

# **Лабораторная работа №11**

**Дисциплина: Операционные системы**

Коновалова Татьяна Борисовна

# Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Библиография	26
5	Выводы	27

## List of Tables

# List of Figures

3.1	Работа с консолью . . . . .	7
3.2	Информация о zip . . . . .	8
3.3	Информация о bzip2 . . . . .	9
3.4	Информация о tar . . . . .	10
3.5	Создание файла . . . . .	10
3.6	Скрипт №1 . . . . .	11
3.7	Проверка работы скрипта . . . . .	12
3.8	Проверка работы скрипта . . . . .	12
3.9	Создание файла . . . . .	13
3.10	Открываем etags . . . . .	13
3.11	Скрипт №2 . . . . .	14
3.12	Проверка работы скрипта . . . . .	15
3.13	Проверка работы скрипта . . . . .	15
3.14	Создание файла . . . . .	16
3.15	Скрипт №3 . . . . .	16
3.16	Скрипт №3 . . . . .	17
3.17	Проверка работы скрипта . . . . .	17
3.18	Проверка работы скрипта . . . . .	18
3.19	Проверка работы скрипта . . . . .	18
3.20	Создание файла . . . . .	19
3.21	Скрипт №4 . . . . .	19
3.22	Проверка работы скрипта . . . . .	20

# 1 Цель работы

Цель данной лабораторной работы — Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

## 2 Задание

1. Сделать отчёт по лабораторной работе №11 в формате Markdown.
2. Изучить основы программирования в оболочке ОС UNIX/Linux.

### 3 Выполнение лабораторной работы

1). Для начала я изучила команды архивации, используя команды «manzip», «manbzip2», «mantar» (алгоритм действий представлен на рис. 3.1 , 3.2 , 3.3 , 3.4 ).

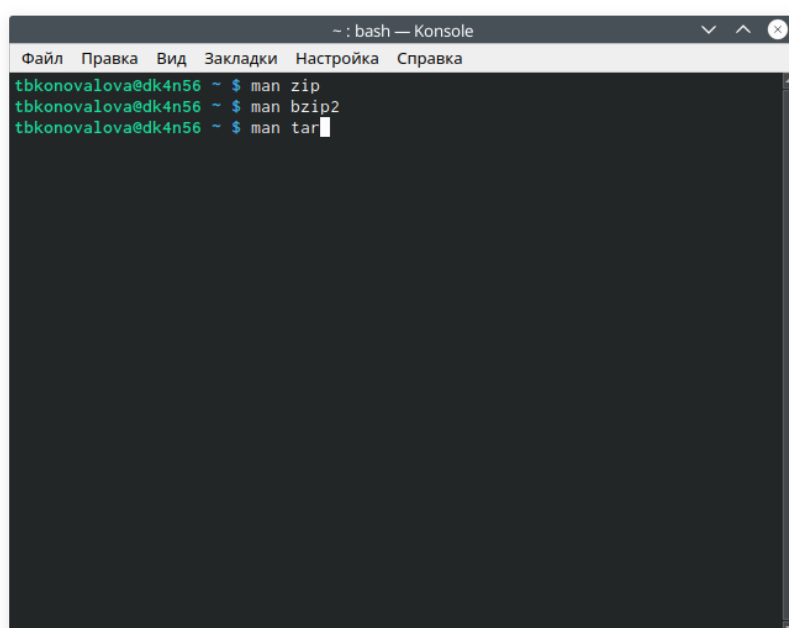
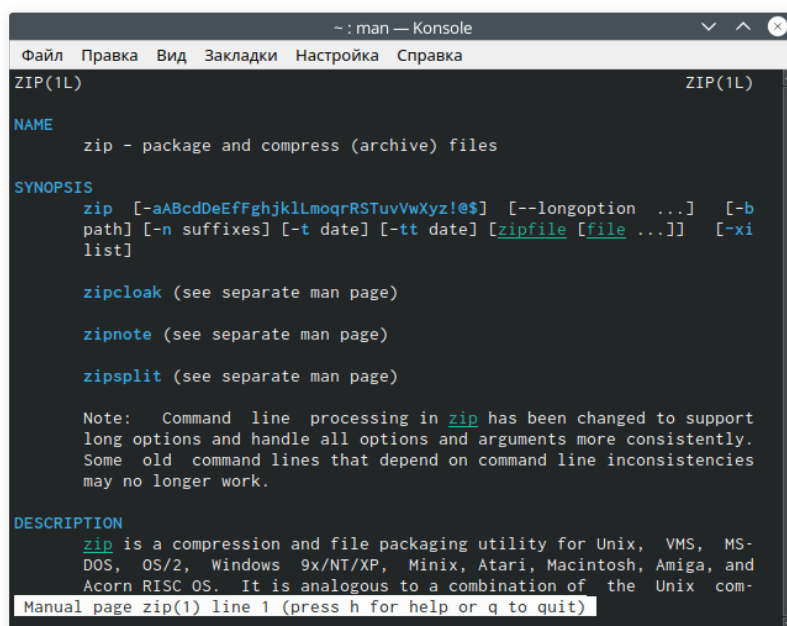


