**9.3. Подстановка параметров**

**Работа с переменными и/или подстановка их значений**

**${parameter}**

То же самое, что и *$parameter*, т.е. значение переменной *parameter*. В отдельных случаях, при возникновении неоднозначности интерпретации, корректно будет работать только такая форма записи: *${parameter}*.

Может использоваться для конкатенации (слияния) строковых переменных.

your\_id=${USER}-on-${HOSTNAME}

echo "$your\_id"

#

echo "Старый \$PATH = $PATH"

PATH=${PATH}:/opt/bin #Добавление /opt/bin в $PATH.

echo "Новый \$PATH = $PATH"

**${parameter-default}**, **${parameter:-default}**

Если параметр отсутствует, то используется значение по-умолчанию.

echo ${username-`whoami`}

# Вывод результата работы команды `whoami`, если переменная $username не установлена.

|  |  |
| --- | --- |
| Note | Формы записи *${parameter-default}* и *${parameter:-default}* в большинстве случаев можно считать эквивалентными. Дополнительный символ : имеет значение только тогда, когда *parameter* определен, но имеет "пустое" (null) значение. |

#!/bin/bash

username0=

# переменная username0 объявлена, но инициализирована "пустым" значением.

echo "username0 = ${username0-`whoami`}"

# Вывод после символа "=" отсутствует.

echo "username1 = ${username1-`whoami`}"

# Переменная username1 не была объявлена.

# Выводится имя пользователя, выданное командой `whoami`.

username2=

# переменная username2 объявлена, но инициализирована "пустым" значением.

echo "username2 = ${username2:-`whoami`}"

# Выводится имя пользователя, выданное командой `whoami`, поскольку

#+здесь употребляется конструкция ":-" , а не "-".

exit 0

*Параметры по-умолчанию* очень часто находят применение в случаях, когда сценарию необходимы какие либо входные аргументы, передаваемые из командной строки, но такие аргументы не были переданы.

DEFAULT\_FILENAME=generic.data

filename=${1:-$DEFAULT\_FILENAME}

# Если имя файла не задано явно, то последующие операторы будут работать

#+ с файлом "generic.data".

#

см. так же [Пример 3-4](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c301.html#EX58), [Пример 28-2](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c13041.html#EX73) и [Пример A-7](http://www.opennet.ru/docs/RUS/bash_scripting_guide/a14477.html#COLLATZ).

Сравните этот подход с [методом списков *and list*, для задания параметров командной строки по-умолчанию](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c12716.html#ANDDEFAULT) .

**${parameter=default}**, **${parameter:=default}**

Если значения параметров не задананы явно, то они принимают значения по-умолчанию.

Оба метода задания значений по-умолчанию до определенной степени идентичны. Символ : имеет значение только когда *$parameter* был инициализирован "пустым" (null) значением, [[1]](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html" \l "FTN.AEN4520) как показано выше.

echo ${username=`whoami`}

# Переменная "username" принимает значение, возвращаемое командой `whoami`.

**${parameter+alt\_value}**, **${parameter:+alt\_value}**

Если параметр имеет какое либо значение, то используется **alt\_value**, иначе -- null ("пустая" строка).

Оба варианта до определенной степени идентичны. Символ : имеет значение только если *parameter* объявлен и "пустой", см. ниже.

echo "###### \${parameter+alt\_value} ########"

echo

a=${param1+xyz}

echo "a = $a" # a =

param2=

a=${param2+xyz}

echo "a = $a" # a = xyz

param3=123

a=${param3+xyz}

echo "a = $a" # a = xyz

echo

echo "###### \${parameter:+alt\_value} ########"

echo

a=${param4:+xyz}

echo "a = $a" # a =

param5=

a=${param5:+xyz}

echo "a = $a" # a =

# Вывод отличается от a=${param5+xyz}

param6=123

a=${param6+xyz}

echo "a = $a" # a = xyz

**${parameter?err\_msg}**, **${parameter:?err\_msg}**

Если parameter инициализирован, то используется его значение, в противном случае -- выводится err\_msg.

Обе формы записи можно, до определенной степени, считать идентичными. Символ : имеет значение только когда *parameter* инициализирован "пустым" значением, см. ниже.

