

# **Отчет по лабораторной работе №11**

**Дисциплина: Операционные системы**

Морозова Анастасия Владимировна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	21

## **Список таблиц**

## Список иллюстраций

2.1	Команды архивации . . . . .	6
2.2	Команды zip . . . . .	7
2.3	Команды bzip2 . . . . .	7
2.4	Команды tar . . . . .	8
2.5	Открытие файла . . . . .	8
2.6	Написание скрипта . . . . .	9
2.7	Проверка работы скрипта . . . . .	9
2.8	Проверка работы скрипта . . . . .	9
2.9	Второй скрипт . . . . .	10
2.10	Обработка произвольного числа аргументов . . . . .	10
2.11	Проверка скрипта . . . . .	11
2.12	Третий скрипт . . . . .	11
2.13	Аналог команды ls . . . . .	12
2.14	Аналог команды ls . . . . .	12
2.15	Проверка скрипта . . . . .	13
2.16	Четвертый скрипт . . . . .	13
2.17	Вычисление кол-ва необходимых файлов . . . . .	14
2.18	Проверка скрипта . . . . .	14

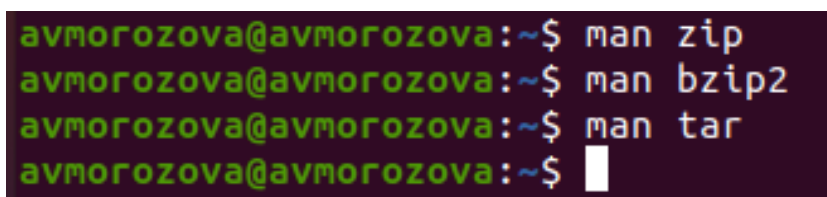
# 1 Цель работы

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

## 2 Выполнение лабораторной работы

1. Изучила команды архивации (команды `manzip`, `manbzip2`, `mantar`)(рис. - fig. 2.1)

- 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]` (рис. -fig. 2.2)
- Синтаксис команды `bzip2` для архивации файла:`bzip2 [опции] [имена файлов]`
- Синтаксис команды `bzip2` для разархивации/распаковки файла: `bunzip2[опции] [архивы.bz2]` (рис. -fig. 2.3)
- Синтаксис команды `tar` для архивации файла:`tar[опции][архив.tar][файлы_для_архивации]`
- Синтаксис команды `tar` для разархивации/распаковки файла:`tar[опции][архив.tar]` (рис. -fig. 2.4)

A screenshot of a terminal window with a dark background and green text. It shows four lines of commands entered at the prompt 'avmorofova@avmorofova:~\$'. The commands are 'man zip', 'man bzip2', 'man tar', and a blank line with a cursor. The output of the 'man' commands is not visible.

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

Рис. 2.1: Команды архивации

```

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 ...]] [-x! 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).

```

Рис. 2.2: Команды zip

```

bzip2(1)                              General Commands Manual          bzip2(1)

NAME

    bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
    bzcata - decompresses files to stdout
    bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS

    bzip2 [ -cdfkqstvzVL123456789 ] [ filenames ... ]
    bzip2 [ -h|--help ]
    bunzip2 [ -fkvsVL ] [ filenames ... ]
    bunzip2 [ -h|--help ]
    bzcata [ -s ] [ filenames ... ]
    bzcata [ -h|--help ]
    bzip2recover filename

DESCRIPTION

    bzip2 compresses files using the Burrows-Wheeler block sorting text
    compression algorithm, and Huffman coding. Compression 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

