

# **Отчёта по лабораторной работе №10**

**НКНбд-02-21**

Акондзо Жордание Лади Гаэл

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Ход работы:</b>	<b>6</b>
<b>3</b>	<b>Выводы</b>	<b>11</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>12</b>

# List of Figures

2.1	Написание программы . . . . .	6
2.2	Приписывавание права и исполнение программы . . . . .	6
2.3	Написание программы . . . . .	7
2.4	Приписывавание права и исполнение программы . . . . .	7
2.5	Написание программы . . . . .	8
2.6	Приписывавание права и исполнение программы . . . . .	8
2.7	Написание программы . . . . .	9
2.8	Приписывавание права и исполнение программы . . . . .	10

## **Список таблиц**

# 1 Цель работы

Цель данной работы — Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

## 2 Ход работы:

1. Написал скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.(рис. 2.1, 2.2 )

```
#!/bin/bash
mkdir ~/backup
cp script01.sh ~/backup/script01.sh
cd ~
tar -cvzf backup.tar.gz backup/
```

Рис. 2.1: Написание программы

```
[zlakondzo@jordani ~]$ vi script01.sh
[zlakondzo@jordani ~]$ ~/script01.sh
bash: /home/zlakondzo/script01.sh: Permission non accordée
[zlakondzo@jordani ~]$ chmod +x script01.sh
[zlakondzo@jordani ~]$ ~/script01.sh
backup/
backup/script01.sh
[zlakondzo@jordani ~]$
```

Рис. 2.2: Приписывавание права и исполнение программы

2. Написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов. (рис. 2.3, 2.4 )

```

for arg in $@
do
    echo $arg
done
~
~
~
~

```

Рис. 2.3: Написание программы

```

[zlakondzo@jordani ~]$ vi script02.sh
[zlakondzo@jordani ~]$ chmod +x script02.sh
[zlakondzo@jordani ~]$ ~/script02.sh 1 2 3 4 5 6 7 8 9 10 11
1
2
3
4
5
6
7
8
9
10
11
[zlakondzo@jordani ~]$ ~/script02.sh 1 2 3 4 5 6 7 8 9 10 11 12
1
2
3
4
5
6
7
8
9
10
11
12
[zlakondzo@jordani ~]$ vi script02.sh

```

Рис. 2.4: Приписывание права и исполнение программы

3. Написал командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога. (рис. 2.5, 2.6 )

```

for A in *
do if test -d $A
  then echo $A: is a directory
  else echo -n $A: is a file and
    if test -w $A
    then echo writeable
    elif test -r $A
    then echo readable
    else echo neither readable nor writeable
  fi
fi
done

```

Рис. 2.5: Написание программы

```

[zlakondzo@jordani ~]$ vi script03.sh
[zlakondzo@jordani ~]$ chmod +x script03.sh
[zlakondzo@jordani ~]$ ~/script03.sh
australia: is a directory
backup: is a directory
backup.tar.gz: is a file andwriteable
bin: is a directory
Bureau: is a directory
Documents: is a directory
file.txt: is a file andwriteable
go: is a directory
Images: is a directory
install-tl-20220428: is a directory
install-tl-unx.tar.gz: is a file andwriteable
/home/zlakondzo/script03.sh: ligne 2 : test: <invalid : opérateur binaire attendu
<invalid path>: is a file and/home/zlakondzo/script03.sh: ligne 5 : test: <invalid : opérateur b
inaire attendu
/home/zlakondzo/script03.sh: ligne 7 : test: <invalid : opérateur binaire attendu
neither readable nor writeable
/home/zlakondzo/script03.sh: ligne 2 : test: <invalid : opérateur binaire attendu
<invalid path>.layout: is a file and/home/zlakondzo/script03.sh: ligne 5 : test: <invalid : opér
ateur binaire attendu
/home/zlakondzo/script03.sh: ligne 7 : test: <invalid : opérateur binaire attendu
neither readable nor writeable
lab07.sh: is a file andwriteable
lab07.sh~: is a file andwriteable
letters: is a directory
logfile.txt: is a file andwriteable
may: is a file andwriteable
memos: is a directory
misk: is a directory
Modèles: is a directory
monthly: is a directory
Musique: is a directory
my_os: is a file andreadable
play: is a directory
Public: is a directory

