# Дополнительная информация о сценариях

## Команда eval

Команда **eval** позволяет интерпретировать переданные аргументы как директивы сценария командной оболочки (результирующие команды исполняются). Данное обстоятельство позволяет использовать значение переменной в качестве переменной.

```
paul@deb503:~/test42$ answer=42
paul@deb503:~/test42$ word=answer
paul@deb503:~/test42$ eval x=\$$word ; echo $x
42
```

Как в командной оболочке **bash**, так и командной оболочке **Korn shell** аргументы могут экранироваться с помощью двойных кавычек.

```
kahlan@solexp11$ answer=42
kahlan@solexp11$ word=answer
kahlan@solexp11$ eval "y=\$$word" ; echo $y
42
```

Иногда команде eval необходимо передавать аргументы таким образом, чтобы командной оболочкой осуществлялся их корректный разбор. Рассмотрите приведенный ниже пример, в котором команда **date** принимает один параметр, являющийся строкой "**1 week ago**".

```
paul@debian6~$ date --date="1 week ago"
Чт мар  8 21:36:25 CET 2012
```

В том случае, если мы сохраняем данную команду в переменную, исполнение команды из этой переменной будет заканчиваться неудачей до того момента, когда мы начнем использовать команду **eval**.

```
paul@debian6~$ lastweek='date --date="1 week ago"'
paul@debian6~$ $lastweek
date: лишний операнд `ago"'

По команде `date --help' можно получить дополнительную информацию.

paul@debian6~$ eval $lastweek
Чт мар  8 21:36:39 CET 2012
```

### Оператор (( ))

Оператор **(( ))** позволяет сравнивать числовые значения.

```
paul@deb503:~/test42$ (( 42 > 33 )) && echo true || echo
false
true
paul@deb503:~/test42$ (( 42 > 1201 )) && echo true ||
echo false
false
paul@deb503:~/test42$ var42=42
paul@deb503:~/test42$ (( 42 == var42 )) && echo true ||
echo false
true
paul@deb503:~/test42$ (( 42 == $var42 )) && echo true ||
echo false
true
paul@deb503:~/test42$ var42=33
paul@deb503:~/test42$ (( 42 == var42 )) && echo true ||
echo false
false
```

### Команда `let`

Встроенная команда командной оболочки **`let`** инструктирует командную оболочку о необходимости вычисления значений арифметических выражений. Она будет возвращать значение `0`, если результат последней арифметической операции не равен `0`.

```
[paul@RHEL4b ~]$ let x="3 + 4" ; echo $x
7
[paul@RHEL4b ~]$ let x="10 + 100/10" ; echo $x
20
[paul@RHEL4b ~]$ let x="10-2+100/10" ; echo $x
18
[paul@RHEL4b ~]$ let x="10*2+100/10" ; echo $x
30
```

Команда **`let`** также может использоваться для перевода значений в различные системы счисления.

```
[paul@RHEL4b ~]$ let x="0xFF" ; echo $x
```

```
255
[paul@RHEL4b ~]$ let x="0xC0" ; echo $x
192
[paul@RHEL4b ~]$ let x="0xA8" ; echo $x
168
[paul@RHEL4b ~]$ let x="8#70" ; echo $x
56
[paul@RHEL4b ~]$ let x="8#77" ; echo $x
63
[paul@RHEL4b ~]$ let x="16#c0" ; echo $x
192
```

Существует различие между непосредственным присваиванием значения переменной и использованием команды **let** для расчета значений арифметических выражений (даже в том случае, если с помощью данной команды осуществляется исключительно присваивание значения переменной).

```
kahlan@solexp11$ dec=15 ; oct=017 ; hex=0x0f
kahlan@solexp11$ echo $dec $oct $hex
15 017 0x0f
kahlan@solexp11$ let dec=15 ; let oct=017 ; let hex=0x0f
kahlan@solexp11$ echo $dec $oct $hex
15 15 15
```

## Оператор case

В некоторых случаях вы можете упростить конструкции из вложенных условных переходов `if`, воспользовавшись конструкцией **case**.

```
[paul@RHEL4b ~]$ ./help
Какое животное вы видите ? лев
Лучше всего быстро убегать!
[paul@RHEL4b ~]$ ./help
Какое животное вы видите ? собака
Не беспокойтесь, угостите ее печеньем.
[paul@RHEL4b ~]$ cat help
#!/bin/bash
```

```
#
# Советы по обращению с дикими животными
#
echo -n "Какое животное вы видите ? "
read animal
case $animal in
        "лев" | "тигр")
                echo "Лучше всего быстро убегать!"
        ;;
        "кот")
                echo "Выпустите мышь..."
        ;;
        "собака")
                echo "Не беспокойтесь, угостите ее пече-
ньем."
        ;;
        "курица" | "гусь" | "утка" )
                echo "Яйца на завтрак!"
        ;;
        "лигр")
                echo "Подойдите и скажите: 'Ах ты, боль-
шой пушистый котенок...'."
        ;;
        "вавилонская рыбка")
                echo "Она выпала из вашего уха ?"
        ;;
        *)
                echo "Вы обнаружили неизвестное животное,
дайте ему имя!"
        ;;
esac
[paul@RHEL4b ~]$
```

## Функции сценариев командной оболочки

**Функции** сценариев командной оболочки могут использоваться для логической группировки команд.

```
kahlan@solexp11$ cat funcs.ksh
#!/bin/ksh

function greetings {
echo Hello World!
echo а также приветствуем пользователя $USER!
}

echo Сейчас мы вызовем функцию
greetings
echo Конец
```

А это пример вывода данного сценария командной оболочки с **функцией**.

```
kahlan@solexp11$ ./funcs.ksh
Сейчас мы вызовем функцию
Hello World!
а также приветствуем пользователя kahlan!
Конец
```

Функция сценария командной оболочки также может принимать параметры.

```
kahlan@solexp11$ cat addfunc.ksh
#!/bin/ksh

function plus {
let result="$1 + $2"
echo  $1 + $2 = $result
}

plus 3 10
plus 20 13
plus 20 22
```

Данный сценарий генерирует следующий вывод:

```
kahlan@solexp11$ ./addfunc.ksh
3 + 10 = 13
20 + 13 = 33
20 + 22 = 42
```