Операционные системы (6 семестр)

Лекция 1.5. Программирование для bash. Расширенные возможности командной оболочки bash.

Ассистент, к.т.н. Митричев Иван Игоревич

Москва 2018

План лекции

- Скобки в командной строке.
- Вычисления в стиле C.
- Условное исполнение команд.
- Команда case.
- Циклы.
- Функции.
- Here-документы, here-строки.
- Подстановка процесса (process substitution).

Скобки [, [[, ((, $(( ))

Запомним:

[- синоним test [[- улучшенный оператор в bash.

[[позволяет:

1) использовать && и || для группировки условий (вместо -a и -o)

2) использовать регулярные выражения. Оператор выбора по шаблону =~

```
if [[ $answer =~ ^y(es)?$ ]]
```

и захват () символов. В результате в переменной ${BASH_REMATCH[1]} находится «es».

3) не комментировать названия переменных

```
if [[ -f $file ]]
```

Обычно [[используют для строк и файлов.

Для сравнения чисел используют (( )) (оператор в стиле C):

```
if ((a > 5)); then echo "a is more than 5"; fi
```

- не возвращает результат

$(( )) - арифметическая подстановка — возвращает результат как текст

Группировка команд

Группировка команд возможна с помощью () или {}

Команды в фигурных скобках выполняются без запуска подпроцесса (подоболочки), а команды в круглых скобках выполняются в собственной подоболочке.

```
true && (echo a; sleep 1 && exit 0 || echo b) && echo c
```

команда exit прервала выполнение команд в скобках, но не скрипта в целом.

Поэтому, "echo c" срабатывает

Результат выполнения:

a

c

```
true && {echo a; sleep 1 && exit 0 || echo b} && echo c
```

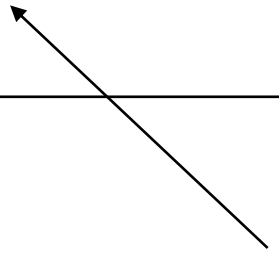
команда exit прервала выполнение всей строки команд

Результат выполнения:

a

Почему операторы для сравнения строк и арифметические операторы не взаимозаменяемы

```
$ [[ 2 -lt 10 ]] && echo less || echo not
less
$ [[ 2 < 10 ]] && echo less || echo not
not
```



Двойные скобки??

Другие примеры (в том числе, для знака равенства)
https://unix.stackexchange.com/a/143800/58326

Работа с переменными в стиле языка C

```
#!/bin/bash
(( a = 23 ))
echo "a (начальное значение) = $a"
(( a++ ))
echo "a (после a++) = $a"
(( a-- ))
echo "a (после a--) = $a"  (( ++a ))
echo "a (после ++a) = $a"  (( --a ))
echo "a (после --a) = $a"

(( t = a<45?7:11 ))   # Тернарный оператор
                      в стиле C.
echo "If a < 45, then t = 7, else t = 11"
echo "t = $t "        # Да!
echo
```

Двойные круглые скобки похожи на инструкцию let (вычисления и результат). Позволяют работать с переменными в стиле языка C.

Пример:

```
$ ./pr3.sh
a (начальное значение) = 23
a (после a++) = 24
a (после a--) = 23
a (после ++a) = 24
a (после --a) = 23
If a < 45, then t = 7, else t = 11
t = 7
```

Пример с https://www.opennet.ru/docs/RUS/bash_scripting_guide/x4862.html

Условное исполнение команд

<команда1> && <команда2> ←

- выполнение второй команды только в случае успеха первой;

<команда1> || <команда2> ←

- выполнение второй команды только в случае неудачи первой;

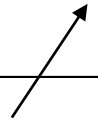
true (всегда возвращает 0) и **false** (всегда возвращает 1).

•команда **true** вернула 0 и **echo "1"** тоже вернула 0,
команда **false**, которая вернула 1 и выполнение команд прекратилось (команда **echo 2** не выполнилась).

```
$ true && echo "1" && false && echo "2"
1
$ true && echo "1" || false && echo "2"
1
2
```

Условный оператор if

```
if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS; ]
… [ else COMMANDS; ] fi
```



- Точка с запятой нужна только в том случае если на одной строке расположено более одного ключевого элемента конструкции **if**.