```

Рис. 2.3: Команды bzip2

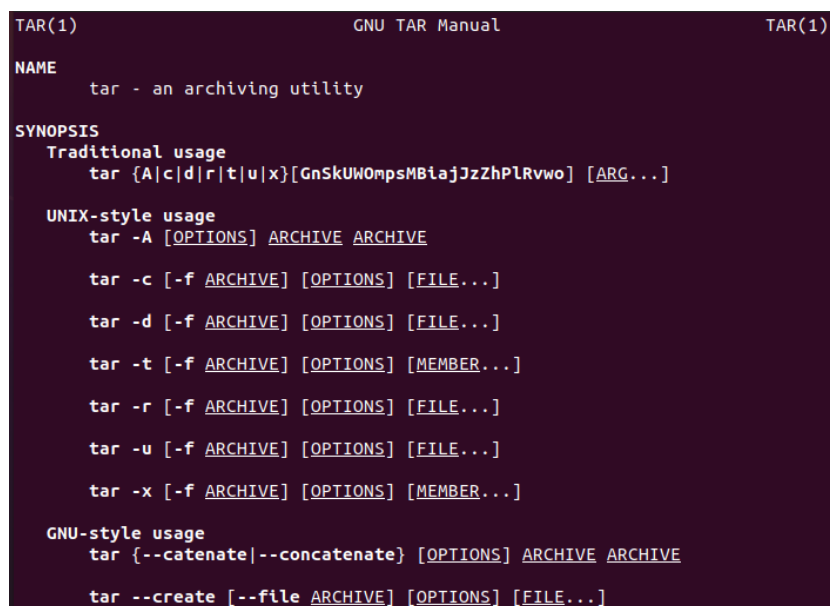


Рис. 2.4: Команды tar

2. Создаю файл, в котором буду писать первый скрипт, и открываю его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды touch backup.sh и emacs &) (рис. -fig. 2.5)

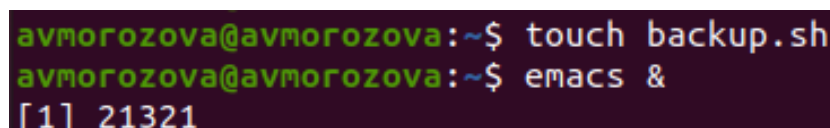
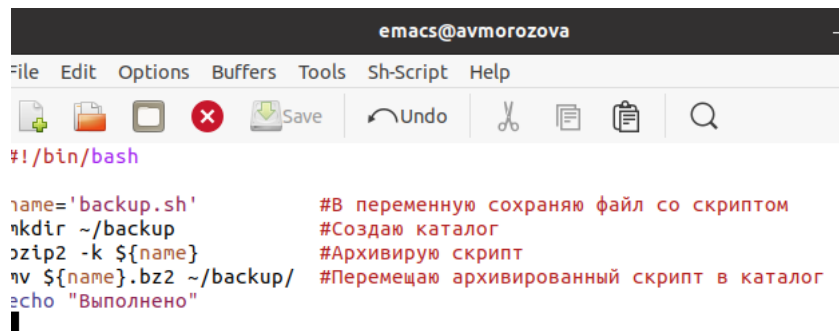


Рис. 2.5: Открытие файла

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 домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. При написании скрипта использовала архиватор bzip2 (рис. -fig. 2.6)



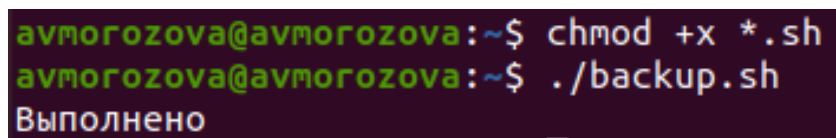


```
emacs@avmorofova
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash

name='backup.sh'      #В переменную сохраняю файл со скриптом
mkdir ~/backup        #Создаю каталог
bzip2 -k ${name}       #Архивирую скрипт
mv ${name}.bz2 ~/backup/ #Перемещаю архивированный скрипт в каталог
echo "Выполнено"
```

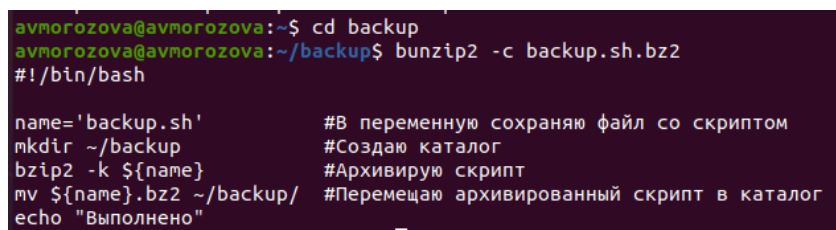
Рис. 2.6: Написание скрипта

Проверила работу скрипта (команда `./backup.sh`), предварительно добавив для него право на выполнение (команда `chmod +x *.sh`). Проверила, появился ли каталог `backup/`, перейдя в него (команда `cd backup/`), посмотрела его содержимое (команда `ls`) и просмотрела содержимое архива (команда `bunzip2 -c backup.sh.bz2`). Скрипт работает корректно. (рис. -fig. 2.7)(рис. -fig. 2.8)



```
avmorofova@avmorofova:~$ chmod +x *.sh
avmorofova@avmorofova:~$ ./backup.sh
Выполнено
```

