

[Каталог документации](#) / [Раздел "Программирование, языки"](#) / [Оглавление документа](#)**Advanced Bash-Scripting Guide: Искусство программирования на языке сценариев командной оболочки**[Назад](#)

Глава 4. Переменные и параметры. Введение.

[Вперед](#)

4.4. Специальные типы переменных

локальные переменные

переменные, область видимости которых ограничена [блоком кода](#) или телом функции (см так же [локальные переменные в функциях](#))

переменные окружения

переменные, которые затрагивают командную оболочку и порядок взаимодействия с пользователем



В более общем контексте, каждый процесс имеет некоторое "окружение" (среду исполнения), т.е. набор переменных, к которым процесс может обращаться за получением определенной информации. В этом смысле командная оболочка подобна любому другому процессу.

Каждый раз, когда запускается командный интерпретатор, для него создаются переменные, соответствующие переменным окружения. Изменение переменных или добавление новых переменных окружения заставляет оболочку обновить свои переменные, и все дочерние процессы (и команды, исполняемые ею) наследуют это окружение.



Пространство, выделяемое под переменные окружения, ограничено. Создание слишком большого количества переменных окружения или одной переменной, которая занимает слишком большое пространство, может привести к возникновению определенных проблем.

```
bash$ eval "`seq 10000 | sed -e 's/.*/export  
var&=ZZZZZZZZZZZZZZZ/'`"
```

```
bash$ du
```

```
bash: /usr/bin/du: Argument list too long
```

(Спасибо S. C. за вышеприведенный пример и пояснения.)

Если сценарий изменяет переменные окружения, то они должны "экспортироваться", т.е. передаваться окружению, локальному по отношению к сценарию. Эта функция возложена на команду **export**.



Сценарий может **экспортировать** переменные только дочернему процессу, т.е. командам и процессам запускаемым из данного сценария. Сценарий, запускаемый из командной строки *не может* экспортировать переменные "на верх" командной оболочки.

Дочерний процесс не может экспортировать переменные родительскому процессу.

позиционные параметры

аргументы, передаваемые скрипту из командной строки -- \$0, \$1, \$2, \$3..., где \$0 -- это название файла сценария, \$1 -- это первый аргумент, \$2 -- второй, \$3 -- третий и так далее. [1] Аргументы, следующие за \$9, должны заключаться в фигурные скобки, например: \${10}, \${11}, \${12}.

Специальные переменные **\$*** и **\$@** содержат все позиционные параметры (аргументы командной строки).

Пример 4-5. Позиционные параметры

```
#!/bin/bash

# Команда вызова сценария должна содержать по меньшей мере 10
# параметров, например
# ./scriptname 1 2 3 4 5 6 7 8 9 10
MINPARAMS=10

echo

echo "Имя файла сценария: \"$0\"."
# Для текущего каталога добавит ./
echo "Имя файла сценария: \"`basename $0`\"."
# Добавит путь к имени файла (см. 'basename')

echo

if [ -n "$1" ]                # Проверяемая переменная заключена в
кавычки.
then
    echo "Параметр #1: $1"    # необходимы кавычки для экранирования
символа #
fi

if [ -n "$2" ]
then
    echo "Параметр #2: $2"
fi

if [ -n "$3" ]
then
    echo "Параметр #3: $3"
```

```
fi

# ...

if [ -n "${10}" ] # Параметры, следующие за $9 должны заключаться в
фигурные скобки
then
    echo "Параметр #10: ${10}"
fi

echo "-----"
echo "Все аргументы командной строки: "$*"

if [ $# -lt "$MINPARAMS" ]
then
    echo
    echo "Количество аргументов командной строки должно быть не менее
$MINPARAMS !"
fi


echo

exit 0
```

Скобочная нотация позиционных параметров дает довольно простой способ обращения к *последнему* аргументу, переданному в сценарий из командной строки. Такой способ подразумевает использование [косвенной адресации](#).

```
args=$#          # Количество переданных аргументов.
lastarg=${!args} # Обратите внимание: lastarg=${!$#} неприменимо.
```

В сценарии можно предусмотреть различные варианты развития событий, в зависимости от имени сценария. Для этого сценарий должен проанализировать аргумент \$0 -- имя файла сценария. Это могут быть и имена символических ссылок на файл сценария.