Многострочный синтаксис без точки с запятой:

```
if COMMANDS
then
    COMMANDS
elif COMMANDS
then
    COMMANDS
else
  COMMANDS
fi
```

Циклы

```
for <имя> in <список>
do
команды, которые используют переменную $<имя>
done
```

- for – цикл перебора значений;

```
while [ условие ];
do
команды
done
```

- while – выполняется, пока истинно условие;

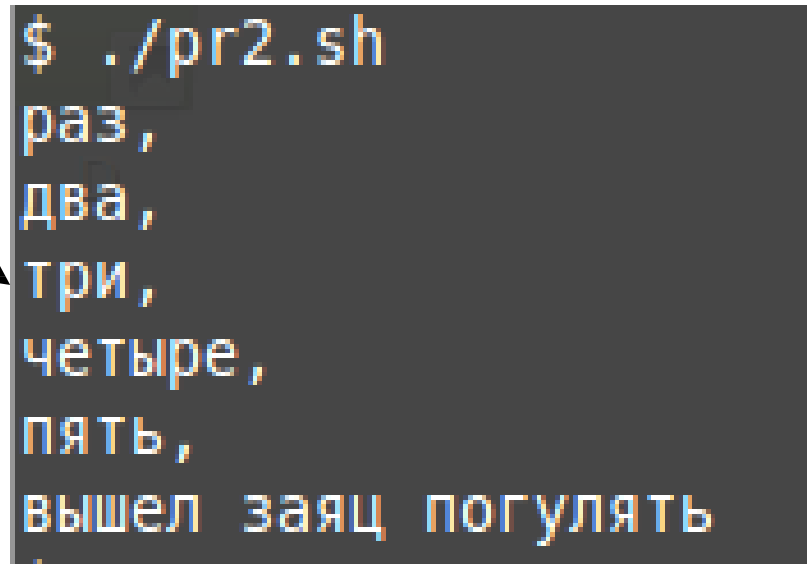
```
until [ условие ];
do
команды
done
```

- until – выполняется, пока условие ложно.

Пример for

Цикл с перебором

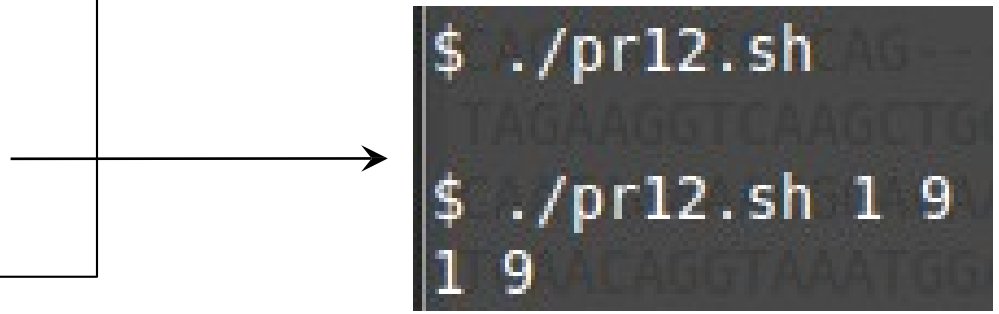
```
#!/bin/bash
# пример цикла по множеству значений
for A in раз два три четыре пять
do
echo "$A,"
done
echo "вышел заяц погулять"
```



Цикл for без списка аргументов

```
#!/bin/bash
# Вызываем с аргументами и
без.
for a
do echo -n "$a "  done
echo
exit 0
```

Данный синтаксис приводит к обработке всех аргументов командной строки, включая пробелы



В цикле for можно использовать командную подстановку.

```
for a in `echo a b`              # 2 аргумента: a и b
for a in  "`echo a b`"           # 1 аргумент: "a b"
for a in `echo`                  # без аргументов
for a in "`echo`"                # один пустой аргумент
```

Пример: копирование файлов в каталог

Имя каталога задано как последний аргумент в командной строке.

```
#!/bin/bash
[ $# -lt 1 ] && exit 1
i=0
for FILE
do
  ((i++))
  if ((i == $#)); then break; fi
  echo "Copying $FILE"
  cp $FILE "${@: -1}"/
done
```

Внутри (( )) используем С-стиль операторов!

