

Лабораторная работа №11

Отчет

Бондарь Алексей Олегович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	16
4	Выводы	22

List of Tables

List of Figures

2.1	Просмотр	6
2.2	man zip	6
2.3	man bzip2	7
2.4	man tar	7
2.5	Создание файла backup.sh	8
2.6	Создание файла	8
2.7	Проверка	9
2.8	Разархивирование	9
2.9	Создание файла	9
2.10	Пример командного файла	10
2.11	Проверка	11
2.12	Создание файла	11
2.13	Пример командного файла	12
2.14	Проверка	13
2.15	Создание файла	13
2.16	Пример командного файла	14
2.17	Проверка	15

1 Цель работы

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

2 Выполнение лабораторной работы

Для начала я изучил команды архивации, используя команды «man zip», «man bzip2», «man tar». (рис. 2.1) , (рис. 2.2) , (рис. 2.3) , (рис. 2.4)

```
aabondarj@dk3n52 ~ $ man zip
aabondarj@dk3n52 ~ $ man bzip2
aabondarj@dk3n52 ~ $ man tar
```

Figure 2.1: Просмотр

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]

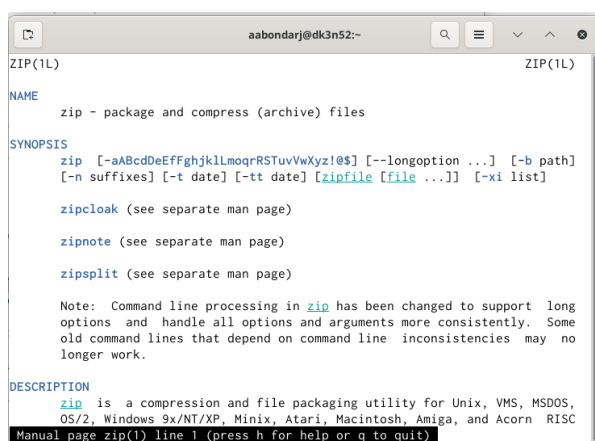
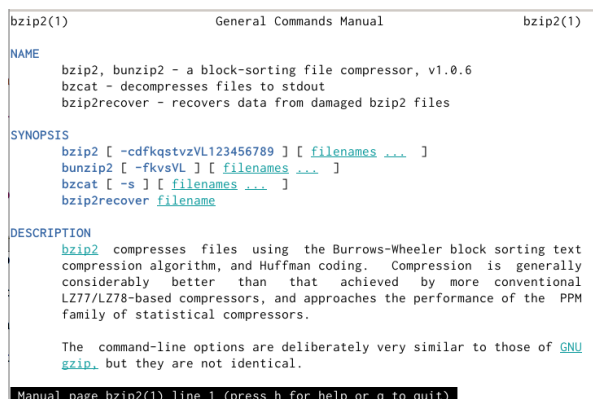


Figure 2.2: man zip

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

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



```
bzip2(1)                                General Commands Manual                                bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.6
  bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
  bzip2 [ -cdfkqstzVL123456789 ] [ filenames ... ]
  bunzip2 [ -fkvsVL ] [ filenames ... ]
  bzip2recover [ -s ] [ filenames ... ]
  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.

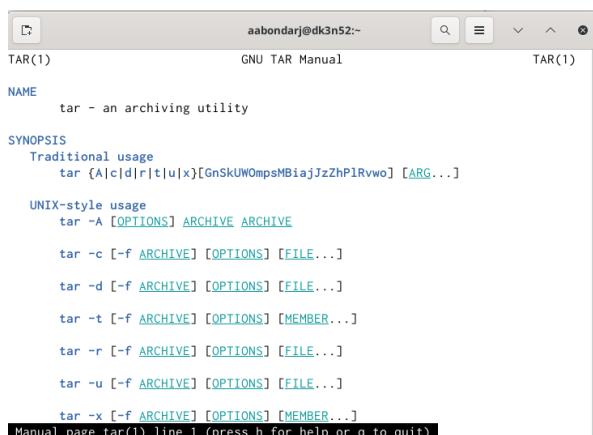
  The command-line options are deliberately very similar to those of GNU
  gzip, but they are not identical.

Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Figure 2.3: `man bzip2`

Синтаксис команды `tar` для архивации файла: `tar` [опции] [архив.tar] [файлы_для_архивации]

Синтаксис команды `tar` для разархивации/распаковки файла: `tar` [опции] [архив.tar]



```
TAR(1)                                GNU TAR Manual                                TAR(1)