**Пример 9-13. Подстановка параметров и сообщения об ошибках**

#!/bin/bash

# Проверка отдельных переменных окружения.

# Если переменная, к примеру $USER, не установлена,

#+ то выводится сообщение об ошибке.

: ${HOSTNAME?} ${USER?} ${HOME?} ${MAIL?}

echo

echo "Имя машины: $HOSTNAME."

echo "Ваше имя: $USER."

echo "Ваш домашний каталог: $HOME."

echo "Ваш почтовый ящик: $MAIL."

echo

echo "Если перед Вами появилось это сообщение,"

echo "то это значит, что все критические переменные окружения установлены."

echo

echo

# ------------------------------------------------------

# Конструкция ${variablename?} так же выполняет проверку

#+ наличия переменной в сценарии.

ThisVariable=Value-of-ThisVariable

# Обратите внимание, в строковые переменные могут быть записаны

#+ символы, которые запрещено использовать в именах переменных.

: ${ThisVariable?}

echo "Value of ThisVariable is $ThisVariable".

echo

echo

: ${ZZXy23AB?"Переменная ZZXy23AB не инициализирована."}

# Если ZZXy23AB не инициализирована,

#+ то сценарий завершается с сообщением об ошибке.

# Текст сообщения об ошибке можно задать свой.

# : ${ZZXy23AB?"Переменная ZZXy23AB не инициализирована."}

# То же самое: dummy\_variable=${ZZXy23AB?}

# dummy\_variable=${ZZXy23AB?"Переменная ZXy23AB не инициализирована."}

#

# echo ${ZZXy23AB?} >/dev/null

echo "Это сообщение не будет напечатано, поскольку сценарий завершится раньше."

HERE=0

exit $HERE # Сценарий завершит работу не здесь.

**Пример 9-14. Подстановка параметров и сообщение о "порядке использования"**

#!/bin/bash

# usage-message.sh

: ${1?"Порядок использования: $0 ARGUMENT"}

# Сценарий завершит свою работу здесь, если входные аргументы отсутствуют,

#+ со следующим сообщением.

# usage-message.sh: 1: Порядок использования: usage-message.sh ARGUMENT

echo "Эти две строки появятся, только когда задан аргумент в командной строке."

echo "Входной аргумент командной строки = \"$1\""

exit 0 # Точка выхода находится здесь, только когда задан аргумент командной строки.

# Проверьте код возврата в обеих случаях, с и без аргумента командной строки.

# Если аргумент задан, то код возврата будет равен 0.

# Иначе -- 1.

**Подстановка параметров и/или экспансия.** Следующие выражения могут служить дополнениями оператора **match** команды **expr**, применяемой к строкам (см. [Пример 12-6](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x6646.html#EX45)). Как правило, они используются при разборе имен файлов и каталогов.

**Длина переменной / Удаление подстроки**

**${#var}**

**String length** (число символов в переменной $var). В случае [массивов](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c12790.html#ARRAYREF), команда **${#array}** возвращает длину первого элемента массива.

|  |  |
| --- | --- |
| Note | Исключения:   * **${#\*}** и **${#@}** возвращает *количество аргументов (позиционных параметров)*. * Для массивов, **${#array[\*]}** и **${#array[@]}** возвращает количество элементов в массиве. |

**Пример 9-15. Длина переменной**

#!/bin/bash

# length.sh

E\_NO\_ARGS=65

if [ $# -eq 0 ] # Для работы скрипта необходим хотя бы один входной параметр.

then

echo "Вызовите сценарий с одним или более параметром командной строки."

exit $E\_NO\_ARGS

fi

var01=abcdEFGH28ij

echo "var01 = ${var01}"

echo "Length of var01 = ${#var01}"

echo "Количество входных параметров = ${#@}"

echo "Количество входных параметров = ${#\*}"

exit 0

**${var#Pattern}**, **${var##Pattern}**

Удаляет из переменной $var наименьшую/наибольшую подстроку, совпадающую с шаблоном $Pattern. Поиск ведется с начала строки $var.

