

# **Отчет по лабораторной работе №10**

*дисциплина:* **Операционные системы**

Егорова Юлия Владимировна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
1.1	Выполнение лабораторной работы . . . . .	4
<b>2</b>	<b>Контрольные вопросы</b>	<b>12</b>
<b>3</b>	<b>Вывод</b>	<b>18</b>

# Список иллюстраций

1.1	Man zip. . . . .	4
1.2	Информация о man zip. . . . .	4
1.3	Man bzip2. . . . .	5
1.4	Информация о man bzip2. . . . .	5
1.5	Man tar. . . . .	5
1.6	Информация о man tar. . . . .	6
1.7	Создание файла. . . . .	6
1.8	Открытие emacs. . . . .	6
1.9	Поиск файла и его заполнение. . . . .	7
1.10	Проверка. . . . .	7
1.11	Проверка. . . . .	8
1.12	Проверка. . . . .	8
1.13	Создание файла и переход в emacs. . . . .	8
1.14	Поиск. . . . .	8
1.15	Заполнение. . . . .	9
1.16	Проверка. . . . .	9
1.17	Создание файла и переход в emacs. . . . .	9
1.18	Заполнение файла текстом. . . . .	10
1.19	Проверка. . . . .	10
1.20	Создание файла и переход в emacs. . . . .	10
1.21	Поиск. . . . .	11
1.22	Текст. . . . .	11
1.23	Команда и проверка. . . . .	11

# 1 Цель работы

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

## 1.1 Выполнение лабораторной работы

1. Вводим команду `man zip`, получаем информацию о ней:

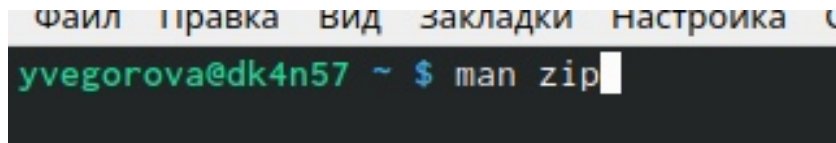


Рис. 1.1: Man zip.

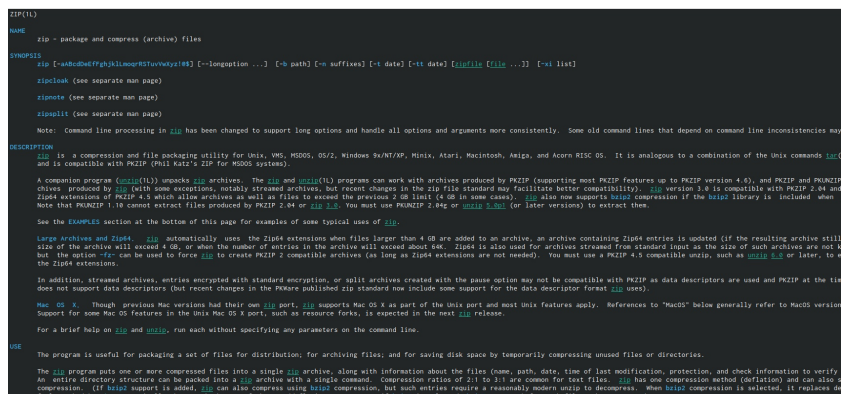


Рис. 1.2: Информация о man zip.

2. Команду `man bzip2` и также получаем информацию:

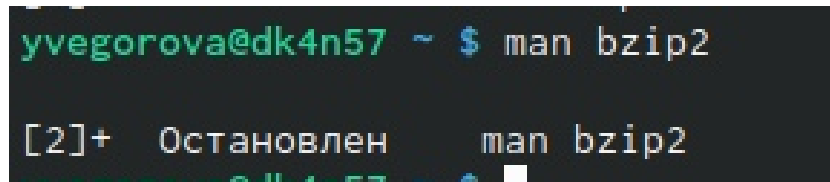


Рис. 1.3: Man bzip2.