```

Рис. 2.6: Приписывавание права и исполнение программы

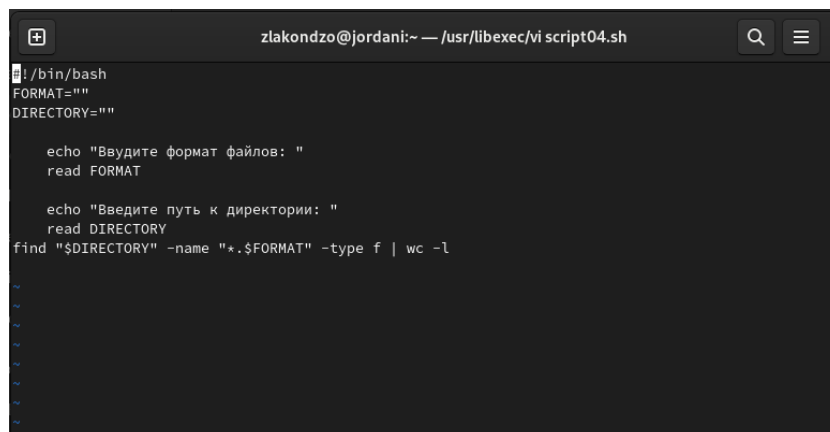


```

memos: is a directory
misk: is a directory
Modèles: is a directory
monthly: is a directory
Musique: is a directory
my_os: is a file andreadable
play: is a directory
Public: is a directory
reports: is a directory
Screenshots: is a directory
script01.sh: is a file andwriteable
script02.sh: is a file andwriteable
script03.sh: is a file andwriteable
ski.plases: is a directory
Téléchargements: is a directory
text.txt: is a file andwriteable
Vidéos: is a directory
work: is a directory
World: is a directory
Загрузки: is a directory
[zlakondzo@jordani ~]$ vi script03.sh
[zlakondzo@jordani ~]$

```

4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки. (рис. 2.7, 2.8 )



```

zlakondzo@jordani:~ — /usr/libexec/vi script04.sh
#!/bin/bash
FORMAT=""
DIRECTORY=""

echo "Введите формат файлов: "
read FORMAT

echo "Введите путь к директории: "
read DIRECTORY
find "$DIRECTORY" -name ".*$FORMAT" -type f | wc -l

~
~
~
~
~
~
~

```

Рис. 2.7: Написание программы

```
zlakondzo@jordani:~  
[zlakondzo@jordani ~]$ vi script04.sh  
[zlakondzo@jordani ~]$ chmod +x script04.sh  
[zlakondzo@jordani ~]$ ~/script04.sh  
Введите формат файлов:  
png  
Введите путь к директории:  
Screenshots  
35  
[zlakondzo@jordani ~]$ ~/script04.sh  
Введите формат файлов:  
txt  
Введите путь к директории:  
/home/zlakondzo  
192  
[zlakondzo@jordani ~]$ ~/script04.sh  
Введите формат файлов:  
pdf  
Введите путь к директории:  
/home/work/  
find: '/home/work/': Aucun fichier ou dossier de ce type  
0  
[zlakondzo@jordani ~]$ ~/script04.sh  
Введите формат файлов:  
pdf  
Введите путь к директории:  
/home/zlakondzo/work  
20  
[zlakondzo@jordani ~]$
```

Рис. 2.8: Приписывавание права и исполнение программы

## 3 Выводы

Во время выполнения работы, мы изучили основы программирования в оболочке ОС UNIX/Linux. Научились писать небольшие командные файлы.

## 4 Контрольные вопросы

### 1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой более гибкие;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей — оболочка Борна, но с дополнительными возможностями.

### 2. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных

программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке `bash`. В других оболочках большинство команд будет совпадать с описанными ниже.

### 3. Как определяются переменные и массивы в языке программирования `bash`?

Командный процессор `bash` обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда: `mark=/usr/andy/bin`. присваивает значение строки символов `/usr/andy/bin` переменной `mark` типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол `$`. Например, команда: `mv afile ${mark}`. переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}`. Например, использование команд: `b=/tmp/andy-ls -l myfile > blssudoapt — getinstalltexlive — luatex.ls/tmp/andy — ls, ls — l > bls` приведёт к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то её значением будет символ пробела. Оболочка `bash` позволяет работать с массивами. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, `set -A states Delaware Michigan “New Jersey”`. Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента.

#### 4. Каково назначение операторов let и read?

Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение - это единичный терм (term), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате. Этот формат — radix#number, где radix (основание системы счисления) - любое число не более 26. Для большинства команд основания систем счисления это - 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток (%). Команда let берет два операнда и присваивает их переменной.

