Информационные технологии

Лекция №3

Оболочка bash

Основные сведения

Оболочка или shell — работа в текстовом режиме (интерфейс командной строки)

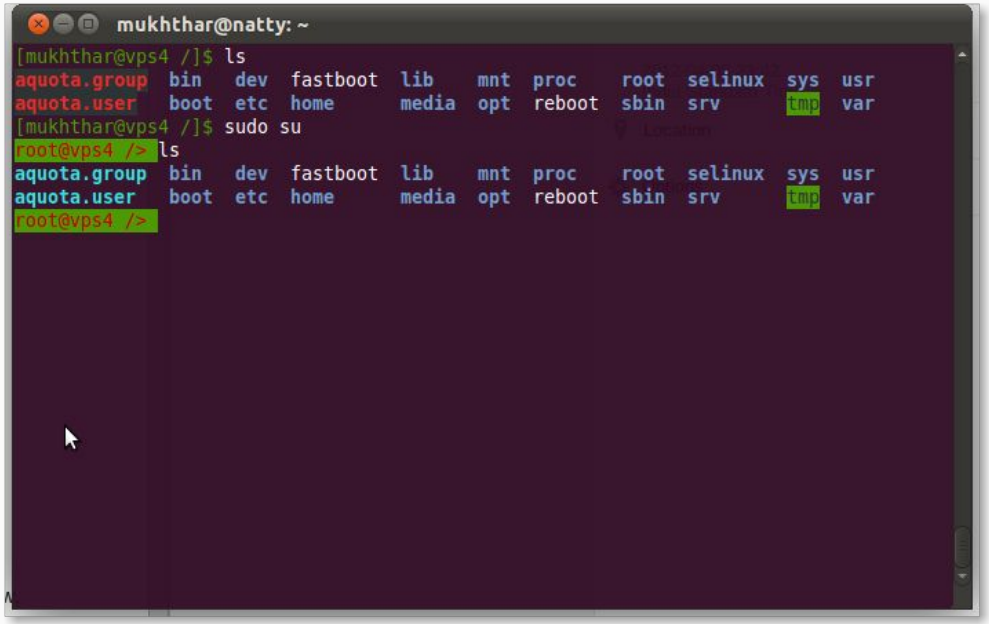
Графический интерфейс пользователя (GUI) — работа в графическом режиме

В командной строке можно запустить любую программу (!)