NAME
  tar - an archiving utility

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

Manual page tar(1) line 1 (press h for help or q to quit)
```

Figure 2.4: `man tar`

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 &`»).(рис. 2.5)

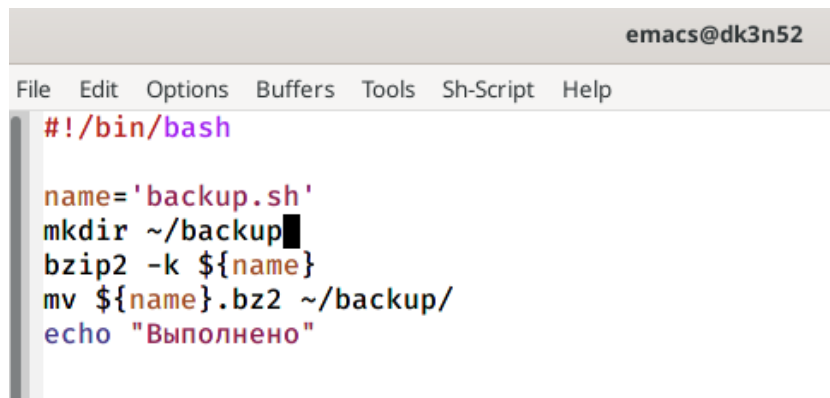
```

aabondarj@dk3n52 ~ $ touch backup.sh
aabondarj@dk3n52 ~ $ emacs &
[1] 3414
aabondarj@dk3n52 ~ $ bash: emacs: команда не найдена
emacs &

```

Figure 2.5: Создание файла backup.sh

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 (рис. 2.6). При написании скрипта использовал архиватор bzip2.



```

emacs@dk3n52
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"

```

Figure 2.6: Создание файла

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/»), посмотрела его содержимое (команда «ls») и просмотрел содержимое архива (команда «bunzip2 -c backup.sh.bz2») (рис. 2.7) и (рис. 2.8). Скрипт работает корректно.


```

aabondarj@dk3n52 ~ $ chmod +x *.
aabondarj@dk3n52 ~ $ ./backup.sh
Выполнено
aabondarj@dk3n52 ~ $ cd backup/
aabondarj@dk3n52 ~/backup $ ls
backup.sh.bz2

```

Figure 2.7: Проверка

```

aabondarj@dk3n52 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"

```

Figure 2.8: Разархивирование

Создал файл, в котором буду писать второй скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и «emacs &»).(рис. 2.9)

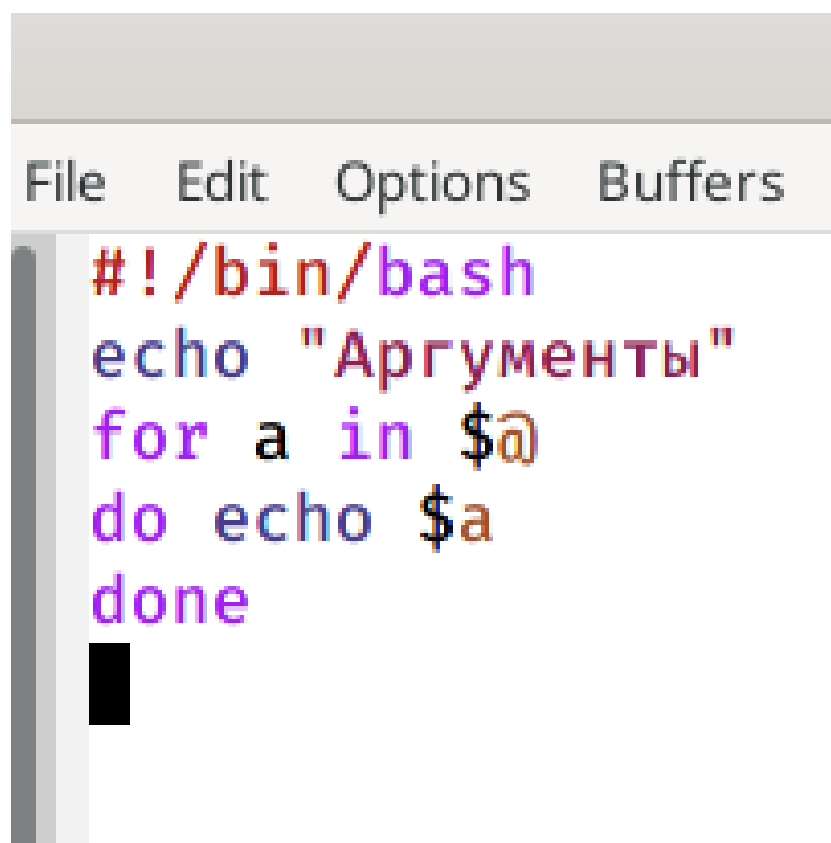
```

aabondarj@dk3n52 ~ $ touch prog2.sh
aabondarj@dk3n52 ~ $ emacs &

```

Figure 2.9: Создание файла

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. 2.10)