Пример использования из [Пример A-8](http://www.opennet.ru/docs/RUS/bash_scripting_guide/a14477.html#DAYSBETWEEN):

# Функцмя из сценария "days-between.sh".

# Удаляет нули, стоящие в начале аргумента-строки.

strip\_leading\_zero () # Ведущие нули, которые согут находиться в номере дня/месяца,

# лучше удалить

val=${1#0} # В противном случае Bash будет интерпретировать числа

return $val # как восьмеричные (POSIX.2, sect 2.9.2.1).

}

Другой пример:

echo `basename $PWD` # Имя текущего рабочего каталога.

echo "${PWD##\*/}" # Имя текущего рабочего каталога.

echo

echo `basename $0` # Имя файла-сценария.

echo $0 # Имя файла-сценария.

echo "${0##\*/}" # Имя файла-сценария.

echo

filename=test.data

echo "${filename##\*.}" # data

# Расширение файла.

**${var%Pattern}**, **${var%%Pattern}**

Удаляет из переменной $var наименьшую/наибольшую подстроку, совпадающую с шаблоном $Pattern. Поиск ведется с конца строки $var.

Bash [версии 2](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c13936.html#BASH2REF) имеет ряд дополнительных возможностей.

**Пример 9-16. Поиск по шаблону в подстановке параметров**

#!/bin/bash

# Поиск по шаблону в операциях подстановки параметров # ## % %%.

var1=abcd12345abc6789

pattern1=a\*c # \* (символ шаблона), означает любые символы между a и c.

echo

echo "var1 = $var1" # abcd12345abc6789

echo "var1 = ${var1}" # abcd12345abc6789 (альтернативный вариант)

echo "Число символов в ${var1} = ${#var1}"

echo "pattern1 = $pattern1" # a\*c (между 'a' и 'c' могут быть любые символы)

echo

echo '${var1#$pattern1} =' "${var1#$pattern1}" # d12345abc6789

# Наименьшая подстрока, удаляются первые 3 символа abcd12345abc6789

^^^^^^ |-|

echo '${var1##$pattern1} =' "${var1##$pattern1}" # 6789

# Наибольшая подстрока, удаляются первые 12 символов abcd12345abc6789

# ^^^^^^ |----------|

echo; echo

pattern2=b\*9 # все, что между 'b' и '9'

echo "var1 = $var1" # abcd12345abc6789

echo "pattern2 = $pattern2"

echo

echo '${var1%pattern2} =' "${var1%$pattern2}" # abcd12345a

# Наименьшая подстрока, удаляются последние 6 символов abcd12345abc6789

# ^^^^^^^^^ |----|

echo '${var1%%pattern2} =' "${var1%%$pattern2}" # a

# Наибольшая подстрока, удаляются последние 12 символов abcd12345abc6789

# ^^^^^^^^^ |-------------|

# Запомните, # и ## используются для поиска с начала строки,

# % и %% используются для поиска с конца строки.

echo

exit 0

**Пример 9-17. Изменение расширений в именах файлов:**

#!/bin/bash

# rfe

# ---

# Изменение расширений в именах файлов.

#

# rfe old\_extension new\_extension

#

# Пример:

# Изменить все расширения \*.gif в именах файлов на \*.jpg, в текущем каталоге

# rfe gif jpg

ARGS=2

E\_BADARGS=65

if [ $# -ne "$ARGS" ]

then

echo "Порядок использования: `basename $0` old\_file\_suffix new\_file\_suffix"

exit $E\_BADARGS

fi

for filename in \*.$1

# Цикл прохода по списку имен файлов, имеющих расширение равное первому аргументу.

do

mv $filename ${filename%$1}$2

# Удалить первое расширение и добавить второе,

done

exit 0

**Подстановка значений переменных / Замена подстроки**

Эти конструкции перекочевали в Bash из *ksh*.

**${var:pos}**

Подстанавливается значение переменной *var*, начиная с позиции *pos*.

**${var:pos:len}**

Подстанавливается значение переменной *var*, начиная с позиции *pos*, не более *len* символов. См. [Пример A-16](http://www.opennet.ru/docs/RUS/bash_scripting_guide/a14477.html#PW).

**${var/Pattern/Replacement}**

Первое совпадение с шаблоном *Pattern*, в переменной *var* замещается подстрокой *Replacement*.

Если подстрока *Replacement* отсутствует, то найденное совпадение будет удалено.