- Командная строка - это программа
- Sh - так называлась программа раньше (shell)
- bash - Bourne again shell - снова оболочка Борна
- Сама по себе оболочка bash не выполняет никаких прикладных задач
- bash обеспечивает выполнение всех приложений: нахождение вызываемых программ, их запуск и организацию ввода/вывода

```
mars@marsmain ~ $ pwd
/home/mars
mars@marsmain ~ $ cd /usr/portage/app-shells/bash
mars@marsmain /usr/portage/app-shells/bash $ ls -al
total 130
drwxr-xr-x  3 portage portage  1024 Jul 25 10:06 .
drwxr-xr-x 33 portage portage  1024 Aug  7 22:39 ..
-rw-r--r--  1 root    root    35808 Jul 25 10:06 ChangeLog
-rw-r--r--  1 root    root    27002 Jul 25 10:06 Manifest
-rw-r--r--  1 portage portage  4645 Mar 23 21:37 bash-3.1_p17.ebuild
-rw-r--r--  1 portage portage  5977 Mar 23 21:37 bash-3.2_p39.ebuild
-rw-r--r--  1 portage portage  6151 Apr  5 14:37 bash-3.2_p48-r1.ebuild
-rw-r--r--  1 portage portage  5988 Mar 23 21:37 bash-3.2_p48.ebuild
-rw-r--r--  1 portage portage  5643 Apr  5 14:37 bash-4.0_p10-r1.ebuild
-rw-r--r--  1 portage portage  6230 Apr  5 14:37 bash-4.0_p10.ebuild
-rw-r--r--  1 portage portage  5648 Apr 14 05:52 bash-4.0_p17-r1.ebuild
-rw-r--r--  1 portage portage  5532 Apr  8 10:21 bash-4.0_p17.ebuild
-rw-r--r--  1 portage portage  5660 May 30 03:35 bash-4.0_p24.ebuild
-rw-r--r--  1 root    root     5660 Jul 25 09:43 bash-4.0_p28.ebuild
drwxr-xr-x  2 portage portage  2048 May 30 03:35 files
-rw-r--r--  1 portage portage   468 Feb  9 04:35 metadata.xml
mars@marsmain /usr/portage/app-shells/bash $ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
<herd>base-system</herd>
<use>
  <flag name='bashlogger'>Log ALL commands typed into bash; should ONLY be
    used in restricted environments such as honeypots</flag>
  <flag name='net'>Enable /dev/tcp/host/port redirection</flag>
  <flag name='plugins'>Add support for loading builtins at runtime via
    'enable'</flag>
</use>
</pkgmetadata>
mars@marsmain /usr/portage/app-shells/bash $ sudo /etc/init.d/bluetooth status
Password:
* status: started
mars@marsmain /usr/portage/app-shells/bash $ ping -q -c1 en.wikipedia.org
PING rr.esams.wikimedia.org (91.198.174.2) 56(84) bytes of data.

--- rr.esams.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 49.820/49.820/49.820/0.000 ms
mars@marsmain /usr/portage/app-shells/bash $ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1               /boot
/dev/sda2               none
/dev/sda3               /
mars@marsmain /usr/portage/app-shells/bash $ date
Sat Aug  8 02:42:24 MSD 2009
mars@marsmain /usr/portage/app-shells/bash $ lsmod
Module                  Size  Used by
rndis_wlan             23424  0
rndis_host              8696  1 rndis_wlan
cdc_ether               5672  1 rndis_host
usbnet                 18688  3 rndis_wlan,rndis_host,cdc_ether
parport_pc             38424  0
fglrx                2388128  20
```



- Главное свойство оболочки, которое делает ее мощным инструментом пользователя — это то, что она включает в себя простой язык программирования.
- Как давно доказано в математике, любой алгоритм можно построить из пары-тройки основных операций и одного условного оператора. Реализацию условных операторов (а также операторов цикла) и берет на себя оболочка.
- Она использует все остальные утилиты и программы (и те, которые имеются в составе операционной системы, и те, что устанавливаются отдельно) как базовые операции поддерживаемого ею языка программирования, обеспечивает передачу им аргументов, а также передачу результатов их работы другим программам или пользователю.
- В результате получается очень мощный язык программирования. И в этом основная сила и одна из существенных функций оболочки.

Специальные символы

` ~ ! @ # $ % ^ & * ( ) _ — [ ] { } : ; ' " / \ > <

Не использовать в программах!

- все что идет после - комментарий

```
#в этой строке мы считаем X
```

Выполнение команд

Оператор ;

имеется возможность задать в одной строке несколько команд, которые будут выполнены последовательно, одна за другой

[user]$ command1 ; command2

оболочка вначале запустит на выполнение команду command1, дождется, пока ее выполнение завершится, после чего запустит command2, дождется ее завершения, после чего снова выведет приглашение командной строки, ожидая следующих действий пользователя.

Выполнение команд

Оператор &
Оператор & используется для того, чтобы организовать исполнение команд в фоновом режиме. Если поставить значок & после команды, то оболочка вернет управление пользователю сразу после запуска команды, не дожидаясь, пока выполнение команды завершится.
Например, если задать в командной строке "command1 & command2 &", то оболочка запустит команду command1, сразу же затем команду command2, и затем немедленно вернет управление пользователю.

Операторы && и ||
Операторы && и || являются управляющими операторами. Если в командной строке стоит command1 && command2, то command2 выполняется в том, и только в том случае, если статус выхода из команды command1 равен нулю, что говорит об успешном ее завершении. Аналогично, если командная строка имеет вид command1 || command2, то команда command2 выполняется тогда, и только тогда, когда статус выхода из команды command1 отличен от нуля.