Figure 3.1: Работа с консолью

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]



```
~: man — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
ZIP(1L)                                     ZIP(1L)
NAME
    zip - package and compress (archive) files

SYNOPSIS
    zip [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$] [--longoption ...] [-b
    path] [-n suffixes] [-t date] [-tt date] [zipfile [file ...]] [-xi
    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, MS-
    DOS, OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and
    Acorn RISC OS. It is analogous to a combination of the Unix com-
Manual page zip(1) line 1 (press h for help or q to quit)
```

Figure 3.2: Информация о zip

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

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



```
~: man — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
bzip2(1)      General Commands Manual      bzip2(1)

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

SYNOPSIS
  bzip2 [ -cdfkqstvl123456789 ] [ filenames ... ]
  bunzip2 [ -fkvsVL ] [ filenames ... ]
  bzcata [ -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.

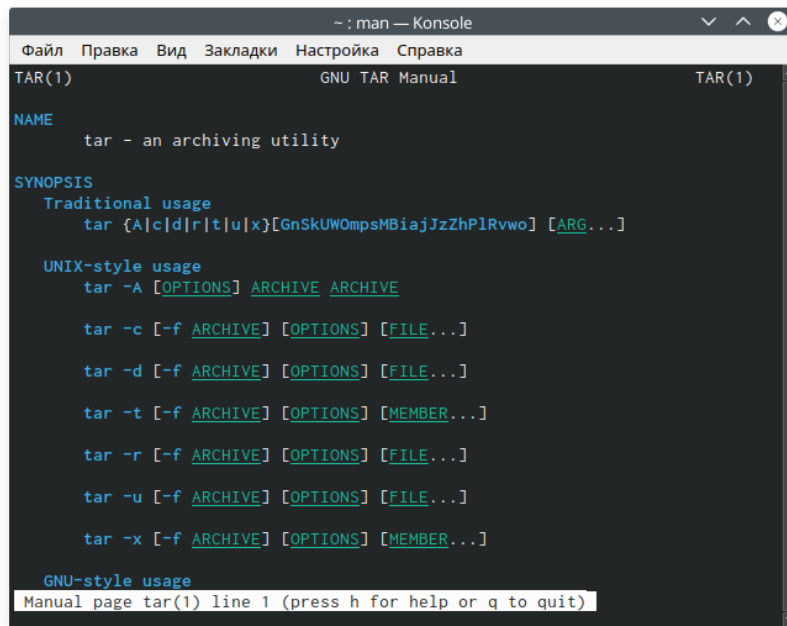
  The command-line options are deliberately very similar to those of
  GNU gzip, but they are not identical.

  bzip2 expects a list of file names to accompany the command-line
  flags. Each file is replaced by a compressed version of itself,
  Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Figure 3.3: Информация о 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]



```
~ : man — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
TAR(1)                                GNU TAR Manual                                TAR(1)

NAME
    tar - an archiving utility

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

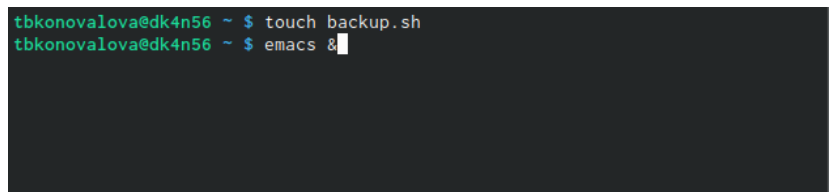
    UNIX-style usage
    tar -A [OPTIONS] ARCHIVE ARCHIVE

