйте c$, чтобы заменить слова до конца строки.

dw
: Тоже, что и cw (и c$) выше, только вход в режим вставки не осуществляется.

dd
: Удалить текущую строку. Используйте счетчик, чтобы удалить несколько строк.

x
: Удалить символ в позиции курсора. Используйте счетчик, чтобы удалить несколько символов.

p
: Вставить последний удаленный текст после текущего символа. Используйте P, чтобы вставить его до текущего символа.

xp
: Комбинация x и p. производит замену символа в позиции курсора и символа справа от него.

Заключение

Мы собрались добавить строку в файл count1.sh. Чтобы сохранить оригинал и сохранить модификацию в count2.sh, мы можем использовать команды vi, после того как открыли файл в vi. Заметим, что <Esc> означает нажать Esc клавишу.

Листинг 116. Команды редактора для добавления строки в count1.sh

```
1G
O
sleep 20<Esc>
:w! count2.sh
:q
```

Просто, когда знаешь как.

В следующем руководстве этой серии рассматривается Тема 104 об Устройствах, файловых систем Linux и Структуре файловой системы (FHS).