

Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №11

дисциплина: *Операционные системы*

Студент: ГАБРИЭЛЬ ТЬЕРРИ

Группа: НКНбд 01-20

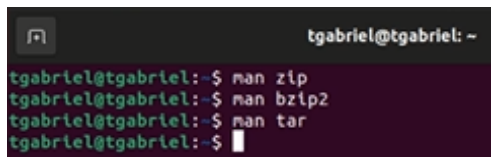
МОСКВА 2021 г.

### Цель работы:

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

### Ход работы:

1. Для начала я изучил команды архивации, используя команды «man zip», «man bzip2», «man tar» (Рисунки 1-4).



```
tgabriel@tgabriel: ~  
tgabriel@tgabriel:~$ man zip  
tgabriel@tgabriel:~$ man bzip2  
tgabriel@tgabriel:~$ man tar  
tgabriel@tgabriel:~$
```

Синтаксис команды zip для архивации файла: zip [опции] [имя файла.zip] [файлы или папки, которые будем архивировать] Синтаксис команды zip для разархивации/распаковки файла: unzip [опции] [файл\_архива.zip] [файлы] -x [исключить] -d [папка]

```
tgabriel@tgabriel: -
ZIP(1) General Commands Manual ZIP(1)

NAME
    zip - package and compress (archive) files

SYNOPSIS
    zip [-aABcdDeEffghjklLMoqrRSTuvVwXyz!@$] [--longoption ...]
        [-b path] [-n suffixes] [-t date] [-tt date] [zipfile [file
        ...]] [-xl list]

    zipcloak (see separate man page)

    zipnote (see separate man page)

    zipsplit (see separate man page)

    Note: Command line processing in zip has been changed to
    support long options and handle all options and arguments
    more consistently. Some old command lines that depend on
    command line inconsistencies may no longer work.

DESCRIPTION
    zip is a compression and file packaging utility for Unix,
    VMS, MSDOS, OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh,
    Amiga, and Acorn RISC OS. It is analogous to a combination
    of the Unix commands tar(1) and compress(1) and is compatible
    with PKZIP (Phil Katz's ZIP for MSDOS systems).

    A companion program (unzip(1)) unpacks zip archives. The zip
    and unzip(1) programs can work with archives produced by
    Manual page zip(1) line 1 (press h for help or q to quit)
```

Синтаксис команды bzip2 для архивации файла: bzip2 [опции] [имена файлов]

Синтаксис команды bzip2 для разархивации/распаковки файла: bunzip2 [опции] [архивы.bz2]

```
tgabriel@tgabriel: -
bzip2(1) General Commands Manual bzip2(1)

NAME
    bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
    bzip2recover - recovers data from damaged bzip2 files

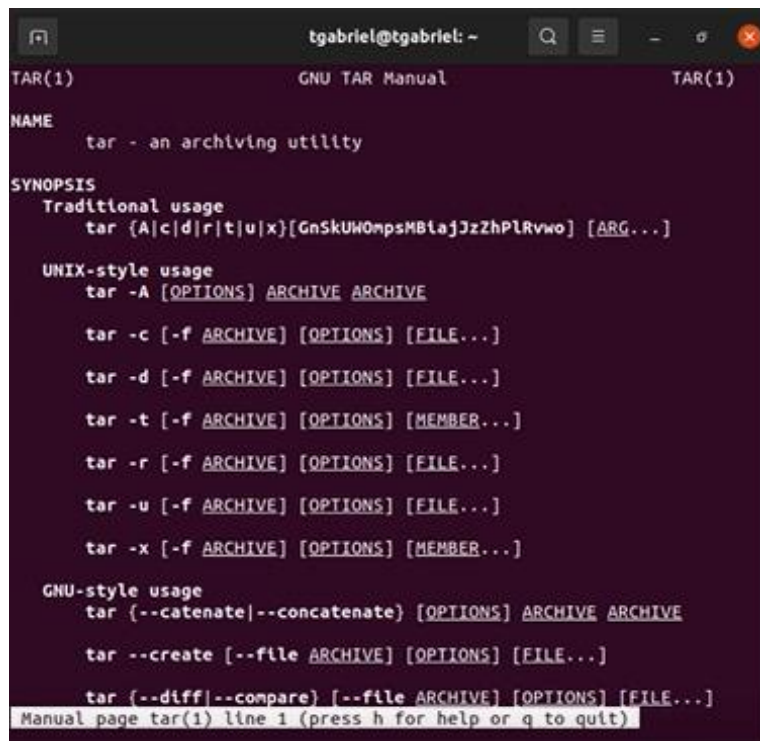
SYNOPSIS
    bzip2 [ -cdfkqstzVL123456789 ] [ filenames ... ]
    bzip2 [ -h|--help ]
    bunzip2 [ -fkvsVL ] [ filenames ... ]
    bunzip2 [ -h|--help ]
    bzip2 [ -s ] [ filenames ... ]
    bzip2recover filename

DESCRIPTION
    bzip2 compresses files using the Burrows-Wheeler block sort-
    ing text compression algorithm, and Huffman coding. Compres-
    sion is generally considerably better than that achieved by
    more conventional LZ77/LZ78-based compressors, and approaches
    the performance of the PPM family of statistical compressors.

    The command-line options are deliberately very similar to
    those of GNU gzip, but they are not identical.

    bzip2 expects a list of file names to accompany the command-
    line flags. Each file is replaced by a compressed version of
    itself, with the name "original_name.bz2". Each compressed
    file has the same modification date, permissions, and, when
    Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Синтаксис команды tar для архивации файла: tar [опции] [архив.tar]  
[файлы\_для\_архивации] Синтаксис команды tar для разархивации/распаковки  
файла: tar [опции] [архив.tar]