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

Figure 3.4: Информация о tar

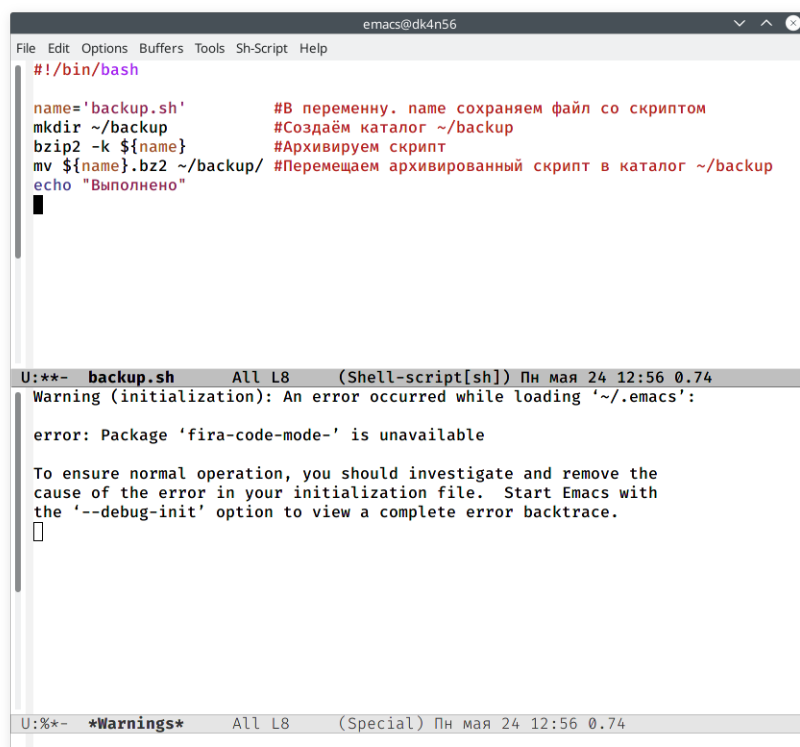
Создала файл, в котором буду писать первый скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (Скриншот 3.5 ).



```
tbkonovalova@dk4n56 ~ $ touch backup.sh
tbkonovalova@dk4n56 ~ $ emacs &
```

Figure 3.5: Создание файла

Написала скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию back up в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar (Скриншот 3.6 ). При написании скрипта использовала архиватор bzip2.



```
emacs@dk4n56
File Edit Options Buffers Tools Sh-Script Help

#!/bin/bash

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

```
U:*** backup.sh All L8 (Shell-script[sh]) Пн мая 24 12:56 0.74
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) Пн мая 24 12:56 0.74
```

Figure 3.6: Скрипт №1

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 -cbackup.sh.bz2») (алгоритм действий представлен на рис. 3.7 , 3.8 ).Скрипт работает корректно.