 Если сценарий ожидает передачи аргументов в командной строке, то при их отсутствии он получит "пустые" переменные, что может вызвать нежелательный побочный эффект. Один из способов борьбы с подобными ошибками -- добавить дополнительный символ в обеих частях операции присваивания, где используются аргументы командной строки.

```
variable1_=$1_
# Это предотвратит появление ошибок, даже при отсутствии входного
аргумента.

critical_argument01=$variable1_

# Дополнительные символы всегда можно "убрать" позднее.
# Это может быть сделано примерно так:
```

```

variable1=${variable1_/_/} # Побочный эффект возникает только если имя
переменной
                                # $variable1_ будет начинаться с символа
" _".
# Здесь используется один из вариантов подстановки параметров,
обсуждаемых в Главе 9.
# Отсутствие шаблона замены приводит к удалению.

# Более простой способ заключается
#+ в обычной проверке наличия позиционного параметра.
if [ -z $1 ]
then
    exit $POS_PARAMS_MISSING
fi

---
```

Пример 4-6. wh, **whois** выяснение имени домена

```

#!/bin/bash

# Команда 'whois domain-name' выясняет имя домена на одном из 3
серверов:
#
#           ripe.net, cw.net, radb.net

# Разместите этот скрипт под именем 'wh' в каталоге /usr/local/bin

# Требуемые символические ссылки:
# ln -s /usr/local/bin/wh /usr/local/bin/wh-ripe
# ln -s /usr/local/bin/wh /usr/local/bin/wh-cw
# ln -s /usr/local/bin/wh /usr/local/bin/wh-radb

if [ -z "$1" ]
then
    echo "Порядок использования: `basename $0` [domain-name]"
    exit 65
fi

case `basename $0` in
# Проверка имени скрипта и, соответственно, имени сервера
    "wh"           ) whois $1@whois.ripe.net;;
    "wh-ripe") whois $1@whois.ripe.net;;
    "wh-radb") whois $1@whois.radb.net;;
    "wh-cw"      ) whois $1@whois.cw.net;;
    *              ) echo "Порядок использования: `basename $0` [domain-
name]";;
esac

exit 0

---
```

Команда **shift** "сдвигает" позиционные параметры, в результате чего параметры "сдвигаются" на одну позицию влево.

\$1 <--- \$2, \$2 <--- \$3, \$3 <--- \$4, и т.д.

Прежний аргумент `$1` теряется, но аргумент `$0` (имя файла сценария) остается без изменений. Если вашему сценарию передается большое количество входных аргументов, то команда **shift** позволит вам получить доступ к аргументам, с порядковым номером больше 9, без использования фигурных скобок.

Пример 4-7. Использование команды shift

```
#!/bin/bash
# Использование команды 'shift' с целью перебора всех аргументов
командной строки.

# Назовите файл с этим сценарием, например "shft",
#+ и вызовите его с набором аргументов, например:
#      ./shft a b c def 23 skidoo

until [ -z "$1" ] # До тех пор пока не будут разобраны все входные
аргументы...
do
    echo -n "$1 "
    shift
done

echo                # Дополнительная пустая строка.

exit 0
```



Команда **shift** может применяться и к входным аргументам функций. См. [Пример 33-10](#).

Примечания

- [1] Аргумент `$0` устанавливается вызывающим процессом. В соответствии с соглашениями, этот параметр содержит имя файла скрипта. См. страницы руководства для **execv** (man execv).

Назад
Переменные Bash не имеют
типа

[К началу](#)
Наверх

[Вперед](#)
Кавычки

Спонсоры:



При поддержке
inferno solutions*

Хостинг:



Hoster.ru
хостинг провайдер

[Закладки на сайте](#)
[Проследить за страницей](#)

Created 1996-2022 by **Maxim Chirkov**
[Добавить](#), [Поддержать](#), [Вебмастеру](#)