#### 5. Какие арифметические операции можно применять в языке программирования bash?

Оператор Синтаксис Результат ! !expr Если expr равно 0, возвращает 1; иначе 0 != expr1 !=expr2 Если expr1 не равно expr2, возвращает 1; иначе 0 %expr1% expr2 Возвращает остаток от деления expr1 на expr2 %= var=%expr Присваивает остаток от деления var на expr переменной var & expr1&expr2 Возвращает побитовое AND выражений expr1 и expr2 && expr1&&expr2 Если и expr1 и expr2 не равны нулю, возвращает 1; иначе 0 &= var &= expr Присваивает var побитовое AND переменных var и выражения expr \* expr1 \* expr2 Умножает expr1 на expr2 = var = expr Умножает expr на значение var и присваивает результат переменной var + expr1 + expr2 Складывает expr1 и expr2 += var += expr Складывает expr со значением var и результат присваивает var - -expr Операция отрицания expr (называется унарный минус) - expr1 - expr2 Вычитает expr2 из expr1 -= var -=expr Вычитает expr из значения var и присваивает результат var / expr / expr2 Делит expr1 на expr2 /= var /= expr Делит var на expr и присваивает результат var < expr1 < expr2. Если expr1 меньше, чем expr2, возвращает 1, иначе возвращает 0 « expr1« expr2 Сдвигает expr1 влево на expr2 бит «= var «= expr Побитовый сдвиг влево значения var на expr <= expr1 <= expr2 Если

$\text{exp1}$  меньше, или равно  $\text{exp2}$ , возвращает 1; иначе возвращает 0  $\text{var} = \text{exp}$   
 Присваивает значение  $\text{exp}$  переменной  $\text{var}$   $\text{var} == \text{exp1} == \text{exp2}$  Если  $\text{exp1}$  равно  $\text{exp2}$ .  
 Возвращает 1; иначе возвращает 0  $\text{var} > \text{exp1} > \text{exp2}$  1. если  $\text{exp1}$  больше, чем  $\text{exp2}$ ;  
 иначе 0  $\text{var} >= \text{exp1} >= \text{exp2}$  1 если  $\text{exp1}$  больше, или равно  $\text{exp2}$ ; иначе 0  $\text{var} \gg \text{exp} \gg \text{exp2}$   
 Сдвигает  $\text{exp1}$  вправо на  $\text{exp2}$  бит  $\text{var} \gg= \text{exp}$  Побитовый сдвиг вправо значения  
 $\text{var}$  на  $\text{exp}$   $\text{var} \wedge \text{exp1} \wedge \text{exp2}$  Исключающее OR выражений  $\text{exp1}$  и  $\text{exp2}$   $\text{var} \wedge= \text{exp}$   
 Присваивает  $\text{var}$  побитовое исключающее OR  $\text{var}$  и  $\text{exp}$   $\text{var} | \text{exp1} | \text{exp2}$  Побитовое  
 OR выражений  $\text{exp1}$  и  $\text{exp2}$   $\text{var} |= \text{exp}$  Присваивает  $\text{var}$  «исключающее OR» пе-  
 ременной  $\text{var}$  и выражения  $\text{exp}$   $\text{var} || \text{exp1} || \text{exp2}$  1 если или  $\text{exp1}$  или  $\text{exp2}$  являются  
 нену- левыми значениями; иначе 0  $\sim \text{exp}$  Побитовое дополнение до  $\text{exp}$ .

## 6. Условия оболочки `bash`, в двойные скобки —(( )).

облегчения программирования можно записывать условия оболочки `bash` в  
 двойные скобки — (( )). Можно присваивать результаты условных выражений пе-  
 ременным, также как и исполь- зовать результаты арифметических вычислений  
 в качестве условий. Хорошим примером сказанного является выполнение неко-  
 торого действия, одновременно декрементируя некоторое значение. например:  
`$ let x=5`

`$ while`

`(( x-=1 ))`

`do`

`something`

`done`

## 7. Какие стандартные имена переменных Вам известны?

Имя переменной (идентификатор) — это строка символов, которая отличает  
 эту переменную от других объектов программы (идентифицирует переменную

в программе). При задании имен переменным нужно соблюдать следующие правила: § первым символом имени должна быть буква. Остальные символы — буквы и цифры (прописные и строчные буквы различаются). Можно использовать символ «\_»; § в имени нельзя использовать символ «.»; § число символов в имени не должно превышать 255; § имя переменной не должно совпадать с зарезервированными (служебными) словами языка. Var1, PATH, trash, mon, day, PS1, PS2 Другие стандартные переменные: –HOME — имя домашнего каталога пользователя. Если команда cd вводится без аргументов, то происходит переход в каталог, указанный в этой переменной. –IFS — последовательность символов, являющихся разделителями в командной строке. Это символы пробел, табуляция и перевод строки(new line). –MAIL — командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение You have mail (у Вас есть почта). –TERM — тип используемого терминала. –LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему. В командном процессоре существует еще несколько стандартных переменных. Значение всех переменных можно посмотреть с помощью команды set.

## **8. Что такое метасимволы?**

Такие символы, как ' < > \* ? | " & являются метасимволами и имеют для командного процессора специальный смысл.

