Параметры сценариев

Параметры сценария

Сценарий командной оболочки **bash** может принимать параметры. Нумерация, которую вы можете увидеть в сценарии ниже, может быть продолжена, если для работы сценария необходимо большее количество параметров. Также в вашем распоряжении имеются специальные параметры, содержащие значения, которые соответствуют количеству параметров, их строковому представлению, а также идентификатору процесса и последнему коду завершения процесса. Полный список специальных параметров размещен на странице руководства командной оболочки **bash**.

```
#!/bin/bash
echo Первым аргументом является $1
echo Вторым аргументом является $2
echo Третьим аргументом является $3

echo \$ $$  - идентификатор процесса (PID) интерпретатора
сценария
echo \# $#  - количество аргументов
echo \? $?  - последний код завершения процесса
echo \* $*  - строковое представление всех аргументов
Ниже представлен вывод данного сценария.
[paul@RHEL4a scripts]$ ./pars one two three
Первым аргументом является one
Вторым аргументом является two
Третьим аргументом является three
$ 5610 - идентификатор процесса (PID) интерпретатора
сценария
# 3  - количество аргументов
? 0  - последний код завершения процесса
* one two three - строковое представление всех аргументов
```

Еще один вариант исполнения данного сценария, но теперь с передачей только двух параметров.

```
[paul@RHEL4a scripts]$ ./pars 1 2
Первым аргументом является 1
```

```
Вторым аргументом является 2
Третьим аргументом является
$ 5612 - идентификатор процесса (PID) интерпретатора
сценария
# 2 - количество аргументов
? 0 - последний код завершения процесса
* 1 2 - строковое представление всех аргументов
[paul@RHEL4a scripts]$
```

А это другой пример сценария, в котором мы используем параметр **$0**. Параметр **$0** содержит имя файла исполняющегося сценария.

```
paul@debian6~$ cat myname
echo имя файла этого сценария $0
paul@debian6~$ ./myname
имя файла этого сценария ./myname
paul@debian6~$ mv myname test42
paul@debian6~$ ./test42
имя файла этого сценария ./test42
```

Обход списка параметров

Оператор обхода списка параметров (**shift**) позволяет произвести разбор всех переданных сценарию **параметров** по очереди. Ниже представлен пример сценария с данным оператором.

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

if [ "$#" == "0" ]
 then
  echo Вы должны передать по крайней мере один параметр
сценарию.
  exit 1
fi

while (( $# ))
 do
```

```
  echo Вы передали сценарию параметр $1
  shift
 done
```

А это вариант вывода приведенного выше сценария.

```
kahlan@solexp11$ ./shift.ksh one
Вы передали сценарию параметр one
kahlan@solexp11$ ./shift.ksh one two three 1201 "33 42"
Вы передали сценарию параметр one
Вы передали сценарию параметр two
Вы передали сценарию параметр three
Вы передали сценарию параметр 1201
Вы передали сценарию параметр 33 42
kahlan@solexp11$ ./shift.ksh
Вы должны передать по крайней мере один параметр сценарию.
```

Ввод в процессе исполнения сценария

Вы можете попросить пользователя ввести данные, воспользовавшись командой **read** в сценарии.

```
#!/bin/bash
echo -n Введите число:
read number
```

Задействование файла конфигурации

Команда **source** (о которой говорилось в одной из глав, посвященных работе с командной оболочкой) может также использоваться для задействования файла конфигурации сценария.

Ниже приведено содержимое демонстрационного файла конфигурации для приложения.

```
[paul@RHEL4a scripts]$ cat myApp.conf
# Файл конфигурации для приложения myApp

# Путь
myAppPath=/var/myApp

# Количество приложений, осуществляющих саморепликацию
```

```
quines=5
А это код приложения, которое использует данный файл.
[paul@RHEL4a scripts]$ cat myApp.bash
#!/bin/bash
#
# Добро пожаловать в приложение myApp
#

. ./myApp.conf

echo Количество самовоспроизводящихся приложений: $quines
```

В ходе исполнения приложение может использовать значения из подключенного файла конфигурации.

```
[paul@RHEL4a scripts]$ ./myApp.bash
Количество самовоспроизводящихся приложений: 5
[paul@RHEL4a scripts]$
```

Получение параметров сценария с помощью функции getopts

Функция **getopts** позволяет осуществлять разбор параметров, передаваемых сценарию посредством командного интерфейса системы. Следующий сценарий позволяет использовать любую комбинацию параметров `a`, `f` и `z`.

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

while getopts ":afz" option;
do
 case $option in
  a)
   echo принят параметр -a
   ;;
  f)
   echo принят параметр -f
   ;;
```

```
  z)
   echo принят параметр -z
   ;;
  *)
   echo "некорректный параметр -$OPTARG"
   ;;
 esac
done
```

Ниже представлен вариант вывода данного сценария. Сначала мы передаем исключительно допустимые параметры, а затем дважды передаем некорректный параметр.

```
kahlan@solexp11$ ./options.ksh
kahlan@solexp11$ ./options.ksh -af
принят параметр -a
принят параметр -f
kahlan@solexp11$ ./options.ksh -zfg
принят параметр -z
принят параметр -f
некорректный параметр -g
kahlan@solexp11$ ./options.ksh -a -b -z
принят параметр -a
некорректный параметр -b
принят параметр -z
```

Также вы можете обрабатывать параметры, которые подразумевают использование аргумента таким образом, как показано в данном примере.

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

while getopts ":af:z" option;
do
 case $option in
  a)
   echo принят параметр -a
```

```
   ;;
  f)
   echo принят параметр -f с аргументом $OPTARG
   ;;
  z)
   echo принят параметр -z
   ;;
  :)
   echo "вместе с параметром -$OPTARG должен быть передан
аргумент"
   ;;
  *)
   echo "некорректный параметр -$OPTARG"
   ;;
 esac
done
```

А это вариант вывода представленного выше сценария.

```
kahlan@solexp11$ ./argoptions.ksh -a -f hello -z
принят параметр -a
принят параметр -f с аргументом hello
принят параметр -z
kahlan@solexp11$ ./argoptions.ksh -zaf 42
принят параметр -z
принят параметр -a
принят параметр -f с аргументом 42
kahlan@solexp11$ ./argoptions.ksh -zf
принят параметр -z
вместе с параметром -f  должен быть передан аргумент
```

Получение параметров функционирования командной оболочки с помощью команды shopt

Вы можете изменять значения переменных, управляющих аспектами функционирования командной оболочки, с помощью встроенной команды командной оболочки **shopt**. В примере ниже сначала осуществляется проверка того, установлено ли значение переменной

cdspell; оказывается, что оно не установлено. После этого с помощью команды shopt осуществляется установка значения данной переменной, а третья команда, в составе которой используется команда shopt, предназначается для проверки того, действительно ли была осуществлена установка значения рассматриваемой переменной. После установки значения данной переменной вы можете допускать незначительные опечатки в ходе использования команды cd. Страница руководства командной оболочки bash содержит полный список допустимых переменных.

```
paul@laika:~$ shopt -q cdspell ; echo $?
1
paul@laika:~$ shopt -s cdspell
paul@laika:~$ shopt -q cdspell ; echo $?
0
paul@laika:~$ cd /Etc
/etc
```