Процессы

Процесс Linux — это экземпляр программы, запущенный в памяти.

Запуск xeyes

Для изучения управления процессами, какой-нибудь процесс необходимо сначала запустить. Выполните следующую команду:

```
$ xeyes center red
```

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

Остановка процесса

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

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

```
$ xeyes center red
ControlZ
[1]+  Stopped xeyes center red
$
```

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

fg и bg

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

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

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

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

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

Использование "&"

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

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

Несколько фоновых процессов

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

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

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

Введение в сигналы

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

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

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

SIGTERM и SIGINT

Если вы хотите убить процесс, есть несколько вариантов. По-умолчанию, kill отправляет SIGTERM, который отличается от SIGINT отправляемого по Control-C, но обычно имеет тот же эффект:

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

Полное убийство

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

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

nohup

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

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

Используем ps для вывода списка процессов

Команда jobs, которую мы использовали ранее выводит только те процессы, которые были запущены в вашей сессии bash. Чтобы увидеть все процессы в вашей системе, используйте ps совместно с опциями a и x:

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

Здесь приведены только первые 5 процессов, поскольку обычно список процессов очень длинный. Команда дает вам «слепок» всего, что в данный момент выполняется на машине, однако в нем много лишней информации. Если бы вы, не указали ax, вы бы получили список только тех процессов, которые принадлежат вам, и которые есть в управляющем терминале. Команда ps x покажет все ваши процессы, даже те, которых нет в управляющем терминале. Если использовать ps a, то будет получен список процессов из терминалов всех пользователей.

Просмотр «леса» и «деревьев»

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

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

«u» и «l» опции ps

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

```
$ ps au
USER        PID   %CPU %MEM     VSZ    RSS TTY      STAT    START  TIME COMMAND
agriffis    403   0.0   0.0    2484   72    tty1   S      2001   0:00 bash
chouser     404   0.0   0.0    2508   92    tty2   S      2001   0:00 bash
root        408   0.0   0.0    1308   248   tty6   S      2001   0:00 /sbin/agetty 3
agriffis    434   0.0   0.0    1008   4     tty1   S      2001   0:00 /bin/sh /usr/X
chouser     927   0.0   0.0    2540   96    pts/1  S      2001   0:00 bash

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

Использование top

Если вы обнаружили, что запускаете ps несколько раз подряд, пытаясь рассмотреть происходящие изменения, возможно вам стоит воспользоваться top. Программа top отображает постоянно обновляющийся список процессов, наряду с другой полезной информацией:

```
$ top
10:02pm  up 19 days,  6:24,  8 users,  load average: 0.04, 0.05, 0.00
75 processes: 74 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:          1.3% user,          2.5% system,   0.0% nice,     96.0% idle
Mem:    256020K av,   226580K used,  29440K free,   0K shrd,      3804K buff
Swap:   136544K av,   80256K used,   56288K free          101760K cached

PID   USER        PRI    NI    SIZE  RSS    SHARE  STAT   LIB   %CPU  %MEM   TIME  COMMAND
628   root        16     0     213M  31M    2304   S      0     1.9   12.5   91:43  X
26934 chouser     17     0     1272  1272   1076   R      0     1.1   0.4    0:00   top
652   chouser     11     0     12016 8840   1604   S      0     0.5   3.4    3:52   gnterm
641chouser        9      0     2936  2808   1416   S      0     0.1   1.0    2:13   sawfish
```

nice

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

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

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

renice

Команда nice может изменять приоритет процессов только во время их запуска. Если вам необходимо изменить приоритет работающего процесса, воспользуйтесь командой renice:

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

$ renice 10 641
641: old priority 0, new priority 10

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

Обработка текста

Возвращаемся к перенаправлению

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

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

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

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

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

Пример с каналами (пайпами)

Вот другой простой пример:

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

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

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

Распаковывающий канал

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

```
$ bzip2 d linux2.4.16.tar.bz2
$ tar xvf linux2.4.16.tar
```

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

```
$ bzip2 dc linux2.4.16.tar.bz2 | tar xvf
```

Сжатый тарбол был распакован и мы обошлись без промежуточного файла.

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

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

Буря обработки текста начинается!

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

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

Удобна как первая команда пайпа, например, cat foo.txt | blah.

sort выведет содержимое файла, указанного в командной строке, в алфавитном порядке. Естественно, sort также может принимать ввод из пайпа. Наберите man sort чтобы ознакомиться с опциями команды, которые управлют вариантами сортировки. uniq принимает уже отсортированный файл или поток данных (через пайп) и удаляет повторяющиеся строки.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Буря закончилась! Перенаправление

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

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

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

Использование ">>"

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

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

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

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