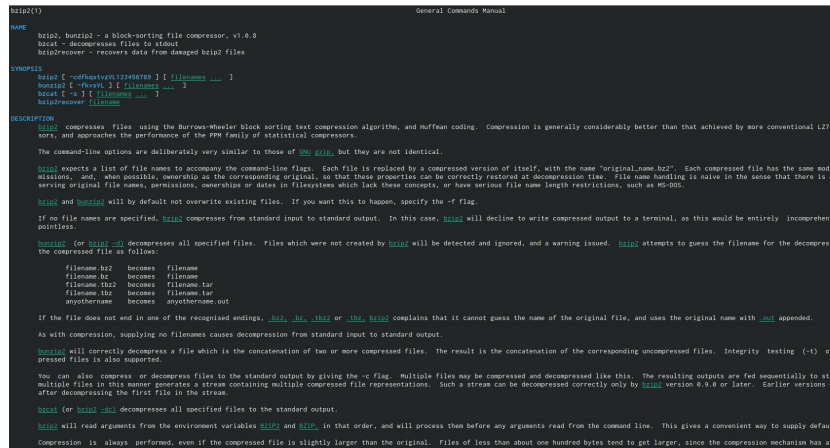


Рис. 1.4: Информация о man bzip2.

### 3. Команду man tar и получаем информацию:

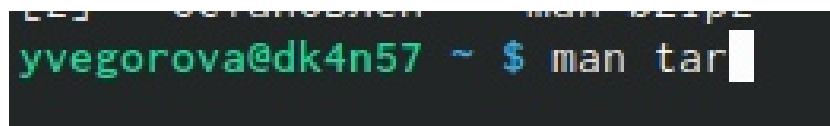


Рис. 1.5: Man tar.

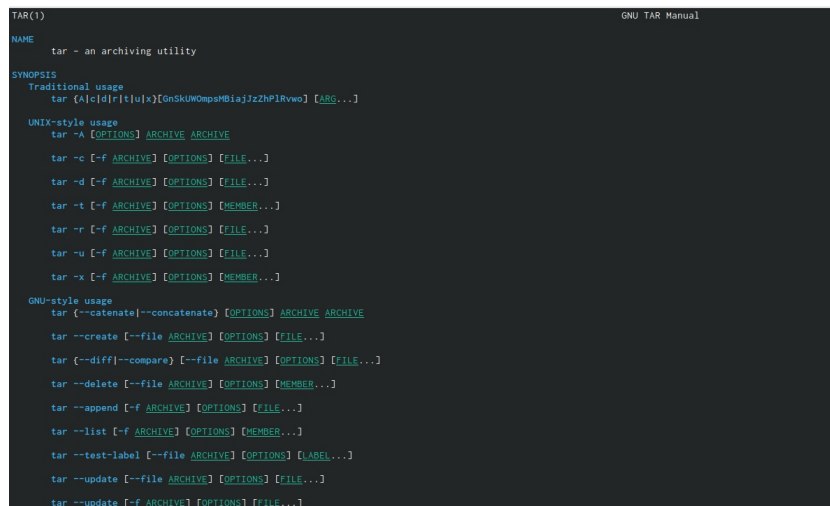


Рис. 1.6: Информация о man tar.

4. Создала файл backup.sh:



Рис. 1.7: Создание файла.

5. Открыла emacs:

