

# **Программирование в командном процессоре ОС UNIX. Командные файлы**

**Отчёт по лабораторной работе №11**

Мурашко В.В.

# Содержание

<b>1</b>	<b>Теоретическое введение</b>	<b>5</b>
<b>2</b>	<b>Цель работы</b>	<b>6</b>
<b>3</b>	<b>Задание</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Выводы</b>	<b>13</b>
<b>6</b>	<b>Библиография</b>	<b>14</b>
<b>7</b>	<b>Контрольные вопросы</b>	<b>15</b>

## **Список таблиц**

## Список иллюстраций

4.1	Справка команды архивации . . . . .	8
4.2	Создание файла и дача ему права . . . . .	9
4.3	Скрипт 1 . . . . .	9
4.4	Скрипт в домашней папке . . . . .	9
4.5	Скрипт 2 . . . . .	10
4.6	Вывод всех переданных аргументов . . . . .	10
4.7	Командный файл . . . . .	11
4.8	Информация . . . . .	11
4.9	Скрипт 3 . . . . .	12
4.10	Вычисление количества файлов . . . . .	12

# 1 Теоретическое введение

## Командные процессоры (оболочки)

Командные процессоры или оболочки – это программы, позволяющие пользователю взаимодействовать с компьютером. Их можно рассматривать как настоящие интерпретируемые языки, которые воспринимают команды пользователя и обрабатывают их. Поэтому командные процессоры также называют интерпретаторами команд. На языках оболочек можно писать программы и выполнять их подобно любым другим программам. UNIX обладает большим количеством оболочек. Наиболее популярными являются следующие четыре оболочки:

- оболочка Борна (Bourne) – первоначальная командная оболочка UNIX: базовый, но полный набор функций;
- C-оболочка – добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд;
- оболочка Корна – напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software)

## 2 Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

## 3 Задание

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в моём домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать 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.
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.

## 4 Выполнение лабораторной работы

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

```
TAR(1)                                GNU TAR Manual

NAME
    tar - an archiving utility

SYNOPSIS
    Traditional usage
        tar {A|c|d|r|t|u|x}[GnSkUWOmpsMBiajJzZhPlRvwo] [ARG...]

    UNIX-style usage
        tar -A [OPTIONS] ARCHIVE ARCHIVE

        tar -c [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -d [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]

        tar -r [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -u [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

    GNU-style usage
```

Рис. 4.1: Справка команды архивации



```
vvmurashko1@dk8n81 ~ $ man tar
vvmurashko1@dk8n81 ~ $ touch script.sh
vvmurashko1@dk8n81 ~ $ chmod +x script.sh
```