```
backup : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
tbkonovalova@dk4n56 ~ $ touch backup.sh
tbkonovalova@dk4n56 ~ $ emacs &
[1] 5437
tbkonovalova@dk4n56 ~ $ ls
-                example2.txt~  lab05.o        laboratory      tmp
abc              example2.txt~  lab06          may             work
abc1            example3.txt~  lab06.asm      monthly         Видео
asdfg          example3.txt~  lab06.o        my_os           Документы
asdfg.asm       example4.txt~  lab07          play            Загрузки
asdfg.o         example4.txt~  lab07.asm      program.asm     Изображения
australia       file.txt       lab07.o        program.lst     Музыка
backup.sh       file.txt       lab07.sh       public          Общедоступные
backup.sh~      games         lab07.sh~      public_html     'Рабочий стол'
conf.txt        GNUstep       lab2           reports         Шаблоны
example1.txt~   lab05         lab2.asm       ski.plases
example1.txt~   lab05.asm     lab2.o        text.txt
tbkonovalova@dk4n56 ~ $ chmod +X *.sh
tbkonovalova@dk4n56 ~ $ ./backup.sh
bash: ./backup.sh: Отказано в доступе
tbkonovalova@dk4n56 ~ $ chmod +x *.sh
tbkonovalova@dk4n56 ~ $ ./backup.sh
Выполнено
tbkonovalova@dk4n56 ~ $ cd backup/
tbkonovalova@dk4n56 ~/backup $ ls
backup.sh.bz2
tbkonovalova@dk4n56 ~/backup $
```

Figure 3.7: Проверка работы скрипта

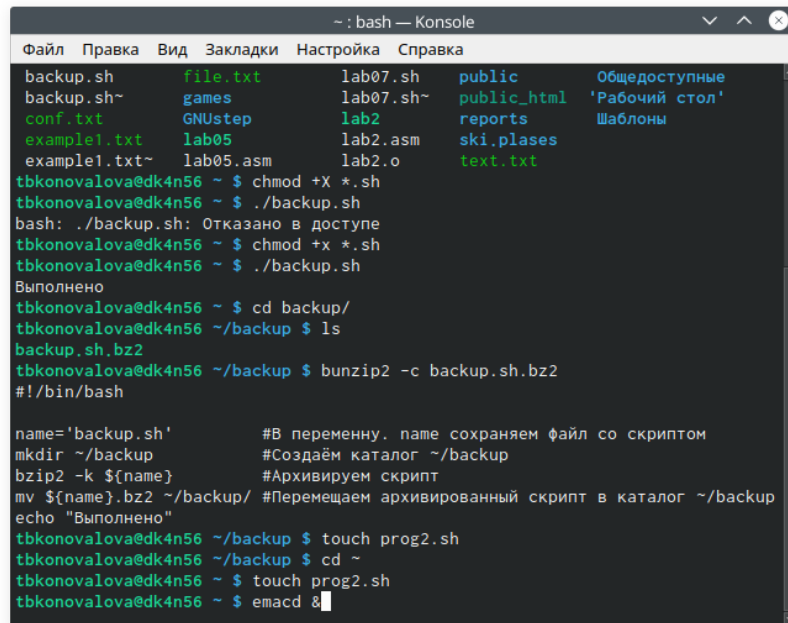
```
backup : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
asdfg.asm       example4.txt~  lab07          play            Загрузки
asdfg.o         example4.txt~  lab07.asm      program.asm     Изображения
australia       file.txt       lab07.o        program.lst     Музыка
backup.sh       file.txt       lab07.sh       public          Общедоступные
backup.sh~      games         lab07.sh~      public_html     'Рабочий стол'
conf.txt        GNUstep       lab2           reports         Шаблоны
example1.txt~   lab05         lab2.asm       ski.plases
example1.txt~   lab05.asm     lab2.o        text.txt
tbkonovalova@dk4n56 ~ $ chmod +X *.sh
tbkonovalova@dk4n56 ~ $ ./backup.sh
bash: ./backup.sh: Отказано в доступе
tbkonovalova@dk4n56 ~ $ chmod +x *.sh
tbkonovalova@dk4n56 ~ $ ./backup.sh
Выполнено
tbkonovalova@dk4n56 ~ $ cd backup/
tbkonovalova@dk4n56 ~/backup $ ls
backup.sh.bz2
tbkonovalova@dk4n56 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'      #В переменную name сохраняем файл со скриптом
mkdir ~/backup        #Создаём каталог ~/backup
bzip2 -k ${name}       #Архивируем скрипт
mv ${name}.bz2 ~/backup/ #Перемещаем архивированный скрипт в каталог ~/backup
echo "Выполнено"
tbkonovalova@dk4n56 ~/backup $
```

Figure 3.8: Проверка работы скрипта

2). Создала файл, в котором буду писать второйскрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и

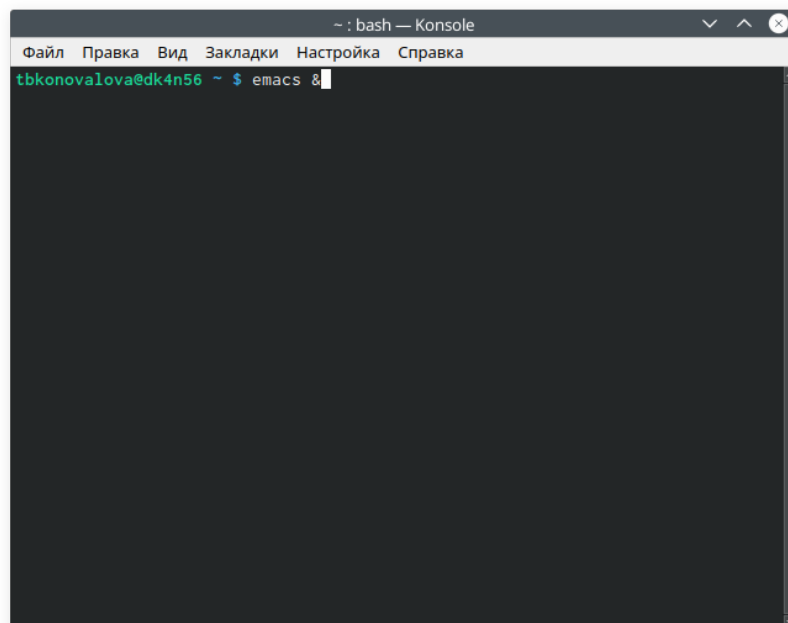
«emacs &») (Скриншоты 3.9, 3.10 ).