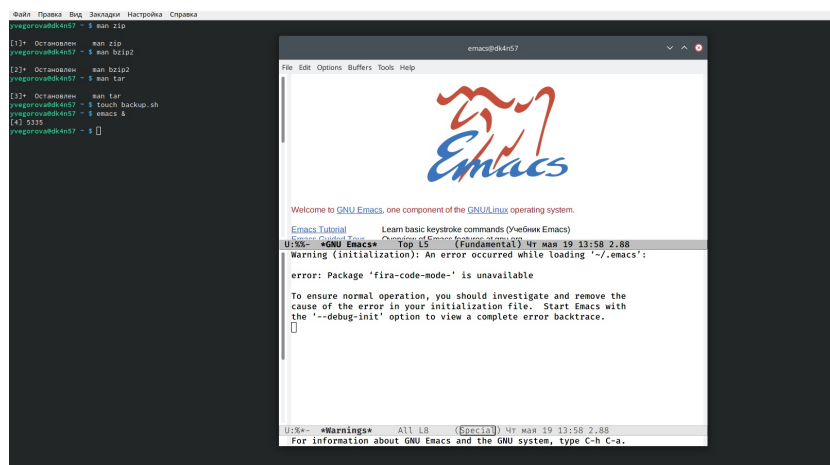
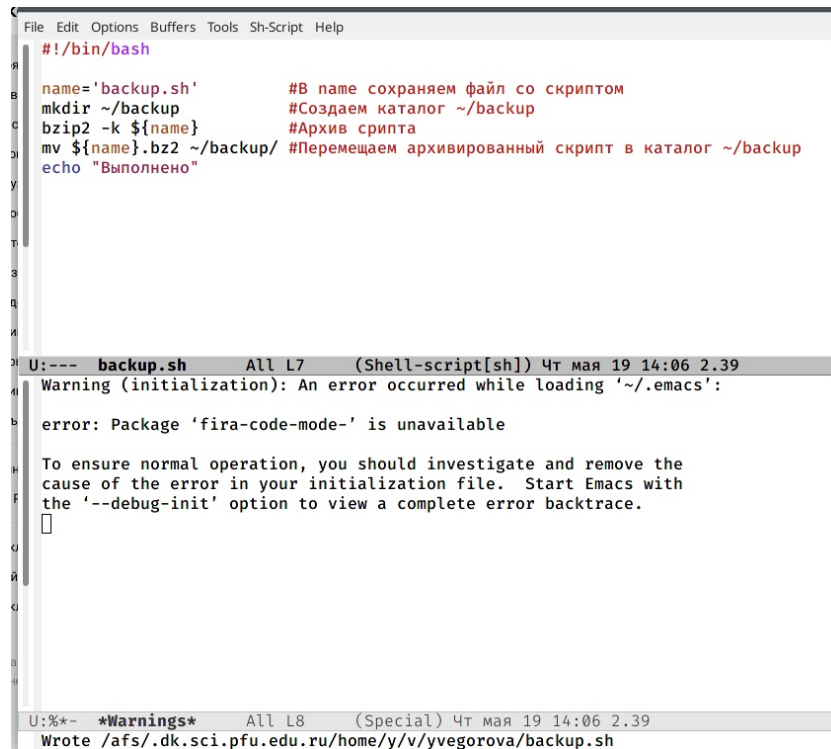


Рис. 1.8: Открытие emacs.

6. По клавишам C-x C-f нашла созданный файл и ввела следующий текст:



```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

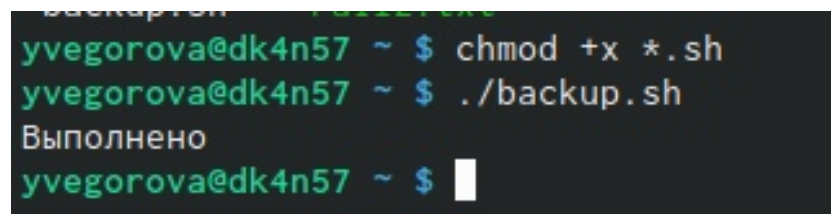
name='backup.sh'      #В name сохраняем файл со скриптом
mkdir ~/backup        #Создаем каталог ~/backup
bzip2 -k ${name}       #Архив скрипта
mv ${name}.bz2 ~/backup/ #Перемещаем архивированный скрипт в каталог ~/backup
echo "Выполнено"
```

U:--- backup.sh All L7 (Shell-script[sh]) Чт мая 19 14:06 2.39  
Warning (initialization): An error occurred while loading '~/.emacs':  
  
error: Package 'fira-code-mode-' is unavailable  
  
To ensure normal operation, you should investigate and remove the  
cause of the error in your initialization file. Start Emacs with  
the '--debug-init' option to view a complete error backtrace.  
□

U:%\*- \*Warnings\* All L8 (Special) Чт мая 19 14:06 2.39  
Wrote /afs/.dk.sci.pfu.edu.ru/home/y/v/yvegorova/backup.sh

Рис. 1.9: Поиск файла и его заполнение.