**${var//Pattern/Replacement}**

**Глобальная замена.** Все найденные совпадения с шаблоном *Pattern*, в переменной *var*, будут замещены подстрокой *Replacement*.

Как и в первом случае, если подстрока *Replacement* отсутствует, то все найденные совпадения будут удалены.

**Пример 9-18. Поиск по шаблону при анализе произвольных строк**

#!/bin/bash

var1=abcd-1234-defg

echo "var1 = $var1"

t=${var1#\*-\*}

echo "var1 (все, от начала строки по первый символ \"-\", включительно, удаляется) = $t"

# t=${var1#\*-} то же самое,

#+ поскольку оператор # ищет кратчайшее совпадение,

#+ а \* соответствует любым предшествующим символам, включая пустую строку.

# (Спасибо S. C. за разъяснения.)

t=${var1##\*-\*}

echo "Если var1 содержит \"-\", то возвращается пустая строка... var1 = $t"

t=${var1%\*-\*}

echo "var1 (все, начиная с последнего \"-\" удаляется) = $t"

echo

# -------------------------------------------

path\_name=/home/bozo/ideas/thoughts.for.today

# -------------------------------------------

echo "path\_name = $path\_name"

t=${path\_name##/\*/}

echo "Из path\_name удален путь к файлу = $t"

# В данном случае, тот эе эффект можно получить так: t=`basename $path\_name`

# t=${path\_name%/}; t=${t##\*/} более общее решение,

#+ но имеет некоторые ограничения.

# Если $path\_name заканчивается символом перевода строки, то `basename $path\_name` не будет работать,

#+ но для данного случая вполне применимо.

# (Спасибо S.C.)

t=${path\_name%/\*.\*}

# Тот же эффект дает t=`dirname $path\_name`

echo "Из path\_name удалено имя файла = $t"

# Этот вариант будет терпеть неудачу в случаях: "../", "/foo////", # "foo/", "/".

# Удаление имени файла, особенно когда его нет,

#+ использование dirname имеет свои особенности.

# (Спасибо S.C.)

echo

t=${path\_name:11}

echo "Из $path\_name удалены первые 11 символов = $t"

t=${path\_name:11:5}

echo "Из $path\_name удалены первые 11 символов, выводится 5 символов = $t"

echo

t=${path\_name/bozo/clown}

echo "В $path\_name подстрока \"bozo\" заменена на \"clown\" = $t"

t=${path\_name/today/}

echo "В $path\_name подстрока \"today\" удалена = $t"

t=${path\_name//o/O}

echo "В $path\_name все символы \"o\" переведены в верхний регистр, = $t"

t=${path\_name//o/}

echo "Из $path\_name удалены все символы \"o\" = $t"

exit 0

**${var/#Pattern/Replacement}**

Если в переменной *var* найдено совпадение с *Pattern*, причем совпадающая подстрока расположена в начале строки (префикс), то оно заменяется на *Replacement*. Поиск ведется с начала строки

**${var/%Pattern/Replacement}**

Если в переменной *var* найдено совпадение с *Pattern*, причем совпадающая подстрока расположена в конце строки (суффикс), то оно заменяется на *Replacement*. Поиск ведется с конца строки

**Пример 9-19. Поиск префиксов и суффиксов с заменой по шаблону**

#!/bin/bash

# Поиск с заменой по шаблону.

v0=abc1234zip1234abc # Начальное значение переменной.

echo "v0 = $v0" # abc1234zip1234abc

echo

# Поиск совпадения с начала строки.

v1=${v0/#abc/ABCDEF} # abc1234zip1234abc

# |-|

echo "v1 = $v1" # ABCDE1234zip1234abc

# |---|

# Поиск совпадения с конца строки.

v2=${v0/%abc/ABCDEF} # abc1234zip123abc

# |-|

echo "v2 = $v2" # abc1234zip1234ABCDEF

# |----|

echo

# ----------------------------------------------------

# Если совпадение находится не с начала/конца строки,

#+ то замена не производится.