```
tgabriel@tgabriel: ~
TAR(1) GNU TAR Manual TAR(1)
NAME
    tar - an archiving utility
SYNOPSIS
    Traditional usage
    tar {A|c|d|r|t|u|x}[GnSkUMpmsMBlaJjZzZhPlRvwo] [ARG...]

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

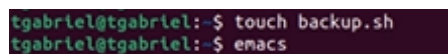
    tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
    tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
    tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
    tar -r [-f ARCHIVE] [OPTIONS] [FILE...]
    tar -u [-f ARCHIVE] [OPTIONS] [FILE...]
    tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

    GNU-style usage
    tar [--catenate|--concatenate] [OPTIONS] ARCHIVE ARCHIVE

    tar --create [--file ARCHIVE] [OPTIONS] [FILE...]

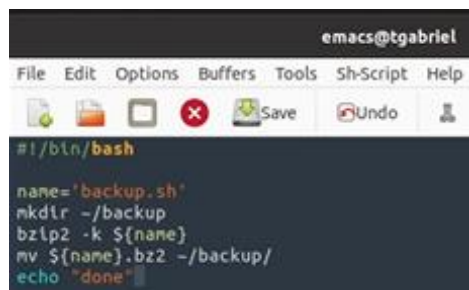
    tar [--diff|--compare] [--file ARCHIVE] [OPTIONS] [FILE...]
Manual page tar(1) line 1 (press h for help or q to quit)
```

Далее я создал файл, в котором буду писать первый скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (Рисунок 5).



```
tgabriel@tgabriel:~$ touch backup.sh
tgabriel@tgabriel:~$ emacs
```

После 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 (Рисунок 6). При написании скрипта использовал архиватор bzip2.



```
emacs@tgabriel
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
name='backup.sh'
mkdir -p /backup
bzip2 -k ${name}
mv ${name}.bz2 -/backup/
echo "done"
#1/bin/bash
```

Проверил работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod +x \*.sh»). Проверил, появился ли каталог

backup/, перейдя в него (команда «cd backup/»), посмотрел его содержимое (команда «ls») и просмотрел содержимое архива (команда «bunzip2 -c backup.sh.bz2») (Рисунки 7, 8). Скрипт работает корректно.

```
tgabriel@tgabriel:~$ ls
abc1      Desktop  ex3.txt  lab07.sh  my_os    ski.places
australia Documents ex4.txt  lab07.sh~ Pictures snap
backup.sh Downloads exit     may      play     Templates
backup.sh~ ex1.txt  feathers monthly  Public   Videos
conf.txt  ex2.txt  file.txt Music    reports  work
tgabriel@tgabriel:~$ chmod +x *.sh
tgabriel@tgabriel:~$ ./backup.sh
done
tgabriel@tgabriel:~$ cd backup/
tgabriel@tgabriel:~/backup$ ls
backup.sh.bz2
tgabriel@tgabriel:~/backup$
```

```
tgabriel@tgabriel:~/backup$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "done"
tgabriel@tgabriel:~/backup$
```

2. Создал файл, в котором буду писать второй скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch file2.sh» и «emacs &») (Рисунок 9).