7. Через клавиши C-x C-s сохранила наш файл и проверила в консоли через следующие команды:



```
yvegorova@dk4n57 ~ $ chmod +x *.sh
yvegorova@dk4n57 ~ $ ./backup.sh
Выполнено
yvegorova@dk4n57 ~ $
```

Рис. 1.10: Проверка.

```
yvegorova@dk4n57 ~ $ cd backup/
yvegorova@dk4n57 ~/backup $ ls
backup.sh.bz2
yvegorova@dk4n57 ~/backup $
```

Рис. 1.11: Проверка.

```
yvegorova@dk4n57 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'      #В папе сохраняем файл со скриптом
mkdir ~/backup        #Создаем каталог ~/backup
bzip2 -k ${name}       #Архив скрипта
mv ${name}.bz2 ~/backup/ #Перемещаем архивированный скрипт в каталог ~/backup
echo "Выполнено"
yvegorova@dk4n57 ~/backup $
```

Рис. 1.12: Проверка.

8. Создала файл prog2.sh и перешла в emacs:

```
yvegorova@dk4n57 ~/backup $ cd
yvegorova@dk4n57 ~ $ touch prog2.sh
yvegorova@dk4n57 ~ $ emacs &
```

Рис. 1.13: Создание файла и переход в emacs.

9. По клавишам C-x C-f нашла созданный файл и ввела следующий текст:

```
U:%*- *Warnings* All L8 (
Find file: ~/prog2.sh
```

Рис. 1.14: Поиск.



```
#!/bin/bash
echo "Аргументы"
for a in $@ #Цикл для прохода по введенным аргументам
do echo $a #Вывод аргумента
done
```

Рис. 1.15: Заполнение.

10. Через клавиши C-x C-s сохранила наш файл и проверила в консоли через команду:

```
yvegorova@dk4n57 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
yvegorova@dk4n57 ~ $
```

Рис. 1.16: Проверка.

11. Создала файл progl.sh и перешла в emacs:

```
0 1 2 3 4
yvegorova@dk4n57 ~ $ touch progl.sh
[5] Завершён emacs
yvegorova@dk4n57 ~ $ emacs &
[4] 8708
```

Рис. 1.17: Создание файла и переход в emacs.

12. Ввела текст:

```
#!/bin/bash
a="$1"          #В переменную а сохраняем путь до заданного каталога
for i in ${a}/* #Цикл, который проходит по всем каталогам и файлам
do
    echo "$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
done
```

Рис. 1.18: Заполнение файла текстом.

13. Осуществила проверку:

```
backup.sh course-directory-student-template
yvegorova@dk4n57 ~ $ ./progl1.sh ~
/afs/.dk.sci.pfu.edu.ru/home/y/v/yvegorova/-
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/y/v/yvegorova/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнено разрешено
/afs/.dk.sci.pfu.edu.ru/home/y/v/yvegorova/backup.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнено разрешено
/afs/.dk.sci.pfu.edu.ru/home/y/v/yvegorova/backup.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/y/v/yvegorova/bin
Каталог
Чтение разрешено
Запись разрешена
Выполнено разрешено
/afs/.dk.sci.pfu.edu.ru/home/y/v/yvegorova/course-directory-student-template
Каталог
Чтение разрешено
```

Рис. 1.19: Проверка.

14. Снова создала файл под названием format.sh и перешла в emacs:

```
Выполнено разрешено
yvegorova@dk4n57 ~ $ touch format.sh
[4] Завершён emacs
yvegorova@dk4n57 ~ $ emacs &
```

Рис. 1.20: Создание файла и переход в emacs.

15. Нашла созданный файл и ввела текст:

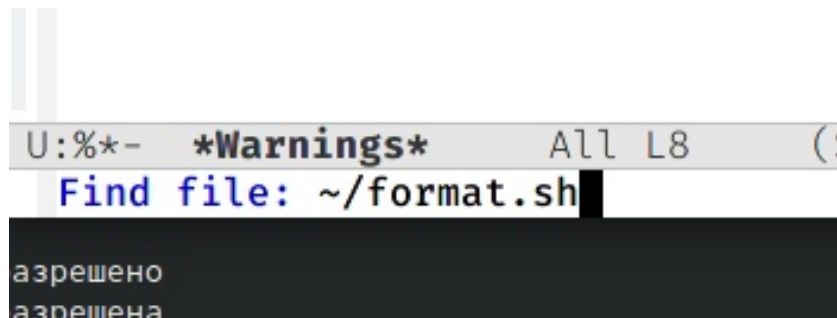


Рис. 1.21: Поиск.

