Unix shell: первые шаги

Елена Большакова

2018-02-11

Содержание

Зачем и для кого статья?

Эта статья рассчитана на тех, кто не имеет предыдущего опыта работы в unix-овой командной строке, но по тем или иным причинам хочет или должен научиться эффективно с нею взаимодействовать.

Здесь не будет пересказа манов (документации), и статья никак не отменяет и не заменяет их чтение. Вместо этого я расскажу о главных вещах (командах, приемах и принципах), которые надо осознать с самого начала работы в unix shell-е, чтобы работа происходила эффективно и приятно.

Статья касается полноценных unix-подобных окружений, с полнофункциональным шеллом (предпочтительно zsh или bash) и полноценным набором стандартных программ.

Что такое шелл

Shell (шелл, он же "командная строка", он же CLI, он же "консоль", он же "терминал", он же "черное окошко с белыми буковками") – это текстовый интерфейс общения с операционной системой (ну, строго говря, это программа, которая таковой интерфейс обеспечивает, но сейчас это различие несущественно).

В целом работа через шелл выглядит так: пользователь (т.е. вы) с клавиатуры вводит команду, нажимает Enter, система выполняет команду, пишет на экран результат выполнения, и снова ожидает ввода следующей команды. См. также рисунок 1.

Шелл – это основной способ для взаимодействия со всеми Unix-подобными серверными системами.

Где встречаются системы с командной строкой?

Популярные варианты, где вас может поджидать unix-овый шелл:

- MacOS (bash);
- удаленный доступ на сервер по работе или для личного веб-проекта;
- домашний файл-сервер с удаленным доступом;
- Ubuntu, PC-BSD на ноутбуке/десктопе – unix-подобные системы сегодня просты в установке и использовании и получают все большее распространение на личных компьютерах.

Какие задачи разумно решать шеллом?

Естественные задачи, для которых шелл пригоден, полезен и незаменим:

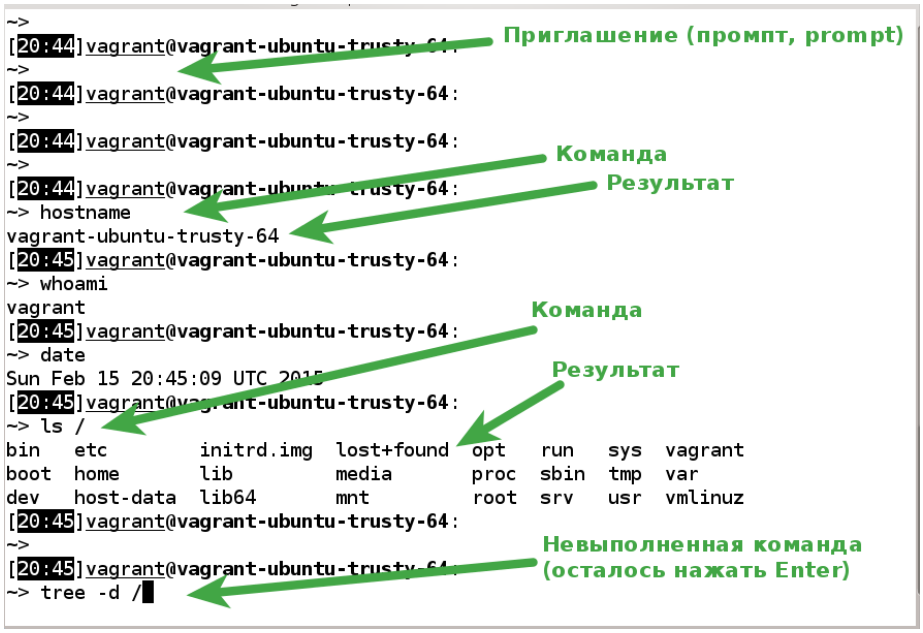

Рис. 1: Типичный вид шелла

- интерактивная работа в терминале:
  - выполнение компиляции, запуск заданий через make;
  - сравнение текстовых файлов;
  - быстрый ad-hoc анализ данных (количество уникальных ip в логе, распределение записей по часам/минутам и т.п.);
  - разовые массовые действия (прибить много процессов; если работаете с системой контроля версий – ревертнуть или зарезолвить кучу файлов);
  - диагностика происходящего в системе (семафоры, локи, процессы, дескрипторы, место на диске и т.п.);
- скриптование:
  - установочные скрипты, для выполнения которых нельзя рассчитывать на наличие других интерпретаторов – это не для новичков;
  - функции для кастомизации интерактивного шелла (влияющие на приглашение, меняющие каталог, устанавливающие переменные окружения) – тоже не совсем для новичков;
  - одноразовые скрипты типа массового перекодирования файлов;
  - makefile-ы.

Абсолютно первые шаги

Начинаем работу: войти и выйти

Убедитесь, что точно знаете, как запустить шелл и как из него выйти.

Если вы работаете за машиной, на которой установлена Ubuntu, вам надо запустить программу Terminal. По окончании работы можно просто закрыть окно.

На MacOS – тоже запустить Terminal.

Для доступа к удаленному серверу – воспользоваться `ssh` (если локально у вас MacOS, Ubuntu или другая unix-like система) или `putty` (если у вас Windows).

Кто я, где я?

Выполните следующие команды:

- `hostname` – выводит имя машины (сервера), на которой вы сейчас находитесь;
- `whoami` – выводит ваш логин (ваше имя в системе);
- `tree -d / |less` – псевдографическое изображение дерева каталогов на машине; выход из пролистывания – `q`;
- `pwd` – выводит каталог, в котором вы сейчас находитесь; в командной строке вы не можете быть "просто так", вы обязательно находитесь в каком-то каталоге (=текущий каталог, рабочий каталог). Вероятно, текущий рабочий каталог выводится у вас в приглашении (prompt).
- `ls` – список файлов в текущем каталоге; `ls /home` – список файлов в указанном каталоге;

История команд (history)

Важное свойство полноценной командной строки – история команд.

Выполните несколько команд: `hostname`, `ls`, `pwd`, `whoami`. Теперь нажмите клавишу "вверх". В строке ввода появилась предыдущая команда. Клавишами "вверх" и "вниз" можно перемещаться вперед и назад по истории. Когда долистаете до `hostname`, нажмите Enter – команда выполнится еще раз.

Команды из истории можно не просто выполнять повторно, а еще и редактировать. Долистайте историю до команды `ls`, добавьте к ней ключ `-l` (получилось `ls -l`, перед минусом пробел есть, а после – нет). Нажмите Enter – выполнится модифицированная команда.

Пролистывание истории, редактирование и повторное выполнение команд – самые типичные действия при работе в командной строке, привыкайте.

Copy-paste

Командная строка очень текстоцентрична: команды – это текст, входные данные для большинства стандартных программ – текст, результат работы – чаще всего тоже текст.

Прекрасной особенностью текста является то, что его можно копировать и вставлять, это верно и для командной строки.

Попробуйте выполнить команду `date +"%y-%m-%d, %A"`

Вводили ли вы ее целиком руками или скопировали из статьи? Убедитесь, что вы можете ее скопировать, вставить в терминал и выполнить.

После того, как научитесь пользоваться `man`'ом, убедитесь, что можете скопировать и выполнить примеры команд из справки. Для проверки найдите в справке по программе `date` раздел `EXAMPLES`, скопируйте и выполните первый приведенный пример (на всякий случай: знак доллара не является частью команды, это условное изображение приглашения к вводу).

Как именно копировать текст из терминала и вставлять его в терминал – зависит от вашей системы и от ее настроек, поэтому дать универсальную инструкцию, к сожалению, не получится. На Ubuntu попробуйте так: копирование – просто выделение мышью, вставка – средняя кнопка мыши. Если не работает, или если у вас другая система – поищите в Интернете или спросите более опытных знакомых.