Рис. 4.2: Создание файла и дача ему права

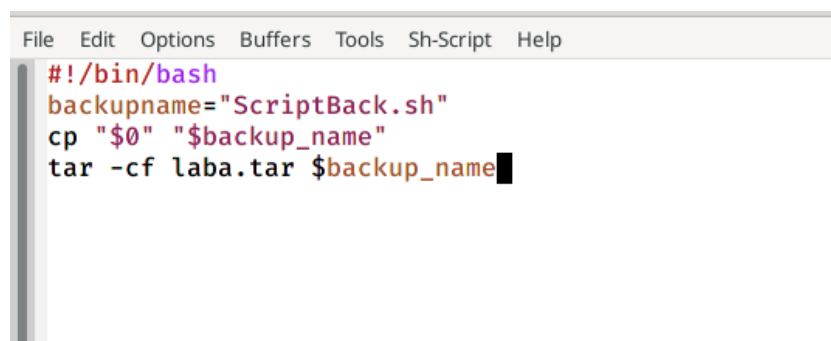


Рис. 4.3: Скрипт 1

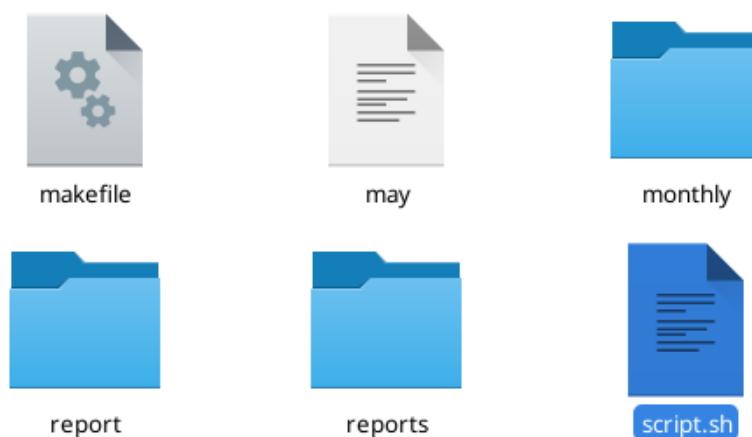
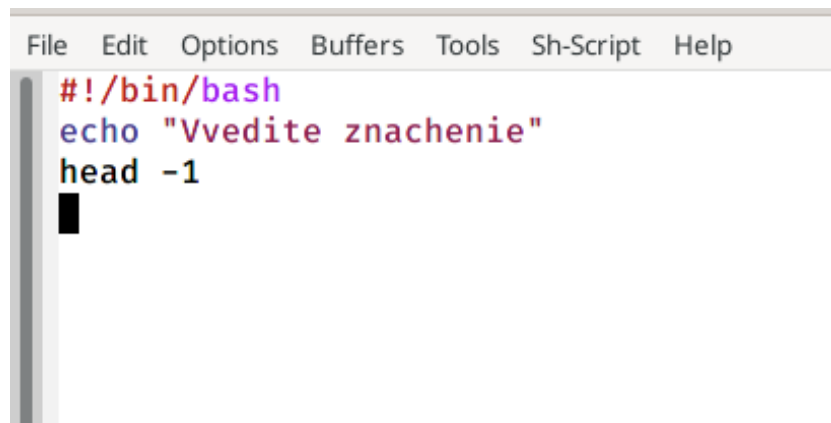


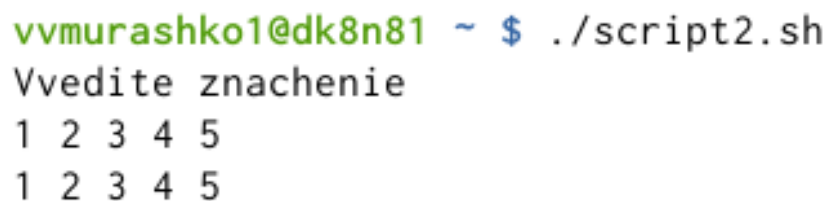
Рис. 4.4: Скрипт в домашней папке

2. Я написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.

A screenshot of a text editor window with a menu bar containing 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The editor contains a shell script with three lines: `#!/bin/bash`, `echo "Vvedite znachenie"`, and `head -1`. A black cursor is positioned at the end of the third line.

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
echo "Vvedite znachenie"
head -1
█
```

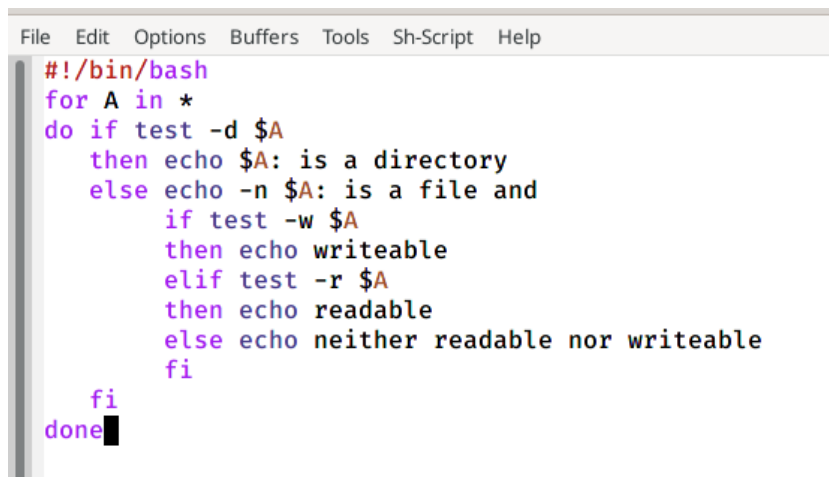
Рис. 4.5: Скрипт 2

A terminal window showing the execution of a script. The prompt is `vvmurashko1@dk8n81 ~ $`. The command `./script2.sh` has been entered. The output consists of the text `Vvedite znachenie` followed by two identical lines of `1 2 3 4 5`.

```
vvmurashko1@dk8n81 ~ $ ./script2.sh
Vvedite znachenie
1 2 3 4 5
1 2 3 4 5
```

Рис. 4.6: Вывод всех переданных аргументов

3. Я написала 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.

A screenshot of a terminal window with a menu bar at the top containing 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The terminal displays a shell script starting with a shebang line, followed by a loop that iterates over all files in the current directory. For each file, it checks if it's a directory. If not, it checks if it's writeable, readable, or neither. The script ends with 'done'.

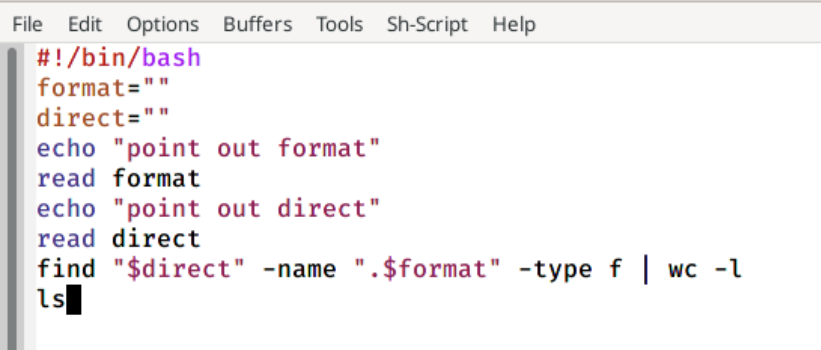
```
#!/bin/bash
for A in *
do if test -d $A
then echo $A: is a directory
else echo -n $A: is a file and
if test -w $A
then echo writeable
elif test -r $A
then echo readable
else echo neither readable nor writeable
fi
fi
done
```

Рис. 4.7: Командный файл