```
tgabriel@tgabriel:~/backup$ cd ~
tgabriel@tgabriel:~$ touch file2.sh
tgabriel@tgabriel:~$ emacs
```

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 (Рисунок 10).

```
emacs@tgabriel
File Edit Options Buffers Tools Sh-Script Help
[Icons] Save Undo
#!/bin/bash
echo "Argument"
for a in $@
do echo $a
done
```

Проверил работу написанного скрипта (команды «./prog2.sh 0 1 2 3 4 5» и «./prog2.sh 0 1 2 3 4 5 6 7 8 9 10»), предварительно добавив для него право на выполнение (команда «chmod +x \*.sh»). Вводил аргументы, количество которых меньше 10 и больше 10 (Рисунок 11). Скрипт работает корректно.

```

tgabriel@tgabriel:~$ chmod +x *.sh
tgabriel@tgabriel:~$ ls
abc1      Desktop  ex4.txt  lab07.sh  Pictures  Templates
australia Documents exit     lab07.sh- play      Videos
backup    Downloads feathers may       Public    work
backup.sh ex1.txt  file2.sh monthly  reports
backup.sh- ex2.txt  file2.sh- Music    ski.places
conf.txt  ex3.txt  file.txt my_os    snap
tgabriel@tgabriel:~$ ./file2.sh 0 1 2 3 4 5
Argument
0
1
2
3
4
5
tgabriel@tgabriel:~$ ./file2.sh 0 1 2 3 4 5 6 7 8 9 10
Argument
0
1
2
3
4
5
6
7
8
9
10
tgabriel@tgabriel:~$ █

```

3. Создал файл, в котором буду писать третий скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch filels.sh» и «emacs &») (Рисунок 12).

```

tgabriel@tgabriel:~$ touch filels.sh
tgabriel@tgabriel:~$ emacs

```

Написал командный файл – аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога (Рисунок 13).



```
emacs@tgabriel
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
a="$1"
for i in $(ls $a)/*
do
    echo "$i"

    if test -f $i
    then echo "regular file"
    fi

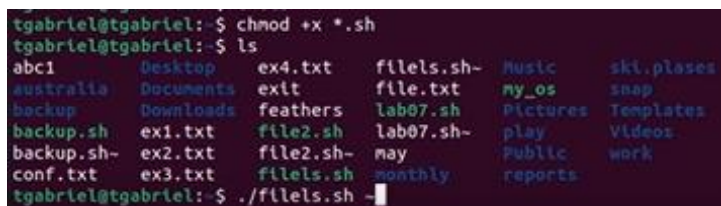
    if test -d $i
    then echo "catalog"
    fi

    if test -r $i
    then echo "reading allowed"
    fi

    if test -w $i
    then echo "registration allowed"
    fi

    if test -x $i
    then echo "execution allowed"
    fi
done
```

Далее проверил работу скрипта (команда «./filels.sh ~»), предварительно добавив для него право на выполнение (команда «chmod +x \*.sh») (Рисунок 14,15).



```
tgabriel@tgabriel:~$ chmod +x *.sh
tgabriel@tgabriel:~$ ls
abc1      Desktop  ex4.txt   filels.sh- Music    ski.places
australia Documents exit      file.txt  my_os    snap
backup    Downloads feathers lab07.sh  Pictures Templates
backup.sh ex1.txt  file2.sh lab07.sh- play     Videos
backup.sh- ex2.txt  file2.sh- may       Public    work
conf.txt  ex3.txt  filels.sh monthly   reports

tgabriel@tgabriel:~$ ./filels.sh ~
```



```
tgabriel@tgabriel: ~  
/home/tgabriel/reports  
catalog  
reading allowed  
registration allowed  
execution allowed  
/home/tgabriel/skl.plases  
catalog  
reading allowed  
registration allowed  
execution allowed  
/home/tgabriel/snap  
catalog  
reading allowed  
registration allowed  
execution allowed  
/home/tgabriel/Templates  
catalog  
reading allowed  
registration allowed  
execution allowed  
/home/tgabriel/Videos  
catalog  
reading allowed  
registration allowed  
execution allowed  
/home/tgabriel/work  
catalog  
reading allowed  
registration allowed  
execution allowed  
tgabriel@tgabriel:~$
```

4. Для четвертого скрипта также создал файл (команда «touch extensions.sh») и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (Рисунок 16).