Ключи и опции

При исследовании истории команд вы уже столкнулись с тем, что у команды `ls` есть по крайней мере два варианта. Если вызвать ее просто так, она выводит простой список:

```
[22:26]akira@latitude-e7240:
~/shell-survival-quide> ls
Makefile  shell-first-steps.md  shell-first-steps.pdf
shell-survival-quide.md  shell-survival-quide.pdf
```

если же добавить ключ `-l`, к каждому файлу выводится подробная информация:

```
[22:28]akira@latitude-e7240:
~/shell-survival-quide> ls -l
total 332
-rw-rw-r-- 1 akira akira    198 Feb 13 11:48 Makefile
-rw-rw-r-- 1 akira akira  15107 Feb 14 22:26 shell-first-steps.md
-rw-rw-r-- 1 akira akira 146226 Feb 13 11:49 shell-first-steps.pdf
-rw-rw-r-- 1 akira akira  16626 Feb 13 11:45 shell-survival-quide.md
-rw-rw-r-- 1 akira akira 146203 Feb 13 11:35 shell-survival-quide.pdf
```

Это очень типичная ситуация: если к вызову команды добавлять специальные модификаторы (ключи, опции, параметры), поведение команды меняется.

Сравните: `tree /` и `tree -d /`, `hostname` и `hostname -f`.

Кроме того, команды могут принимать в качестве параметров имена файлов, каталогов или просто текстовые строки. Попробуйте:

```
ls -ld /home
ls -l /home
grep root /etc/passwd
```

man

`man` – справка по командам и программам, доступным на вашей машине, а также по системным вызовам и стандартной библиотеке C.

Попробуйте: `man grep`, `man atoi`, `man chdir`, `man man`.

Пролистывание вперед и назад делается кнопками “вверх”, “вниз”, “PageUp”, “PageDown”, выход из просмотра справки – кнопкой `q`. Поиск определенного текста в справочной статье: нажимите `/` (прямой слеш), введите текст для поиска, нажимите Enter. Перемещение к следующим вхождениям – клавиша `n`.

Все справочные статьи делятся на категории. Самые важные:

- 1 – исполняемые программы и шелльные команды (`wc`, `ls`, `pwd` и т.п.);
- 2 – системные вызовы (`fork`, `dup2` и т.п.)
- 3 – библиотечные функции (`printf`, `scanf`, `cos`, `exec`).

Указывать, из какой именно категории надо показать справку, нужно в случаях совпадений имен. Например, `man 3 printf` описывает функцию из стандартной библиотеки C, а `man 1 printf` – консольную программу с таким же именем.

Посмотреть список всех доступных на машине справочных статей можно с помощью команды `man -k .` (точка – тоже часть комады).

less

Когда в небольшом окне терминала надо просмотреть очень длинный текст (содержимое какого-то файла, длинный man и т.п.), используют специальные программы-“пейджеры” (от слова page/страница, то есть постраничные листатели). Самый популярный листатель – `less`, и именно он обеспечивает вам пролистывание, когда вы читаете man-ы.

Попробуйте и сравните поведение:

```
cat /etc/bash.bashrc
cat /etc/bash.bashrc |less
```

Можно передать файл в пролистыватель сразу в параметрах:

```
less /etc/bash.bashrc
```

Пролистывание вверхи и вниз – кнопки “вверх”, “вниз”, “PageUp”, “PageDown”, выход – кнопка `q`. Поиск определенного текста: нажимите `/` (прямой слеш), введите текст для поиска, нажимите Enter. Перемещение к следующим вхождениям – клавиша `n`. (Узнаете инструкцию про `man`? Ничего удивительного, для вывода справки тоже используется `less`.)

Права

С любым файлом или каталогом связан набор “прав”: право на чтение файла, право на запись в файл, право исполнять файл. Все пользователи делятся на три категории: владелец файла, группа владельца файла, все прочие пользователи.

Посмотреть права на файл можно с помощью `ls -l`. Например:

```
> ls -l Makefile
-rw-r--r-- 1 akira students    198 Feb 13 11:48 Makefile
```

Этот вывод означает, что владельцу (akira) можно читать и писать файл, группе (students) – только читать, всем прочим пользователя – тоже только читать.