```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
backup.sh  file.txt  lab07.sh  public  Общедоступные
backup.sh~  games  lab07.sh~  public_html  'Рабочий стол'
conf.txt  GNUstep  lab2  reports  Шаблоны
example1.txt  lab05  lab2.asm  ski_places
example1.txt~  lab05.asm  lab2.o  text.txt
tbkonovalova@dk4n56 ~ $ chmod +X *.sh
tbkonovalova@dk4n56 ~ $ ./backup.sh
bash: ./backup.sh: Отказано в доступе
tbkonovalova@dk4n56 ~ $ chmod +x *.sh
tbkonovalova@dk4n56 ~ $ ./backup.sh
Выполнено
tbkonovalova@dk4n56 ~ $ cd backup/
tbkonovalova@dk4n56 ~/backup $ ls
backup.sh.bz2
tbkonovalova@dk4n56 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'      #В переменную name сохраняем файл со скриптом
mkdir ~/backup        #Создаём каталог ~/backup
bzip2 -k ${name}       #Архивируем скрипт
mv ${name}.bz2 ~/backup/ #Перемещаем архивированный скрипт в каталог ~/backup
echo "Выполнено"
tbkonovalova@dk4n56 ~/backup $ touch prog2.sh
tbkonovalova@dk4n56 ~/backup $ cd ~
tbkonovalova@dk4n56 ~ $ touch prog2.sh
tbkonovalova@dk4n56 ~ $ emacs &
```

Figure 3.9: Создание файла

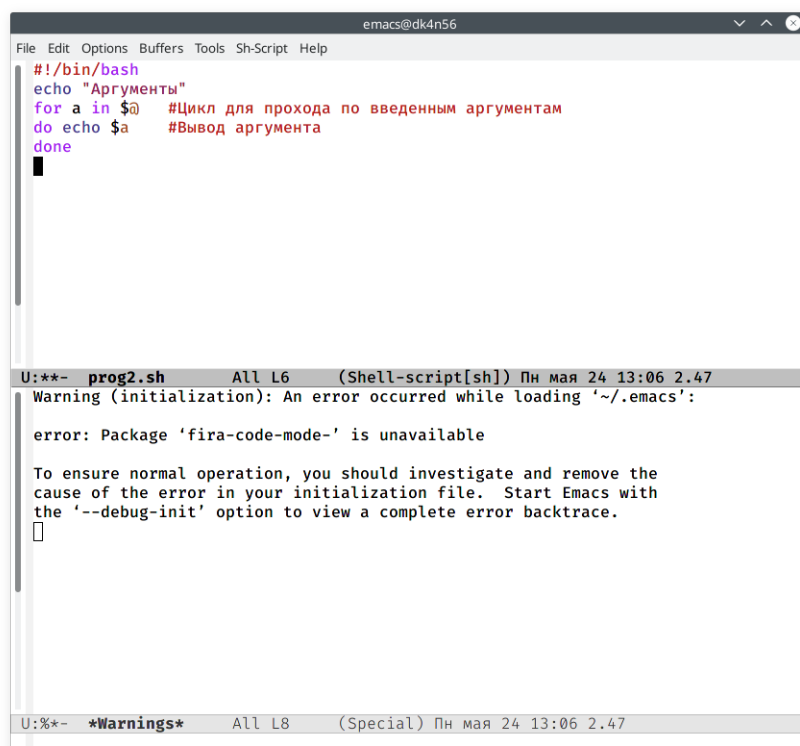


```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
tbkonovalova@dk4n56 ~ $ emacs &
```

Figure 3.10: Открываем emacs

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 (Рисунок 3.11 ).



The screenshot shows an Emacs window titled 'emacs@dk4n56'. The main buffer contains a shell script named 'prog2.sh' with the following content:

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

Below the script, the output of the script is shown in a separate buffer. The output is:

```
U:*** prog2.sh All L6 (Shell-script[sh]) Пн мая 24 13:06 2.47
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.
```

The bottom status bar of the Emacs window shows 'U:%\*- \*Warnings\* All L8 (Special) Пн мая 24 13:06 2.47'.

Figure 3.11: Скрипт №2

Проверила работу написанного скрипта (команды «./prog2.sh0 1 2 3 4» и «./prog2.sh0 1 2 3 45 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 (алгоритм действий представлен на рис. 3.12 , 3.13 ). Скрипт работает корректно.