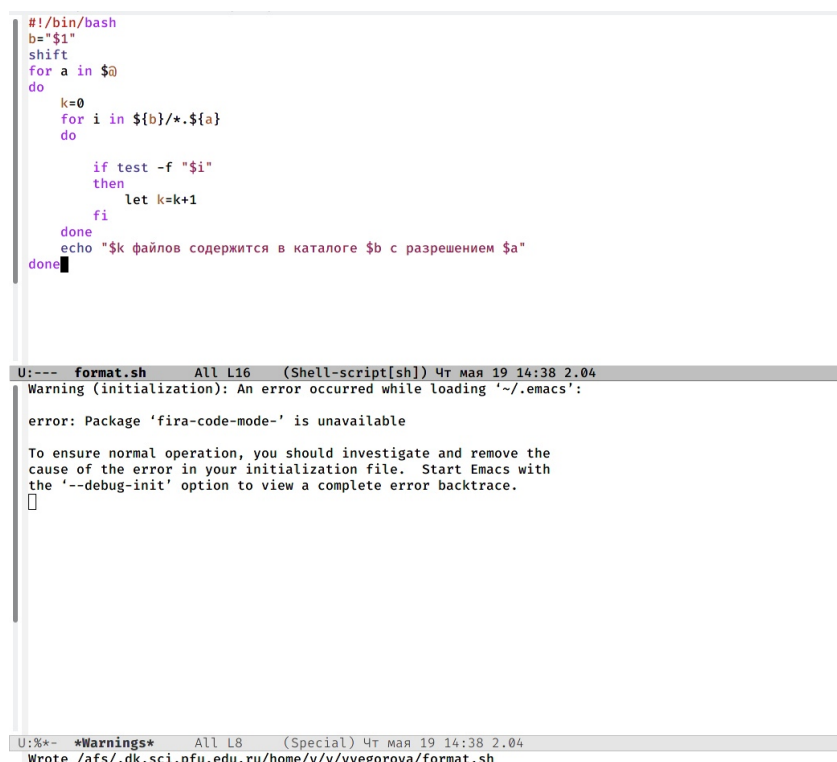


Рис. 1.22: Текст.

16. Ввела следующую команду и выполнила проверку:

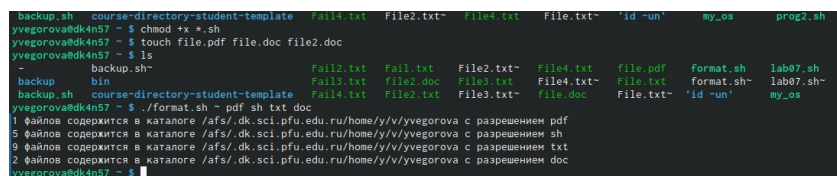


Рис. 1.23: Команда и проверка.

## 2 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, с

C-оболочка (или csh) – надстройка на оболочкой Борна, использующая Сподобный синтаксис

Оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления программой

BASH – сокращение от BourneAgainShell (опять оболочка Борна), в основе своей совместимая

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 "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 единственный 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

PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. Если программа, запущенная командным процессором, требует ввода, то эти переменные определяют, что будет введено.

HOME: имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то программа переходит в этот каталог.

IFS: последовательность символов, являющихся разделителями в командной строке, например пробелы.

MAIL: командный процессор каждый раз перед выводом на экран промптера проверяет, есть ли почта для пользователя.

TERM: тип используемого терминала.

LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при запуске оболочки.

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

9). Снятие специального смысла с метасимвола называется экранированием мета символа. Экранирование может быть осуществлено с помощью предшествующего мета символу символа, который, в свою очередь, является мета символом. 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'|` 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 осуществить 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 `unsetc`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 `$` является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов `$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 пр

`$!` –номер процесса, в рамках которого выполняется последняя вызванная на выполне

`--`–значение флагов командного процессора;

`${#}` –возвращает целое число –количествослов, которые были результатом `$`;

`${#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 заменен

`${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 выв



`${name+value}` -это выражение работает противоположно `${name-value}`. Если переменная

`${name#pattern}` -представляет значение переменной `name` с удалённым самым коротким

`${#name[*]}` и `${#name[@]}`-эти выражения возвращают количество элементов в массиве

## **3 Вывод**

В ходе данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux, научилась писать небольшие командные файлы.