Если при работе вы получаете сообщение `permission denied`, это значит, что у вас недостаточно прав на объект, с которым вы хотели работать.

Подробнее читайте в `man chmod`.

STDIN, STDOUT, конвейеры (пайпы)

С каждой исполняющейся программой связаны 3 стандартных потока данных: поток входных данных `STDIN`, поток выходных данных `STDOUT`, поток для вывода ошибок `STDERR`.

Запустите программу `wc`, введите текст `Good day today`, нажмите Enter, введтие текст `good day`, нажмите Enter, нажмите Ctrl+d. Программа `wc` покажет статистику по количеству букв, слов и строк в вашем тексте и завершится:

```
> wc
good day today
good day
      2       5      24
```

В данном случае вы подали в `STDIN` программы двухстрочный текст, а в `STDOUT` получили три числа.

Теперь запустите команду `head -n3 /etc/passwd`, должно получиться примерно так:

```
> head -n3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

В этом случае программа `head` ничего не читала из `STDIN`, а в `STDOUT` написала три строки.

Можно представить себе так: программа – это труба, в которую втекает `STDIN`, а вытекает `STDOUT`.

Важнейшее свойство юниксовой командной строки состоит в том, что программы-"трубы" можно соединять между собой: выход (`STDOUT`) одной программы передавать в качестве входных данных (`STDIN`) другой программе.

Такая конструкция из соединенных программ называется по-английски pipe (труба), по-русски – конвейер или пайп.

Объединение программ в конвейер делается символом `|` (вертикальная черта)

Выполните команду `head -n3 /etc/passwd |wc`, получится примерно следующее:

```
> head -n3 /etc/passwd |wc
      3       3     117
```

Произошло вот что: программа `head` выдала в `STDOUT` три строки текста, которые сразу же попали на вход программе `wc`, которая в свою очередь подсчитала количество символов, слов и строк в полученном тексте.

В конвейер можно объединять сколько угодно программ. Например, можно добавить к предыдущему конвейеру еще одну программу `wc`, которая подсчитает, сколько слов и букв было в выводе первой `wc`:

```
> head -n3 /etc/passwd |wc |wc
      1       3      24
```

Составление конвейеров (пайпов) – очень частое дело при работе в командной строке. Пример того, как это делается на практике, читайте в разделе "Составление конвейера-однострочника".

Перенаправление ввода-вывода

Вывод (`STDOUT`) програмы можно не только передать другой программе по конвейеру, но и просто записать в файл. Такое перенаправление делается с помощью `>` (знак "больше"):

```
date > /tmp/today.txt
```

В результате выполнения этой команды на диске появится файл `/tmp/today.txt`. Посмотрите его содержимое с помощью `cat /tmp/today.txt`

Если файл с таким именем уже существовал, его старое содержимое будет уничтожено. Если файл не существовал, он будет создан. Каталог, в котором создается файл, должен существовать до выполнения команды.

Если надо не перезаписать файл, а добавить вывод в его конец, используйте `>>`:

```
date >> /tmp/today.txt
```

Проверьте, что теперь записано в файле.

Кроме того, программе можно вместо `STDIN` передать любой файл. Попробуйте:

```
wc </etc/passwd
```

Что делать, когда что-то непонятно

Если вы сталкиваетесь с поведением системы, которое не понимаете, или хотите добиться определенного результата, но не знаете, как именно, советую действовать в следующем порядке (кстати, это относится не только к шеллам):

- насколько возможно четко сформулируйте вопрос или задачу – нет ничего сложнее, чем решать "то, не знаю что";
- вспомните, сталкивались ли вы уже с такой же или подобной проблемой – в этом случае стоит попробовать решение, которое сработало в прошлый раз;
- почитайте подходящие man-ы (если понимаете, какие man-ы подходят в вашем случае) – возможно, вы найдете подходящие примеры использования команд, нужные опции или ссылки на другие команды;
- подумайте: нельзя ли немного поменять задачу? – возможно, чуть-чуть изменив условия, вы получите задачу, которую уже умеете решать;
- задайте свой четко сформулированный вопрос в поисковой системе – возможно, ответ есть на Stack Overflow или других сайтах;

Если ничего из перечисленного не помогло – обратитесь за советом к преподавателю, опытному коллеге или товарищу. И не бойтесь задавать "глупые" вопросы – не стыдно не знать, стыдно не спрашивать.

Если вы разобрались со сложной проблемой (самостоятельно, с помощью Интернета или других людей) – запишите свое решение на случай, если такая же проблема снова возникнет у вас или ваших товарищей. Записывать можно в простой текстовый файл, в Evernote, публиковать в соц.сетях.

Топ полезных команд, конструкций и приемов

Методы работы

Скопировать-и-вставить – из man-ов, из статей на StackOverflow и т.п. Командная строка состоит из текста, пользуйтесь этим: копируйте и используйте примеры команд, записывайте удачные находки на память, публикуйте их в твиттерах и блогах.

Читать man. Nuff said.

Вытащить из истории предыдущую команду, добавить в конвейер еще одну команду, запустить, повторить. См. также раздел “Составление конвейера-однострочника”.

Базовые команды

- переход в другой каталог: `cd`;
- просмотр содержимого файлов: `cat`, `less`, `head`, `tail`;
- манипуляции с файлами: `cp`, `mv`, `rm`;
- просмотр содержимого каталогов: `ls`, `ls -l`, `ls -lS`;
- структура каталогов: `tree`, `tree -d` (можно передать в качестве параметра каталог);
- поиск файлов: `find . -name ...`;

Аналитика

- `wc`, `wc -l`;
- `sort -k` – сортировка по указанному полю;
- `sort -n` – числовая соритровка;
- `diff` – сравнение файлов;
- `grep`, `grep -v`, `grep -w`, `grep '\<word\>'`, `grep -E` – поиск текста;
- `uniq`, `uniq -c` – уникализация строк;
- `awk` – в варианте `awk '{print $1}'`, чтобы оставить только первое поле из каждой строки, `$1` можно менять на `$2`, `$3` и т.д.;

Диагностика системы

- `ps axuww` – информация о процессах (запущенных программах), работающих на машине;
- `top` – интерактивный просмотр самых ресурсоемких процессов;
- `df` – занятое и свободное место на диске;
- `du` – суммарный размер файлов в каталоге (рекурсивно с подкаталогами);
- `strace`, `ktrace` – какие системные вызовы выполняет процесс;
- `lsof` – какие файлы использует процесс;
- `netstat -na`, `netstat -nap` – какие порты и сокеты открыты в системе.

Некоторых программ у вас может не быть, их надо установить дополнительно. Кроме того, некоторые опции этих программ доступны только привилегированным пользователям (`root`'у).

Массовое и полуавтоматическое выполнение

На первых порах пропускайте этот раздел, эти команды и конструкции понадобятся вам тогда, когда доберетесь до несложного шелльного скриптинга.

- `test` – проврека условий;
- `while read` – цикл по строчкам `STDIN`;
- `xargs` – подстановка строк из `STDIN` в параметры указанной программе;
- `seq` – генерация последовательностей натуральных чисел;
- `()` – объединить вывод нескольких команд;
- `;` – выполнить одно за другим;
- `&&` – выполнить при условии успешного завершения первой команды;
- `||` – выполнить при условии неудачного завершения первой команды;
- `tee` – продублировать вывод программы в `STDOUT` и в файл на диске.

Разное

- `date` – текущая дата;
- `curl` – скачивает документ по указаному url и пишет результат на `STDOUT`;
- `touch` – обновить дату модификации файла;
- `kill` – послать процессу сигнал;
- `true` – ничего не делает, возвращает истину, полезна для организации вечных циклов;
- `sudo` – выполнить команду от имени `root`'а.

Составление конвейера-однострочника

Давайте рассмотрим пример реальной задачи: требуется прибить все процессы `task-6-server`, запущенные от имени текущего пользователя.

Шаг 1. Понять, какая программа выдает примерно нужные данные, хотя бы и не в чистом виде. Для нашей задачи стоит получить список всех процессов в системе: `ps axuww`. Запустить.

Шаг 2. Посмотреть на полученные данные глазами, придумать фильтр, который выкинет часть ненужных данных. Часто это `grep` или `grep -v`. Клавишей "Вверх" вытащить из истории предыдущую команду, приписать к ней придуманный фильтр, запустить.