```
vvmurashko1@dk8n81 ~ $ ./file.sh
asdfg: is a file andwriteable
asdfg.asm: is a file andwriteable
asdfg.lst: is a file andwriteable
asdfg.map: is a file andwriteable
asdfg.o: is a file andwriteable
australia: is a directory
conf.txt: is a file andwriteable
feathers: is a file andwriteable
file.sh: is a file andwriteable
file.sh~: is a file andwriteable
file.txt: is a file andwriteable
games: is a directory
GNUstep: is a directory
lab02: is a directory
lab03: is a directory
lab03-1: is a file andwriteable
lab03-1.asm: is a file andwriteable
lab03-1.lst: is a file andwriteable
lab03-1.map: is a file andwriteable
lab03-1.o: is a file andwriteable
```

Рис. 4.8: Информация

4. Я написала 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.



```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
format=""
direct=""
echo "point out format"
read format
echo "point out direct"
read direct
find "$direct" -name ".$format" -type f | wc -l
ls
```

Рис. 4.9: Скрипт 3

```
vvmurashko1@dk8n81 ~ $ ./script3.sh
point out format
.sh
point out direct
/home/vvmurashko1/
find: '/home/vvmurashko1/': Нет такого файла или каталога
0
asdfg      australia  file.txt   lab03-1    lab03a
asdfg.asm  conf.txt   games      lab03-1.asm lab03b
asdfg.lst  feathers  GNUstep    lab03-1.lst lab04
asdfg.map  file.sh    lab02      lab03-1.map lab05
asdfg.o    file.sh~   _ lab03     lab03-1.o   lab06
```

Рис. 4.10: Вычисление количества файлов

## 5 Выводы

Я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.

## 6 Библиография

<https://infopedia.su/24x10498.html>

[https://esystem.rudn.ru/pluginfile.php/1142517/mod\\_resource/content/2/008-lab\\_shell\\_prog\\_1.pdf](https://esystem.rudn.ru/pluginfile.php/1142517/mod_resource/content/2/008-lab_shell_prog_1.pdf)

## 7 Контрольные вопросы

1. 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, содержащая базовый, но при этом полный набор функций;

- C-оболочка (или csh) — надстройка над оболочкой Борна, использующая подобный синтаксис команд с возможностью сохранения истории выполнения команд;

- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;

- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

2. 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.

3. 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 `.`, `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/andyls -l myfile > blssudoapt — getinstalltexlive — luatexls/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. 5. 6. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (*term*), обычно целочисленный. Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7.

Команда `let` также расширяет другие выражения `let`, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения.

Команда `let` не ограничена простыми арифметическими выражениями.

Команда `read` позволяет читать значения переменных со стандартного ввода:

```
echo "Please enter Month and Day of Birth ?"
```

```
read mon day trash
```

В переменные `mon` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать её.

7. – `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, например, 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

8. 9. Такие символы, как `' < > * ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл. 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 нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме `$, ', , "`.

10. 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 её интерпретацию.

11. 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 её трассировку; `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек; `-fu` — обозначает 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. `ls -lrt` Если есть `d`, то является файл каталогом

13. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Удалить функцию можно с помощью команды `unset` с флагом `-f`.

Команда `typeset` имеет четыре опции для работы с функциями:

- `-f` — перечисляет определённые на текущий момент функции;
- `-ft` — при последующем вызове функции иницирует её трассировку;
- `-fx` — экспортирует все перечисленные функции в любые дочерние программы

мы

оболочек;

– `-fu` — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове

оболочка просматривает переменную `FPATH`, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции.

14. 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. Рассмотрим это на 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. – `$*` — 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`-му элементу массива;

– `${name[*]}` — перечисляет все элементы массива, разделённые 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 удалённым самым коротким левым образцом (`pattern`);
- `${#name[*]}` и `${#name[@]}` — эти выражения возвращают количество элементов в массиве `name`.