Запуск команд

Как происходит запуск команд
Можно кратко сказать, что оболочка должна найти код команды, загрузить его в память, передать команде аргументы, заданные в командной строке, а после завершения выполнения соответствующего процесса передать каким-то образом пользователю или другому процессу результаты выполнения данной команды.

Поиск кода команды
Команды бывают встроенные (те, код которых включен в код самой оболочки) и внешние (код которых расположен в отдельном файле на диске).
Встроенную команду оболочка всегда найдет, а для поиска внешней команды пользователь, в принципе, должен указать оболочке полный путь до соответствующего файла.
Однако для облегчения жизни пользователей оболочка умеет искать внешние команды в каталогах, которые перечислены в специально заданных "путях поиска".

Только если она не находит нужных файлов в таких каталогах, она решает, что пользователь ошибся
при вводе имени команды.

Стандартный ввод/вывод

Потоки ввода-вывода
Когда программа запускается на выполнение, в ее распоряжение предоставляются три потока (или канала):

- стандартный ввод (standard input или stdin). По этому каналу данные передаются программе;
- стандартный вывод (standard output или stdout). По этому каналу программа выводит результаты своей работы;
- стандартный поток сообщений об ошибках (standard error или stderr). По этому каналу программы выдают информацию об ошибках.

Из стандартного входа программа может только читать, а два других потока могут использоваться программой только для записи.
По умолчанию входной поток связан с клавиатурой, а выходной поток и поток сообщений об ошибках направлены на терминал пользователя.
Другими словами, вся выходная информация запущенной пользователем команды или программы, а также все сообщения об ошибках, выводятся в окно терминала.

Выходные сообщения можно перенаправить, например, в файл

Перенаправление ввода/вывода

Операторы >, < и >>
Для обозначения перенаправления используются символы ">", "<" и ">>".
Чаще всего используется перенаправление вывода команды в файл.

```
[user]$ ls -l > /home/jim/dir.txt
```

По этой команде в файле /home/jim/dir.txt будет сохранен перечень файлов и подкаталогов того каталога, который был текущим на момент выполнения команды ls; при этом если указанного файла не существовало, то он будет создан; если он существовал, то будет перезаписан; если же вы хотите, чтобы вывод команды был дописан в конец существующего файла, то надо вместо символа > использовать >>.
Наличие пробелов до или после символов > или >> несущественно и служит только для удобства пользователя.
Вы можете направить вывод не только в файл, но и на вход другой команды или на устройство (например, принтер).
Для подсчета числа слов в файле /home/jim/report.txt можно использовать следующую команду:

```
[user]$ cat /home/jim/report.txt > wc -w
```

Перенаправление ввода/вывода

Оператор > служит для перенаправления выходного потока.
По отношению к входному потоку аналогичную функцию выполняет оператор <.
Пример команды для подсчета числа слов в определенном файле:

```
[user]$ wc -w < /home/jim/report.txt
```

Этот вариант перенаправления часто используется в различных скриптах, применительно к тем командам, которые обычно воспринимают ввод (или ожидают ввода) с клавиатуры. В скрипте же, автоматизирующем какие-то рутинные операции, можно дать команде необходимую информацию из файла, в который заранее записано то, что нужно ввести для выполнения этой команды.

Поскольку символы <, > и >> действуют на стандартные потоки, их можно использовать не только тем привычным образом, как это делается обычно, но и несколько по-другому.

Так, следующие команды эквивалентны:

```
[user]$ cat > file
[user]$ cat>file
[user]$ >file cat
[user]$ > file cat
```

Оператор |
Особым вариантом перенаправления вывода является организация программного канала (иногда называет трубопроводом или конвейером).
Для этого две или несколько команд, таких, что вывод предыдущей служит вводом для следующей, соединяются символом вертикальной черты — "|".
При этом стандартный выходной поток команды, расположенной слева от символа |, направляется на стандартный ввод программы, расположенной справа от символа |.

```
[user]$ cat myfile | grep Linux | wc -l
```

Эта строка означает, что вывод команды cat, т. е. текст из файла myfile, будет направлен на вход команды grep, которая выделит только строки, содержащие слово "Linux". Вывод команды grep будет, в свою очередь, направлен на вход команды wc -l , которая подсчитает число таких строк.