## **9. Как экранировать метасимволы?**

Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который, в свою очередь, является метасимволом. Для



экранирования группы метасимволов, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме \$, ', , ". Например, `-echo` выведет на экран символ, `-echo ab'|'cd` выдаст строку `ab|cd`.

#### **10. Как создавать и запускать командные файлы?**

Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде `bash командный_файл [аргументы]` Чтобы не вводить каждый раз последовательности символов `bash`, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `chmod +x имя_файла` Теперь можно вызывать свой командный файл на выполнение просто, вводя его имя с терминала так, как будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит ее интерпретацию.

#### **11. Как определяются функции в языке программирования bash?**

Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями: `-f` — перечисляет определенные на текущий момент функции; `-ft` — при последующем вызове функции иницирует ее трассировку; `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек; `-fu` — означает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции.

#### **12. Каким образом можно выяснить, является файл каталогом или обычным файлом?**

ls -lrt Если есть d, то является файл каталогом

### 13. Каково назначение команд set, typeset и unset?

Используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например, set -A states Delaware Michigan "New Jersey" Далее можно сделать добавление в массив, например, states[49]=Alaska . Индексация массивов начинается с нулевого элемента. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set. Наиболее распространенным является сокращение, избавляющееся от слова let в программах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте typeset -i для объявления и присвоения переменной, и при последующем использовании она становится целой. Или можете использовать ключевое слово integer (псевдоним для typeset -l) и объявлять переменные целыми. Таким образом, выражения типа  $x=y+z$  воспринимаются как арифметические. Группу команд можно объединить в функцию. Для этого существует ключевое слово function , после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды unset с флагом -f . Команда typeset имеет четыре опции для работы с функциями: – -f — перечисляет определенные на текущий момент функции; – -ft — при последующем вызове функции инициализирует ее трассировку; – -fx — экспортирует все перечисленные функции в любые дочерние программы оболочек; – -fu — означает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную FPATH , отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции. В переменные month и day будут считаны соответствующие значения, введенные с клавиатуры, а переменная trash нужна для того, чтобы отобразить всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды unset.

#### 14. Как передаются параметры в командные файлы?

Символ `$` является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов `$i`, где  $0 < i < 10$ , вместо нее будет осуществлена подстановка значения параметра с порядковым номером  $i$ , т.е. аргумента командного файла с порядковым номером  $i$ . Использование комбинации символов `$0` приводит к подстановке вместо нее имени данного командного файла. Примере: пусть к командному файлу `where` имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1`. Если Вы введете с терминала команду: `where andy`, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, на терминал будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал не будет выведено ничего. Команда `grep` производит контекстный поиск в тексте, поступающем со стандартного ввода, для нахождения в этом тексте строк, содержащих последовательности символов, переданные ей в качестве аргументов, и выводит результаты своей работы на стандартный вывод. В этом примере команда `grep` используется как фильтр, обеспечивающий ввод со стандартного ввода и вывод всех строк, содержащих последовательность символов `andy`, на стандартный вывод. В ходе интерпретации этого файла командным процессором вместо комбинации символов `$1` осуществляется подстановка значения первого и единственного параметра `andy`. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, то на терминале. Вы увидите примерно следующее: `$ where andy andy ttyG Jan 14 09:12 $` Определим функцию, которая изменяет каталог и печатает список файлов: `$ function clist { > cd $1 > ls > }`. Теперь при вызове команды `clist` каталог будет изменен каталог и выведено его содержимое.

## 15. Назовите специальные переменные языка **bash** и их назначение.

- \$\* — отображается вся командная строка или параметры оболочки;
- \$? — код завершения последней выполненной команды;
- \$\$ — уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
- #! — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
- \$- — значение флагов командного процессора;
- \${#} — возвращает целое число — количество слов, которые были результатом \$;
- \${#name} — возвращает целое значение длины строки в переменной name;
- \${name[n]} — обращение к n-ному элементу массива;
- \${name[\*]} — перечисляет все элементы массива, разделенные пробелом;
- \${name[@]} — то же самое, но позволяет учитывать символы пробелы в самих переменных;
- \${name:-value} — если значение переменной name не определено, то оно будет заменено на указанное value;
- \${name:value} — проверяется факт существования переменной;
- \${name=value} — если name не определено, то ему присваивается значение value;
- \${name?value} — останавливает выполнение, если имя переменной не определено, и выводит value, как сообщение об ошибке;
- \${name+value} — это выражение работает противоположно \${name-value}. Если переменная определена, то подставляется value;
- \${name#pattern} — представляет значение переменной name с удаленным самым коротким левым образцом (pattern);
- \${#name[\*]} и \${#name[@]} — эти выражения возвращают количество элементов в массиве name.

\$# вместо нее будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на вып