```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
tbkonovalova@dk4n56 ~ $ chmod +x *.sh
tbkonovalova@dk4n56 ~ $ ls
-          example2.txt~  lab06      monthly   work
abc        example2.txt~  lab06.asm  my_os     Видео
abc1       example3.txt~  lab06.o    play      Документы
asdfg      example3.txt~  lab07      prog2.sh  Загрузки
asdfg.asm  example4.txt~  lab07.asm  prog2.sh~ Изображения
asdfg.o    example4.txt~  lab07.o    program.asm Музыка
australia  feathers       lab07.sh   program.lst Общедоступные
backup     file.txt       lab07.sh~  public    'Рабочий стол'
backup.sh  games         lab2       public_html Шаблоны
backup.sh~ GNUstep       lab2.asm   reports
conf.txt   lab05         lab2.o     ski.places
example1.txt lab05.asm    laboratory text.txt
example1.txt~ lab05.o      may        tmp
tbkonovalova@dk4n56 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
tbkonovalova@dk4n56 ~ $ ./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11
Аргументы
0
1
```

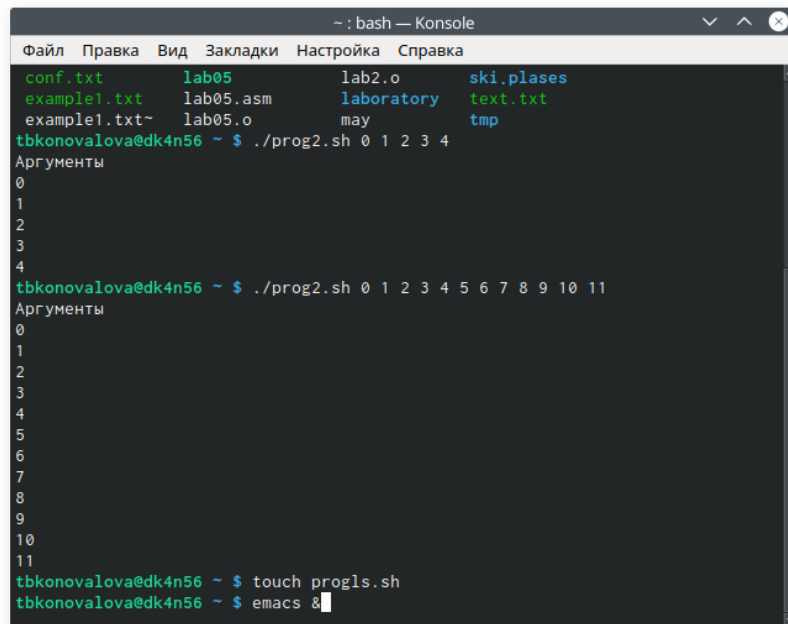
Figure 3.12: Проверка работы скрипта

```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
backup.sh~  GNUstep    lab2.asm   reports
conf.txt    lab05      lab2.o     ski.places
example1.txt lab05.asm  laboratory text.txt
example1.txt~ lab05.o    may        tmp
tbkonovalova@dk4n56 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
tbkonovalova@dk4n56 ~ $ ./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
tbkonovalova@dk4n56 ~ $
```

Figure 3.13: Проверка работы скрипта

3). Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touchprogl.sh» и

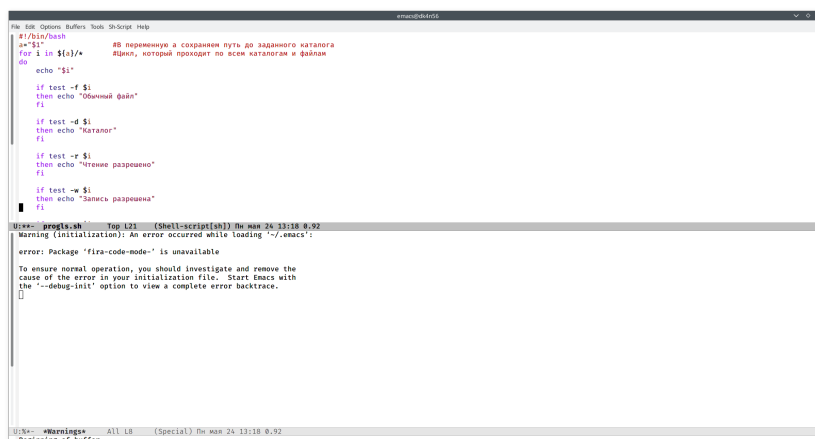
«emacs&») (алгоритм действий представлен на рис. 3.14 ).