# ----------------------------------------------------

v3=${v0/#123/000} # Совпадение есть, но не в начале строки.

echo "v3 = $v3" # abc1234zip1234abc

# ЗАМЕНА НЕ ПРОИЗВОДТСЯ!

v4=${v0/%123/000} # Совпадение есть, но не в конце строки.

echo "v4 = $v4" # abc1234zip1234abc

# ЗАМЕНА НЕ ПРОИЗВОДТСЯ!

exit 0

**${!varprefix\*}**, **${!varprefix@}**

Поиск по шаблону всех, ранее объявленных переменных, имена которых начинаются с *varprefix*.

xyz23=whatever

xyz24=

a=${!xyz\*} # Подстановка имен объявленных переменных, которые начинаются с "xyz".

echo "a = $a" # a = xyz23 xyz24

a=${!xyz@} # То же самое.

echo "a = $a" # a = xyz23 xyz24

# Эта возможность была добавлена в Bash, в версии 2.04.

**Примечания**

|  |  |
| --- | --- |
| [[1]](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html" \l "AEN4520) | Если $parameter "пустой",в неинтерактивных сценариях, то это будет приводить к завершению с [кодом возврата 127](http://www.opennet.ru/docs/RUS/bash_scripting_guide/a14876.html#EXITCODESREF) ("command not found"). |

|  |  |  |
| --- | --- | --- |
| [Назад](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4171.html) | [К началу](http://www.opennet.ru/docs/RUS/bash_scripting_guide/index.html) | [Вперед](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4704.html) |
| Работа со строками | [Наверх](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c3270.html) | Объявление переменных: **declare** и **typeset** |

# 9.2. Работа со строками

Bash поддерживает на удивление большое количество операций над строками. К сожалению, этот раздел Bash испытывает недостаток унификации. Одни операции являются подмножеством операций [подстановки параметров](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html#PARAMSUBREF), а другие -- совпадают с функциональностью команды UNIX -- [expr](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x6646.html#EXPRREF). Это приводит к противоречиям в синтаксисе команд и перекрытию функциональных возможностей, не говоря уже о возникающей путанице.

**Длина строки**

${#string}

expr length $string

expr "$string" : '.\*'

stringZ=abcABC123ABCabc

echo ${#stringZ} # 15

echo `expr length $stringZ` # 15

echo `expr "$stringZ" : '.\*'` # 15

**Пример 9-10. Вставка пустых строк между параграфами в текстовом файле**

#!/bin/bash

# paragraph-space.sh

# Вставка пустых строк между параграфами в текстовом файле.

# Порядок использования: $0 <FILENAME

MINLEN=45 # Возможно потребуется изменить это значение.

# Строки, содержащие количество символов меньшее, чем $MINLEN

#+ принимаются за последнюю строку параграфа.

while read line # Построчное чтение файла от начала до конца...

do

echo "$line" # Вывод строки.

len=${#line}

if [ "$len" -lt "$MINLEN" ]

then echo # Добавление пустой строки после последней строки параграфа.

fi

done

exit 0

**Длина подстроки в строке (подсчет совпадающих символов ведется с начала строки)**

expr match "$string" '$substring'

где $substring -- [регулярное выражение](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c11895.html#REGEXREF).

expr "$string" : '$substring'

где $substring -- регулярное выражение.

stringZ=abcABC123ABCabc

# |------|

echo `expr match "$stringZ" 'abc[A-Z]\*.2'` # 8

echo `expr "$stringZ" : 'abc[A-Z]\*.2'` # 8

**Index**

expr index $string $substring

Номер позиции первого совпадения в $string c первым символом в $substring.

stringZ=abcABC123ABCabc

echo `expr index "$stringZ" C12` # 6

# позиция символа C.

echo `expr index "$stringZ" 1c` # 3

# символ 'c' (в #3 позиции) совпал раньше, чем '1'.

Эта функция довольно близка к функции strchr() в языке C.

**Извлечение подстроки**

${string:position}

Извлекает подстроку из $string, начиная с позиции $position.

Если строка $string -- "\*" или "@", то извлекается [позиционный параметр](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c3270.html#POSPARAMREF) (аргумент), [[1]](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4171.html" \l "FTN.AEN4257) с номером $position.