Программные каналы используются для того, чтобы скомбинировать несколько маленьких программ, каждая из которых выполняет только определенные преобразования над своим входным потоком, для создания обобщенной команды, результатом которой будет какое-то более сложное преобразование.

Параметры и переменные

Понятие параметра в bash подобно понятию переменной в обычных языках программирования. Именем (или идентификатором) параметра может быть слово, состоящее из алфавитных символов, цифр и знаков подчеркивания (только первый символ этого слова не может быть цифрой), а также число или один из следующих специальных символов:
*, @, #, ?, - (дефис), $, !, 0, _ (подчеркивание).
Параметр задан или установлен, если ему присвоено значение. Значением может быть и пустая строка.
Значения переменным присваиваются с помощью оператора следующего вида

```
[user]$ name=value
```

где name — имя переменной, а value — присваиваемое ей значение (может быть пустой строкой). Значением может быть любой текст. Если значение содержит специальные символы, то его надо взять в кавычки.
Если переменная задана, то ее можно удалить, используя встроенную команду оболочки unset.
Для того чтобы вывести значение одной конкретной переменной, можно воспользоваться командой

```
[user]$ echo $name
```

(нужно использовать символ $ перед его именем).

Параметры

Разновидности параметров
Параметры разделяются на три класса:

- позиционные параметры,
- специальные параметры (именами которых как раз и служат перечисленные только что специальные символы)
- переменные оболочки.

Значениями позиционных параметров являются аргументы, которые были заданы при запуске оболочки (первый аргумент является значением позиционного параметра 1, и т. д.).

```
[user]$ command <param1> <param2> <param3>

$1 - param1
$2 - param2
$3 - param3
```

Изменить значение позиционного параметра можно с помощью встроенной команды set. Значения этих параметров изменяются также на время выполнения оболочкой одной из функций

Переменные окружения

PATH	Пути для поиска программ -- список директорий, разделенных двоеточиями
PROMPT	Вид приглашения shell
TERM	Тип терминала
HOME	Домашняя директория пользователя
USER	Login-имя пользователя
SHELL	Имя основного shell
DISPLAY	Имя дисплея для X-программ
EDITOR	Текстовый редактор, который будут использовать программы mc, vipw, crontab и т.д. вместо vi

Переменные окружения

Чтобы присвоить значение переменной окружения или изменить его, используется команда export.

```
export DISPLAY=localhost:0
```

Вокруг символа "=" не должно быть пробелов
Для того ,чтобы посмотреть все переменные окружения и их значения, нужно запустить команду env или команду export без параметров.
Чтобы посмотреть значение переменной, можно воспользоваться командой echo:

```
echo $DISPLAY
echo $PATH
```

Для удаления переменной окружения используется команда unset.
Одной из важнейших переменных окружения является переменная PATH.
Она задает перечень путей к каталогам, в которых bash осуществляет поиск файлов (в частности, файлов с командами) в тех случаях, когда полный путь к файлу не задан в командной строке.
Отдельные каталоги в этом перечне разделяются двоеточиями. По умолчанию переменная PATH включает каталоги /usr/local/bin, /bin, /usr/bin, /usr/X11R6/bin
Для того, чтобы добавить каталог в этот список, нужно выполнить следующую команду:

```
[root]# export PATH=$PATH:new_path.
```

При осуществлении поиска оболочка просматривает каталоги именно в том порядке, как они перечислены в переменной PATH.

Запуск программ

в Unix-подобных ОС (включая и Linux) команды ищутся только в директориях, перечисленных в PATH - если "." там не указана (а обычно так и есть), то в текущей директории программа искаться не будет. Для запуска программы из текущей директории надо явно указать путь,например:

```
[user]$ ./myprog
```