```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
conf.txt      lab05          lab2.o        ski.places
example1.txt  lab05.asm     laboratory    text.txt
example1.txt~ lab05.o        may          tmp
tbkonovalova@dk4n56 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
tbkonovalova@dk4n56 ~ $ ./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
tbkonovalova@dk4n56 ~ $ touch prog1s.sh
tbkonovalova@dk4n56 ~ $ emacs &
```

Figure 3.14: Создание файла

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 (Скриншоты 3.15 , 3.16 ).



```
File Edit Options Buffers Tools Shell Scripts Help
emacs@dk4n56:~$
#!/bin/bash
# Переменная с сохраненным путем до заданного каталога
# $1, который проходит по всем каталогам и файлам
for i in $(ls)
do
    echo "$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

U***: prog2s.sh Top L21 (Shell-script[sh]) Mon Mar 24 13:18 8.92
Warning (initialization): An error occurred while loading ~/.emacs*:
error: Package "first-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) Mon Mar 24 13:18 8.92
Beginning of buffer
```

Figure 3.15: Скрипт №3





Figure 3.16: Скрипт №3

Далее проверила работу скрипта (команда «./progl.sh~»), предварительно добавив для него право на выполнение (команда «chmod+x\*.sh») (алгоритм действий представлен на рис. 3.17 , 3.18 , 3.19 ). Скрипт работает корректно.

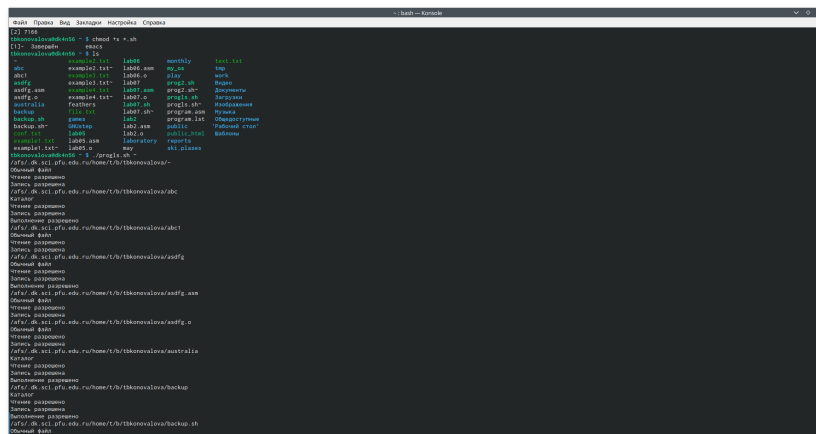


Figure 3.17: Проверка работы скрипта

```
#!/bin/bash
set -e
set -x

# Create a directory for the test
mkdir -p /tmp/test_dir

# Create a file in the directory
touch /tmp/test_dir/test_file.txt

# Check if the file exists
if [ -f /tmp/test_dir/test_file.txt ]; then
    echo "File exists"
else
    echo "File does not exist"
fi
```

Figure 3.18: Проверка работы скрипта