Рис. 2.7: Проверка работы скрипта



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

name='backup.sh'      #В переменную сохраняю файл со скриптом
mkdir ~/backup        #Создаю каталог
bzip2 -k ${name}       #Архивирую скрипт
mv ${name}.bz2 ~/backup/ #Перемещаю архивированный скрипт в каталог
echo "Выполнено"
```

Рис. 2.8: Проверка работы скрипта

3. Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды `touch ex2.sh` и `emacs &`) (рис. -fig. 2.9)

```
avmorofova@avmorofova:~$ touch ex2.sh
avmorofova@avmorofova:~$ emacs &
[1] 22814
```

Рис. 2.9: Второй скрипт

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 (рис. -fig. 2.10)

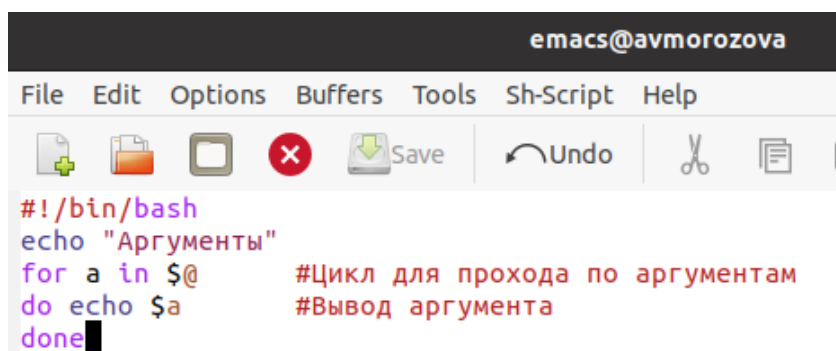


Рис. 2.10: Обработка произвольного числа аргументов

Проверила работу написанного скрипта (команды ./prog2.sh 0 1 2 3 4 и ./ex2.sh 0 1 2 3 4 5 6 7 8 9 10 11), предварительно добавив для него право на выполнение (команда chmod +x\*.sh). Вводила аргументы, количество которых меньше 10 и больше 10. Скрипт работает корректно. (рис. -fig. 2.11)

```
avmorofova@avmorofova:~$ chmod +x *.sh
avmorofova@avmorofova:~$ ./ex2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12 13
Аргументы
0
1
2
3
4
5
6
7
8
9
10
11
12
13
avmorofova@avmorofova:~$
```

Рис. 2.11: Проверка скрипта

4. Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды touch progl.sh и emacs &)(рис. -fig. 2.12)

```
avmorofova@avmorofova:~$ touch ex3.sh
avmorofova@avmorofova:~$ emacs &
```

Рис. 2.12: Третий скрипт

Написала 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 (рис. -fig. 2.13)(рис. -fig. 2.14)

```
#!/bin/bash
a="$1"
for i in ${a}/*
do
    echo "$i"

    if test -f $i
    then echo "Обычный файл"
    fi

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

    if test -r $i
    then echo "Чтение разрешено"
    fi

    if test -w $i
    then echo "Запись разрешена"
    fi

    if test -x $i
    then echo "Выполнение разрешено"
```

Рис. 2.13: Аналог команды ls

```
    fi
done
```

Рис. 2.14: Аналог команды ls

Далее проверила работу скрипта (команда `./proglsh.sh~`), предварительно добавив для него право на выполнение (команда `chmod +x*.sh`) (рис. -fig. 2.15)

```

avmorofozova@avmorofozova:~$ chmod +x *.sh
avmorofozova@avmorofozova:~$ ./ex3.sh ~
/home/avmorofozova/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/avmorofozova/backup.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/avmorofozova/backup.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/home/avmorofozova/Desktop
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/avmorofozova/Documents
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/avmorofozova/Downloads

```

Рис. 2.15: Проверка скрипта

5. Создала файл для четвертого скрипта (команда `touch format.sh`)и открыла его в редакторе `emacs`, используя клавиши «Ctrl-x» и «Ctrl-f» (команда `emacs &`). Переместила курсор в конец строки клавишей «Ctrl-e»(рис. -fig. 2.16)

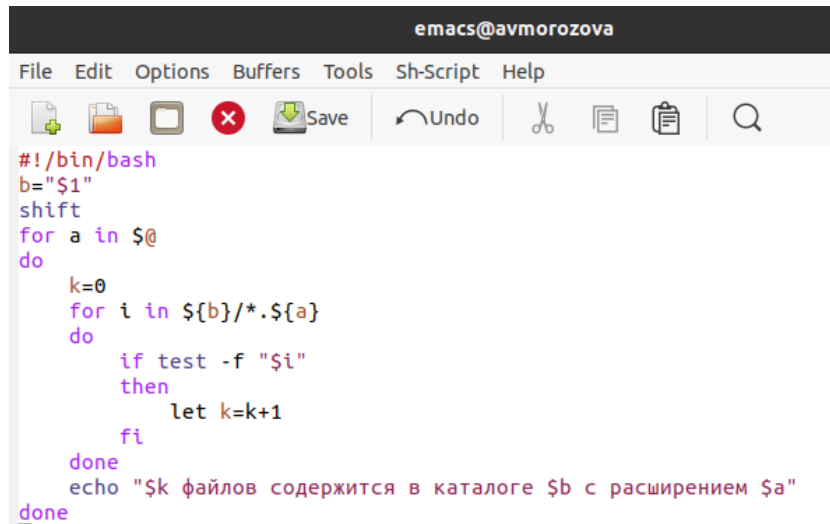
```