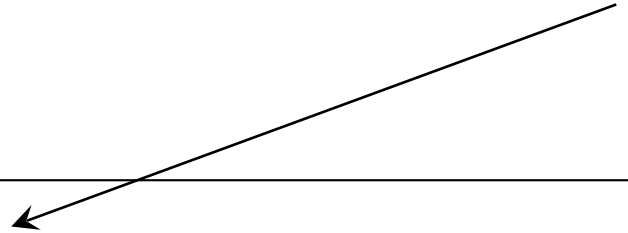
```
$./t.sh s1 s2 many1
Copying s1
Copying s2
```

Пример while

C-подобный синтаксис оформления цикла while.

expr — устаревая встроенная команда оболочки, сходная с $(( )) и (( ))

```
#!/bin/bash  A="10"
B="5"
C=`expr $A + $B`
printf "A=10 B=5 \n C=expr \$A + \$B C=%d \n" "$C"
 # пример цикла по I
I=0
while [ $I -lt 15 ]  do
printf "0x%02x " "$I"
I=`expr $I + 1`
done
```



```
$ ./pr1.sh
A=10 B=5
 C=expr $A + $B C=15
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e
```

Пример until

```
#!/bin/bash
until [ "$var1" = end ]  do
 echo "Введите значение переменной #1 "
 echo "(end — выход)"
 read var1
 echo "значение переменной #1 = $var1"
done
```

Оператор **until** проверяет условие завершения цикла до итерации (отличие от Pascal).

```
$ ./pr4.sh
Введите значение переменной #1
(end - выход)
4
значение переменной #1 = 4
Введите значение переменной #1
(end - выход)
end
значение переменной #1 = end
```

Итерация возможна только в том случае, если условие ложно.

Вложенные циклы

```
#!/bin/bash
outer=1         # Счетчик внешнего цикла.
for a in 1 2 3
do
  inner=1       # Сброс счетчика вложенного
  for b in 1 2 3цикла.
  do
        printf "$outer, $inner          "
        let "inner+=1"
        done
        let "outer+=1"
        echo # Конец строки
  done
```

Как внешний, так и внутренний циклы могут быть прерваны командой **break**.

```
$ ./pr6.sh
1, 1      1, 2      1, 3
2, 1      2, 2      2, 3
3, 1      3, 2      3, 3
```

Генерирование последовательностей чисел

```
$for i in {1..15..5}; do echo $i; done
1
6
11
```

- последовательность с инкрементом 5

```
$ for i in $(seq 0 5 15); do echo i=$i; done
i=0
i=5
i=10
i=15
```

- задание последователь-ности seq

```
$ for i in "$(seq 0 5 15)"; do echo i=$i; done
i=0 5 10 15
```

- влияние кавычек

```
$ for ((i=0;i<5;i++)); do echo i=$i; done
i=0
i=1
i=2
i=3
i=4
```

- for в C-стиле

Команда case

```
case слово in
шаблон1 )
команды
                ;;
шаблон2 )
команды
                ;;
esac
```

- Команда case проверяет значение заданной переменной на совпадение с шаблонами.

```
select variable [in list]  do
команды…
break
done
```

- Команда select позволяет вывести пользователю контекстное меню выбора опций, где каждая опция получает свой номер

Функции

http://tldp.org/LDP/abs/html/functions.html

```
function имя
{
команды
}
```

или

```
имя ()
{
        команды
}
```

Функции должны быть описаны в файле до первого вызова.

Часто функции описывают в отдельных файлах и пользуются inline-подстановкой.

Можно вызывать функции из функций, в т.ч., рекурсивно.

Примеры:

```
#!/bin/bash
function test
{
        echo 'I am test'
}
test
```

```
#!/bin/bash
mat() {
    a=1
    ((a++))
    echo $a
}
mat
```

Скобку можно оставить на предыдущей строке.

Передача параметров в функции

```
<имя функции> <параметр1> <параметр2>
...
```

```
for var in "$@“
или
$1, ..., ${N}, а также shift
```

- вызов функции с параметрами;
- доступ к параметрам из тела функции.

Пример: использование аргументов.

```
#!/bin/bash  mat()  {
    a=$1
    b=$2
    c=$(( a + b ))
    echo $c
}
mat $1 $2
```

```
$ ./mat.sh 1 1
2
```

Heredoc. Подстановка процесса

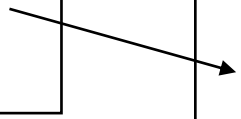
Синтаксис here-document:

В качестве ограничителя может быть использована любая строка.

```
команда(-ы) <<ограничитель
строки текста/команды
ограничитель
```

Пример:

```
cat <<End-of-message
This is line 1.
This is line 2.
End-of-message
```



```
This is line 1.
This is line 2.
```

Пример однострочного heredoc = here-string

```
String="Love Linux"
read -r -a Words <<< "$String"
echo "First word of String is: ${Words[0]}" # Love
echo "Second word of String is: ${Words[1]}" # Linux
```

Подстановка процесса – аналог командной подстановки, используется для передачи результатов исполнения одной команды другой команде. Синтаксис

```
>(command)
<(command)
```

Пример:

```
cat <(ls -l)
# То же самое, что и ls -l | cat
```