Циклы в сценариях

Команда `test []`

Команда **`test`** позволяет установить, является ли какое-либо выражение истинным или ложным. Давайте начнем с проверки, больше ли целочисленное значение `10` целочисленного значения `55`.

```
[paul@RHEL4b ~]$ test 10 -gt 55 ; echo $?
```

```
1
[paul@RHEL4b ~]$
```

Команда test возвращает значение 1, если выражение является ложным. И, как вы увидите в следующем примере, команда test будет возвращать значение 0, если выражение будет являться истинным.

```
[paul@RHEL4b ~]$ test 56 -gt 55 ; echo $?
0
[paul@RHEL4b ~]$
```

Если же вам удобнее работать со строками true (истина) и false (ложь), вы можете использовать команду test таким образом, как показано ниже.

```
[paul@RHEL4b ~]$ test 56 -gt 55 && echo true || echo
false
true
[paul@RHEL4b ~]$ test 6 -gt 55 && echo true || echo false
false
```

Команда test также может заменяться квадратными скобками, поэтому команды из примера ниже полностью аналогичны командам из примера выше.

```
[paul@RHEL4b ~]$ [ 56 -gt 55 ] && echo true || echo false
true
[paul@RHEL4b ~]$ [ 6 -gt 55 ] && echo true || echo false
false
```

Ниже приведены примеры реализаций некоторых проверок. Обратитесь к странице руководства man test для ознакомления с дополнительными возможностями реализации различных проверок.

```
[ -d foo ]              Существует ли директория foo ?
[ -e bar ]              Существует ли файл bar ?
[ '/etc' = $PWD ]       Эквивалентна ли строка /etc значе-
нию переменной $PWD ?
[ $1 != 'secret' ]      Отличается ли значение первого па-
раметра сценария от строки secret ?
[ 55 -lt $bar ]         Меньше ли целочисленное значение
55 значения переменной $bar ?
```

```
[ $foo -ge 1000 ]      Является ли значение переменной
$foo большим или равным целочисленному значению 1000 ?
[ "abc" < $bar ]       Будет ли строка abc расположена
выше значения переменной $bar в списке после сортировки ?
[ -f foo ]             Является ли foo обычным файлом ?
[ -r bar ]             Является ли bar читаемым файлом ?
[ foo -nt bar ]        Новее ли файл foo файла bar ?
[ -o nounset ]         Активирован ли параметр командной
оболочки nounset ?
```

Операторы проверок могут комбинироваться с операторами, соответствующими логическим операциям "И" и "ИЛИ".

```
paul@RHEL4b:~$ [ 66 -gt 55 -a 66 -lt 500 ] && echo true
|| echo false
true
paul@RHEL4b:~$ [ 66 -gt 55 -a 660 -lt 500 ] && echo true
|| echo false
false
paul@RHEL4b:~$ [ 66 -gt 55 -o 660 -lt 500 ] && echo true
|| echo false
true
```

Условный переход `if then else`

Конструкция **`if then else`** предназначена для выбора варианта кода. В том случае, если определенное условие выполняется, будет исполняться какой либо код, в противном случае будет исполняться какой-либо другой код. В примере ниже осуществляется проверка существования файла, после чего в том случае, если предположение о существовании файла подтверждается, осуществляется вывод соответствующего сообщения.

```
#!/bin/bash

if [ -f isit.txt ]
then echo файл isit.txt существует!
else echo файл isit.txt не найден!
fi
```

В том случае, если мы сохраним данный код сценария в файле с именем 'choice', он сможет быть исполнен аналогичным образом.

```
[paul@RHEL4a scripts]$ ./choice
файл isit.txt не найден!
[paul@RHEL4a scripts]$ touch isit.txt
[paul@RHEL4a scripts]$ ./choice
файл isit.txt существует!
[paul@RHEL4a scripts]$
```

Условный переход `if then elif`

Вы можете разместить новый оператор условного перехода **`if`** внутри блока **`else`**, воспользовавшись оператором **`elif`**. Ниже приведен простой пример такой записи.

```
#!/bin/bash
count=42
if [ $count -eq 42 ]
then
  echo "42 является корректным значением."
elif [ $count -gt 42 ]
then
  echo "Слишком много."
else
  echo "Не достаточно."
fi
```

Цикл `for`

В примере ниже представлен синтаксис классического цикла **`for`** в командной оболочке bash.

```
for i in 1 2 4
do
   echo $i
done
```

Пример использования цикла **`for`**, скомбинированного с вызовом встраиваемой командной оболочки.

```
#!/bin/ksh
for counter in `seq 1 20`
do
   echo отсчет от 1 до 20, текущее значение $counter
```

```
   sleep 1
done
```

Сценарий, полностью аналогичный представленному выше, может быть создан и без задействования встраиваемой командной оболочки путем использования реализованного в рамках командной оболочки bash объявления диапазона значений **{от значения..до значения}**.

```
#!/bin/bash
for counter in {1..20}
do
   echo отсчет от 1 до 20, текущее значение $counter
   sleep 1
done
```

В данном цикле **for** используется механизм поиска файлов по шаблону (реализованный в рамках механизма раскрытия команд). В случае размещения приведенной инструкции непосредственно в командной строке, она будет функционировать аналогично.

```
kahlan@solexp11$ ls
count.ksh  go.ksh
kahlan@solexp11$ for file in *.ksh ; do cp $file
$file.backup ; done
kahlan@solexp11$ ls
count.ksh  count.ksh.backup  go.ksh  go.ksh.backup
```

Цикл while

Ниже приведен простой пример использования цикла **while**.

```
i=100;
while [ $i -ge 0 ] ;
do
   echo Обратный отсчет от 100 до 0, текущее значение $i;
   let i--;
done
```

Бесконечные циклы могут реализовываться с помощью объявлений **while true** или **while :** , где символ **:** является эквивалентом **отсутствующей операции** в командных оболочках **Korn shell** и **bash**.

```
#!/bin/ksh
# бесконечный цикл
while :
do
 echo hello
 sleep 1
done
```

Цикл until

Ниже приведен простой пример использования цикла until.

```
let i=100;
until [ $i -le 0 ] ;
do
   echo Обратный отсчет от 100 до 1, текущее значение $i;
   let i--;
done
```