${string:position:length}

Извлекает $length символов из $string, начиная с позиции $position.

stringZ=abcABC123ABCabc

# 0123456789.....

# Индексация начинается с 0.

echo ${stringZ:0} # abcABC123ABCabc

echo ${stringZ:1} # bcABC123ABCabc

echo ${stringZ:7} # 23ABCabc

echo ${stringZ:7:3} # 23A

# Извлекает 3 символа.

# Возможна ли индексация с "правой" стороны строки?

echo ${stringZ:-4} # abcABC123ABCabc

# По-умолчанию выводится полная строка.

# Однако . . .

echo ${stringZ:(-4)} # Cabc

echo ${stringZ: -4} # Cabc

# Теперь выводится правильно.

# Круглые скобки или дополнительный пробел "экранируют" параметр позиции.

# Спасибо Dan Jacobson, за разъяснения.

Если $string -- "\*" или "@", то извлекается до $length позиционных параметров (аргументов), начиная с $position.

echo ${\*:2} # Вывод 2-го и последующих аргументов.

echo ${@:2} # То же самое.

echo ${\*:2:3} # Вывод 3-х аргументов, начиная со 2-го.

expr substr $string $position $length

Извлекает $length символов из $string, начиная с позиции $position.

stringZ=abcABC123ABCabc

# 123456789......

# Индексация начинается с 1.

echo `expr substr $stringZ 1 2` # ab

echo `expr substr $stringZ 4 3` # ABC

expr match "$string" '\($substring\)'

Находит и извлекает первое совпадение $substring в $string, где $substring -- это [регулярное выражение](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c11895.html#REGEXREF).

expr "$string" : '\($substring\)'

Находит и извлекает первое совпадение $substring в $string, где $substring -- это регулярное выражение.

stringZ=abcABC123ABCabc

# =======

echo `expr match "$stringZ" '\(.[b-c]\*[A-Z]..[0-9]\)'` # abcABC1

echo `expr "$stringZ" : '\(.[b-c]\*[A-Z]..[0-9]\)'` # abcABC1

echo `expr "$stringZ" : '\(.......\)'` # abcABC1

# Все вышеприведенные операции дают один и тот же результат.

expr match "$string" '.\*\($substring\)'

Находит и извлекает первое совпадение $substring в $string, где $substring -- это регулярное выражение. Поиск начинается с конца $string.

expr "$string" : '.\*\($substring\)'

Находит и извлекает первое совпадение $substring в $string, где $substring -- это регулярное выражение. Поиск начинается с конца $string.

stringZ=abcABC123ABCabc

# ======

echo `expr match "$stringZ" '.\*\([A-C][A-C][A-C][a-c]\*\)'` # ABCabc

echo `expr "$stringZ" : '.\*\(......\)'` # ABCabc

**Удаление части строки**

${string#substring}

Удаление самой короткой, из найденных, подстроки $substring в строке $string. Поиск ведется с начала строки

${string##substring}

Удаление самой длинной, из найденных, подстроки $substring в строке $string. Поиск ведется с начала строки

stringZ=abcABC123ABCabc

# |----|

# |----------|

echo ${stringZ#a\*C} # 123ABCabc

# Удаление самой короткой подстроки.

echo ${stringZ##a\*C} # abc

# Удаление самой длинной подстроки.

${string%substring}

Удаление самой короткой, из найденных, подстроки $substring в строке $string. Поиск ведется с конца строки

${string%%substring}

Удаление самой длинной, из найденных, подстроки $substring в строке $string. Поиск ведется с конца строки

stringZ=abcABC123ABCabc

# ||

# |------------|

echo ${stringZ%b\*c} # abcABC123ABCa

# Удаляется самое короткое совпадение. Поиск ведется с конца $stringZ.

echo ${stringZ%%b\*c} # a

# Удаляется самое длинное совпадение. Поиск ведется с конца $stringZ.

**Пример 9-11. Преобразование графических файлов из одного формата в другой, с изменением имени файла**

#!/bin/bash

# cvt.sh:

# Преобразование всех файлов в заданном каталоге,

#+ из графического формата MacPaint, в формат "pbm".

# Используется утилита "macptopbm", входящая в состав пакета "netpbm",

#+ который сопровождается Brian Henderson (bryanh@giraffe-data.com).