avmorofozova@avmorofozova:~$ touch ex4.sh
avmorofozova@avmorofozova:~$ emacs &
[3] 23393

```

Рис. 2.16: Четвертый скрипт

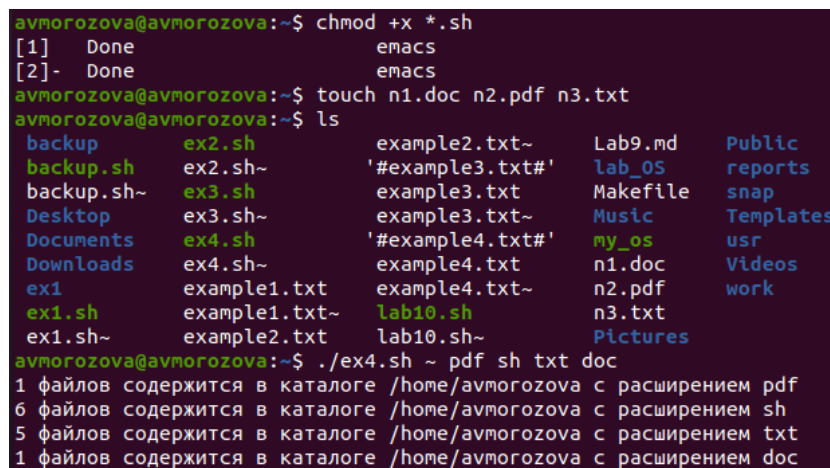
Написала командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdfи т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки (рис. -fig. 2.17)



```
#!/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 файлов содержится в каталоге $b с расширением $a"
done
```

Рис. 2.17: Вычисление кол-ва необходимых файлов

Проверила работу написанного скрипта (команда ./format.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»)(рис. -fig. 2.18)



```
avmorofova@avmorofova:~$ chmod +x *.sh
[1] Done emacs
[2]- Done emacs
avmorofova@avmorofova:~$ touch n1.doc n2.pdf n3.txt
avmorofova@avmorofova:~$ ls
backup      ex2.sh      example2.txt~  Lab9.md      Public
backup.sh~  ex2.sh~    '#example3.txt#'  lab_05      reports
backup.sh~  ex3.sh      example3.txt    Makefile     snap
Desktop     ex3.sh~    example3.txt~   Music        Templates
Documents   ex4.sh      '#example4.txt#'  my_os       usr
Downloads   ex4.sh~    example4.txt    example4.txt  Videos
ex1          example1.txt  example4.txt~   n1.doc       work
ex1.sh      example1.txt~  lab10.sh       n2.pdf
ex1.sh~     example2.txt  lab10.sh~      n3.txt
avmorofova@avmorofova:~$ ./ex4.sh ~ pdf sh txt doc
1 файлов содержится в каталоге /home/avmorofova с расширением pdf
6 файлов содержится в каталоге /home/avmorofova с расширением sh
5 файлов содержится в каталоге /home/avmorofova с расширением txt
1 файлов содержится в каталоге /home/avmorofova с расширением doc
```

Рис. 2.18: Проверка скрипта

## 6. 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 C-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 `.`, «*mva file{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 "NewJersey"`». 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 monday 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 (newline).
- 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 `*`,
- `-echoab'|'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 `type set` 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 `$` является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов `$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`.

## **3 Выводы**

В ходе выполнения лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux. Научилась писать небольшие командные файлы.