```
ps axuww |grep `whoami`
```

– только процессы текущего пользователя.

Шаг 3. Повторять пункт 2, пока не получатся чистые нужные данные.

```
ps axuww |grep `whoami` | grep '\<task-6-server\>'
```

– все процессы с нужным именем (плюс, может быть, лишние вроде vim task-6-server.c и т.п.),

```
ps axuww |grep `whoami` | grep '\<task-6-server\>' | grep -v vim
ps axuww |grep `whoami` | grep '\<task-6-server\>' | grep -v vim
|grep -v less
```

– только процессы с нужным именем

```
ps axuww |grep `whoami` | grep '\<task-6-server\>' | grep -v vim
|grep -v less |awk '{print $2}'
```

– pid-ы нужных процессов, п. 3 выполнен

Шаг 4. Применить подходящий финальный обработчик. Клавишей "Вверх" вытаскиваем из истории предыдущую команду и добавляем обработку, которая завершит решение задачи:

- `|wc -l` чтобы посчитать количество процессов;
- `>pids` чтобы записать pid-ы в файл;
- `|xargs kill -9` убить процессы.

Задания для тренировки

Хотите попрактиковаться в новых умениях? Попробуйте выполнить следующие задания:

- получите список всех файлов и каталогов в вашем домашнем каталоге;
- получите список всех `man`-статей из категории 2 (системные вызовы);

- посчитайте, сколько раз в man-е по программе `grep` встречается слово grep;
- посчитайте, сколько процессов запущено в данный момент от имени пользователя `root`;
- найдите, какая команда встречается в максимальном количестве категорий справки (man);
- подсчитайте, сколько раз встречается слово var на странице http://ya.ru.

Подсказка: вам понадобится `find`, `grep -o`, `awk '{print $1}'`, регулярные выражения в `grep`, `curl -s`.

Что изучать дальше?

Если командная строка начинает вам нравиться, не останавливайтесь, продолжайте совершенствовать свои навыки.

Вот некоторые программы, которые определенно вам пригодятся, если вы будете жить в командной строке:

- `awk`
- `sed`
- `find` со сложными опциями
- `apropos`
- `locate`
- `telnet`
- `netcat`
- `tcpdump`
- `rsync`
- `screen`
- `ssh`
- `tar`
- `zgrep`, `zless`
- `visudo`
- `crontab -e`
- `sendmail`

Кроме того, со временем стоит освоить какой-нибудь скриптовый язык, например, `perl` или `python`, или даже их оба.

Кому это надо?

А стоит ли вообще изучать сегодня командную строку и шелльный скриптинг? Определенно стоит. Приведу только несколько примеров из требований Facebook к кандидатам, которые хотят поступить на работу в FB.

Data Scientist, Economic Research: Comfort with the command line and with Unix core tools; preferred: adeptness with a scripting language such as Python, or previous software engineering experience.

MySQL Database Engineer: High degree of proficiency in Shell scripting (Bash, Awk, etc); high degree of proficiency in Linux administration.

Manufacturing Quality Engineer, Server: Scripting skills in Bash, Perl, or Python is desirable.

Data Platform Engineer: 2 years experience with Unix/Linux systems.

DevOps Engineer, Data: 2 years experience with Unix/Linux system administration and programming.

Вопросы?

Если у вас есть вопросы по этой статье или вообще по работе в юниксовой командной строке, можете задать их мне по емейлу `e.a.bolshakova <at> yandex.ru`.

Немного полезных и интересных ссылок

- 15 интересных команд Linux
- Survival guide for Unix newbies
- Интересные приемы программирования на Bash
  оригинал: Better Bash Scripting in 15 Minutes
- Shell programming with bash: by example, by counter-example
- Простые способы сделать консольную утилиту удобнее
- Debug your programs like they're closed source!