# Netpbm -- стандартный пакет для большинства дистрибутивов Linux.

OPERATION=macptopbm

SUFFIX=pbm # Новое расширение файла.

if [ -n "$1" ]

then

directory=$1 # Если каталог задан в командной строке при вызове сценария

else

directory=$PWD # Иначе просматривается текущий каталог.

fi

# Все файлы в каталоге, имеющие расширение ".mac", считаются файлами

#+ формата MacPaint.

for file in $directory/\* # Подстановка имен файлов.

do

filename=${file%.\*c} # Удалить расширение ".mac" из имени файла

#+ ( с шаблоном '.\*c' совпадают все подстроки

#+ начинающиеся с '.' и заканчивающиеся 'c',

$OPERATION $file > "$filename.$SUFFIX"

# Преобразование с перенаправлением в файл с новым именем

rm -f $file # Удаление оригинального файла после преобразования.

echo "$filename.$SUFFIX" # Вывод на stdout.

done

exit 0

# Упражнение:

# --------

# Сейчас этот сценарий конвертирует \*все\* файлы в каталоге

# Измените его так, чтобы он конвертировал \*только\* те файлы,

#+ которые имеют расширение ".mac".

**Замена подстроки**

${string/substring/replacement}

Замещает первое вхождение $substring строкой $replacement.

${string//substring/replacement}

Замещает все вхождения $substring строкой $replacement.

stringZ=abcABC123ABCabc

echo ${stringZ/abc/xyz} # xyzABC123ABCabc

# Замена первой подстроки 'abc' строкой 'xyz'.

echo ${stringZ//abc/xyz} # xyzABC123ABCxyz

# Замена всех подстрок 'abc' строкой 'xyz'.

${string/#substring/replacement}

Подстановка строки $replacement вместо $substring. Поиск ведется с начала строки $string.

${string/%substring/replacement}

Подстановка строки $replacement вместо $substring. Поиск ведется с конца строки $string.

stringZ=abcABC123ABCabc

echo ${stringZ/#abc/XYZ} # XYZABC123ABCabc

# Поиск ведется с начала строки

echo ${stringZ/%abc/XYZ} # abcABC123ABCXYZ

# Поиск ведется с конца строки

## 9.2.1. Использование awk при работе со строками

В качестве альтернативы, Bash-скрипты могут использовать средства [awk](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x14802.html#AWKREF) при работе со строками.

**Пример 9-12. Альтернативный способ извлечения подстрок**

#!/bin/bash

# substring-extraction.sh

String=23skidoo1

# 012345678 Bash

# 123456789 awk

# Обратите внимание на различия в индексации:

# Bash начинает индексацию с '0'.

# Awk начинает индексацию с '1'.

echo ${String:2:4} # с 3 позиции (0-1-2), 4 символа

# skid

# В эквивалент в awk: substr(string,pos,length).

echo | awk '

{ print substr("'"${String}"'",3,4) # skid

}

'

# Передача пустого "echo" по каналу в awk, означает фиктивный ввод,

#+ делая, тем самым, ненужным предоставление имени файла.

exit 0

## 9.2.2. Дальнейшее обсуждение

Дополнительную информацию, по работе со строками, вы найдете в разделе [Section 9.3](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html) и в [секции](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x6646.html#EXPEXTRSUB), посвященной команде [expr](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x6646.html#EXPRREF). Примеры сценариев:

1. [Пример 12-6](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x6646.html#EX45)
2. [Пример 9-15](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html#LENGTH)
3. [Пример 9-16](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html#PATTMATCHING)
4. [Пример 9-17](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html#RFE)
5. [Пример 9-19](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html#VARMATCH)

### Примечания

|  |  |
| --- | --- |
| [[1]](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4171.html" \l "AEN4257) | Применяется к аргументам командной строки или входным параметрам [функций](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c12483.html#FUNCTIONREF). |

|  |  |  |
| --- | --- | --- |
| [Назад](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c3270.html) | [К началу](http://www.opennet.ru/docs/RUS/bash_scripting_guide/index.html) | [Вперед](http://www.opennet.ru/docs/RUS/bash_scripting_guide/x4462.html) |
| К вопросу о переменных | [Наверх](http://www.opennet.ru/docs/RUS/bash_scripting_guide/c3270.html) |  |