```
File Edit Options Buffers
#!/bin/bash
echo "Аргументы"
for a in $@
do echo $a
done
█
```

Figure 2.10: Пример командного файла

Проверил работу написанного скрипта (команды «./prog2.sh 0 1 2 3 4» и «./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Вводил аргументы, количество которых меньше 10 и больше 10.(рис. 2.11)

```

aabondarj@dk3n52 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
aabondarj@dk3n52 ~ $ ./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11
Аргументы
0
1
2
3
4
5
6
7
8
9
10
11

```

Figure 2.11: Проверка

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

```

aabondarj@dk3n52 ~ $ touch progl.s.sh
aabondarj@dk3n52 ~ $ emacs &

```

Figure 2.12: Создание файла

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.(рис. 2.13)

```
#!/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"
    fi
done
```

Figure 2.13: Пример командного файла

Далее проверил работу скрипта (команда «./progl.sh ~»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh») (рис. 2.14). Скрипт работает корректно.

```

aabondarj@dk3n52 ~ $ ./progl.s.sh ~
/afs/.dk.sci.pfu.edu.ru/home/a/a/aabondarj/2021-05-14 14-:
./progl.s.sh: строка 7: test: /afs/.dk.sci.pfu.edu.ru/home,
./progl.s.sh: строка 11: test: /afs/.dk.sci.pfu.edu.ru/home
./progl.s.sh: строка 15: test: /afs/.dk.sci.pfu.edu.ru/home
./progl.s.sh: строка 19: test: /afs/.dk.sci.pfu.edu.ru/home
./progl.s.sh: строка 22: test: /afs/.dk.sci.pfu.edu.ru/home
/afs/.dk.sci.pfu.edu.ru/home/a/a/aabondarj/asdfg
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/a/a/aabondarj/asdfg.asm
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/a/a/aabondarj/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено

```

Figure 2.14: Проверка

Для четвертого скрипта также создал файл (команда «touchformat.sh») и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &»).(рис. 2.15)

```

aabondarj@dk3n52 ~ $ touch format.sh
aabondarj@dk3n52 ~ $ emacs &
_ _

```

Figure 2.15: Создание файла

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 (рис. 2.16)

```
#!/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
```

Figure 2.16: Пример командного файла

Проверил 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. 2.17). Скрипт работает корректно.

```

aabondarj@dk3n52 ~ $ touch file.pdf file1.doc file2.doc
aabondarj@dk3n52 ~ $ ls
'2021-05-14 14-28-02.mkv'  lab10.sh
asdfg                    lab2
asdfg.asm                lab2.asm
backup                   lab.asm
backup.sh                labor
backup.sh~              newdir
conf.txt                 prog2.sh
DB.txt                   prog2.sh~
Ex                       progls.sh
example1.txt             progl.s.sh~
example2.txt             program.asm
example2.txt~            program.lst
example3.txt             public
example3.txt~            public_html
example4.txt             tmp
example4.txt~            tramvay
Ex.cpp                   tramvay.cpp
Ex.o                     tramvay.o
file1.doc                work
file2.doc                Алексей
file.pdf                 Алексей2
file.txt                 Алексей2.cpp
format.sh                Алексей.cpp
format.sh~              Видео
GNUstep                  Д3333.odt
HW.cpp                   Документы
lab                       Загрузки
lab02                    Изображения
lab03a                   Музыка
lab03b                   Общедоступные
lab05                    отчет_лаб_шаблон77.odt
lab05.asm                отчет_лаб_шаблон.odt
lab06                    Раб.txt
lab06.asm                'Рабочий стол'
lab07                    'Снимок экрана от 2020-09-04 15-38-41.png'
lab07.asm                'Снимок экрана от 2020-09-04 15-39-01.png'
'#lab10.sh#'            Шаблоны
aabondarj@dk3n52 ~ $ ./format.sh ~ pdf sh txt doc
1 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/a/aabondarj с расширением pdf
5 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/a/aabondarj с расширением sh
8 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/a/aabondarj с расширением txt
2 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/a/aabondarj с расширением doc
aabondarj@dk3n52 ~ $ man zip
aabondarj@dk3n52 ~ $ man bzip2
aabondarj@dk3n52 ~ $ man tar
aabondarj@dk3n52 ~ $ █

```

Figure 2.17: Проверка

3 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 на оболочкой Борна, использующая Сподобный 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/Linuxподобных 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 month day` 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 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 `Youhavemail` (у Вас есть почта).
- `TERM`: тип используемого терминала.
- `LOGNAME`: содержит регистрационное имя пользователя, которое устанавли-

ливается 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: `«bashкомандный_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+хмия_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 `unsetсфлагом -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 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`.

4 Выводы

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