```
tgabriel@tgabriel:~$ touch extensions.sh  
tgabriel@tgabriel:~$ emacs
```

Написал 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 (Рисунок 17).

```
emacs@tgabriel  
File Edit Options Buffers Tools Sh-Script Help  
Save Undo  
#!/bin/bash  
b="$1"  
shift  
for a in $@  
do  
    k=0  
    for i in ${b}/*.${a}  
    do  
        if test -f "$i"  
        then  
            let k=k+1  
        fi  
    done  
    echo "$k files are contained in the $b directory with the $a"  
done
```

Проверил работу написанного скрипта (команда «./extensions.sh ~ pdf sh txt doc»), предварительно добавив для него право на выполнение (команда «chmod +x \*.sh»), а также создав дополнительные файлы с разными расширениями (команда «touch file.pdf file1.doc file2.doc») (Рисунок 18).

```
tgabriel@tgabriel:~$ chmod +x *.sh
tgabriel@tgabriel:~$ ls
abc1      ex1.txt      file1.pdf    lab07.sh-   ski.places
australia ex2.txt      file2.doc    may         snap
backup    ex3.txt      file2.sh     monthly     Templates
backup.sh ex4.txt      file2.sh-    Music       Videos
backup.sh- exit        file3.jpg    my_os       work
conf.txt  extensions.sh filels.sh    Pictures
Desktop  extensions.sh- filels.sh-   play
Documents extensions.sh  file.txt    Public
Downloads feathers   lab07.sh    reports
tgabriel@tgabriel:~$ ./extensions.sh ~ pdf sh txt doc jpg
1 files are contained in the /home/tgabriel directory with the pdf
6 files are contained in the /home/tgabriel directory with the sh
6 files are contained in the /home/tgabriel directory with the txt
1 files are contained in the /home/tgabriel directory with the doc
1 files are contained in the /home/tgabriel directory with the jpg
tgabriel@tgabriel:~$
```

## Вывод:

В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.

## 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;
  - С-оболочка (или csh) – надстройка на оболочке Борна, использующая подобный синтаксис команд с возможностью сохранения истории выполнения команд;
  - оболочка Корна (или ksh) – напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
  - BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С 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 `$`. Например, команда `«mv afile ${mark}»` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. 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. Оболочка `bash` поддерживает встроенные арифметические функции. 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. 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, введённые с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить всю избыточно введённую информацию и игнорировать её.
5. В языке программирования `bash` можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%).
6. В (( )) можно записывать условия оболочки `bash`, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.
7. Стандартные переменные:
  - `PATH`: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной `PATH`, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.
  - `PS1` и `PS2`: эти переменные предназначены для отображения промптера командного процессора. `PS1` – это промптер командного процессора, по умолчанию его значение равно символу `$` или `#`. Если какая-то интерактивная

программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >.

- 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. Такие символы, как ' < > \* ? | " &, являются метасимволами и имеют для командного процессора специальный смысл.
  9. 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 \$, ' , , ". Например, – echo \* выведет на экран символ , – echo ab'|cd выведет на экран строку ab|\*cd.
  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, заключённых в фигурные скобки. Удалить функцию можно с помощью команды unset с флагом -f.
  12. Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами «test -f [путь до файла]» (для проверки, является ли обычным файлом) и «test -d [путь до файла]» (для проверки, является ли каталогом).
  13. Команду «set» можно использовать для вывода списка переменных окружения. В системах Ubuntu и Debian команда «set» также выведет список

функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «set | more». Команда «typeset» предназначена для наложения ограничений на переменные. Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.

14. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. 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 гделибо в командном файле комбинации символов \$i, где  $0 < i < 10$ , вместо неё 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 неё имени данного командного файла.
15. Специальные переменные:
  - \$\* – 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`.