```
#!/bin/bash
set -e
set -x

# Create a directory for the test
mkdir -p /tmp/test_dir

# Create a file in the directory
touch /tmp/test_dir/test_file.txt

# Check if the file exists
if [ -f /tmp/test_dir/test_file.txt ]; then
    echo "File exists"
else
    echo "File does not exist"
fi
```

Figure 3.19: Проверка работы скрипта

4). Для четвертого скрипта создала файл (команда «touch format.sh») и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (Скриншот 3.20).

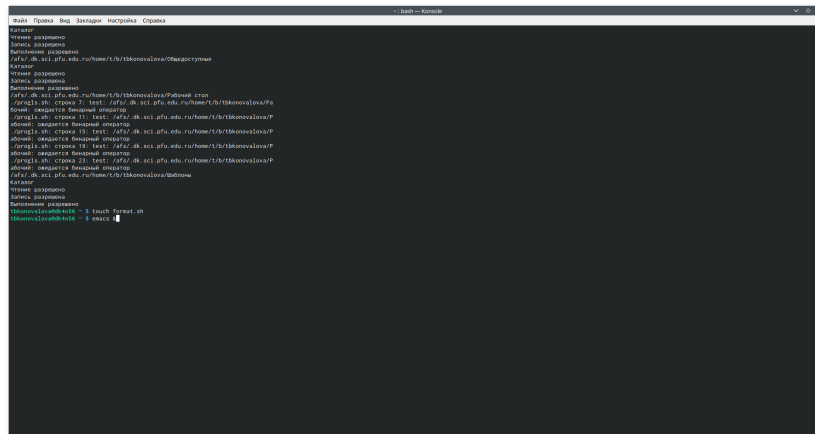


Figure 3.20: Создание файла

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 (Скриншот 3.21 ).

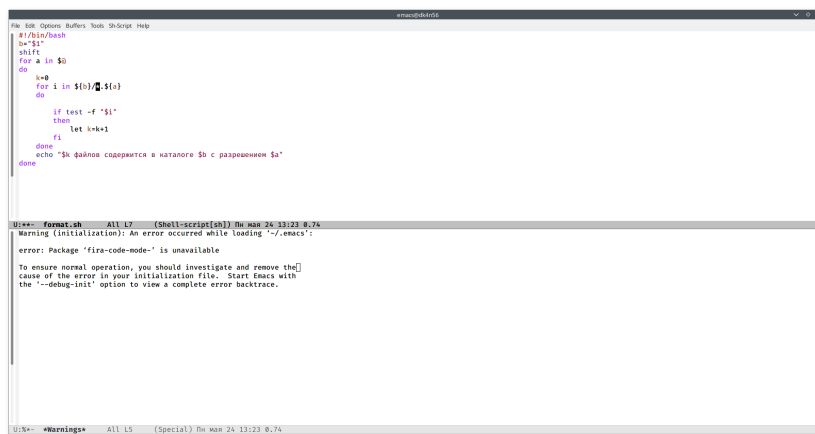


Figure 3.21: Скрипт №4

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») (Рисунок 3.22 ).Скрипт работает корректно.

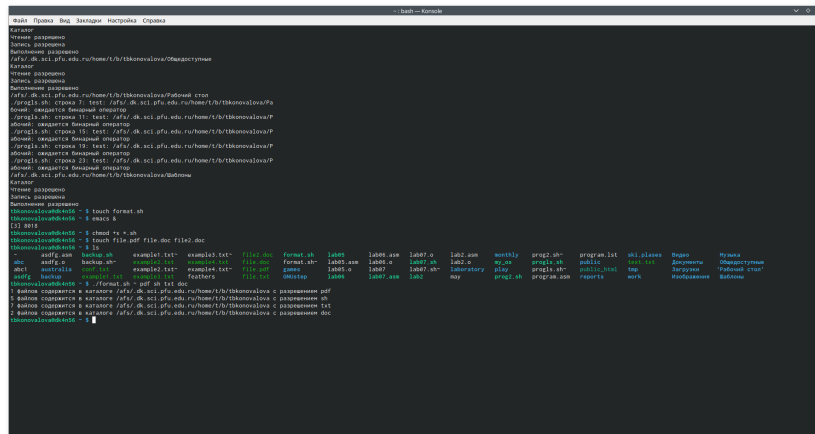


Figure 3.22: Проверка работы скрипта

## Ответы на 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: 1. оболочка Борна (Bourne shell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; 2. C-оболочка (или csh) – надстройка на оболочке Борна, использующая подобный синтаксис команд с возможностью сохранения истории выполнения команд; 3. Оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; 4. 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 "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 отобрать 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: 1. 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.

2. 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 >.
3. HOME: имя домашнего каталога пользователя. Если команда cd вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
4. IFS: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (newline).
5. 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).
6. TERM: тип используемого терминала.
7. 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 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'|'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 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 `unset` 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 окру-

жения. 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: 1. \$\* –отображается вся командная строка или параметры оболочки; 2. \$? –код завершения последней выполненной команды; 3. \$\$ –уникальный идентификатор процесса, в рамках которого выполняется командный процессор; 4. \$! –номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; 5. \$--значение флагов командного процессора; 6. \$# –возвращает целое число –количество слов, которые были результатом \$; 7. \$#name –возвращает целое значение длины строки в переменной name; 8. \${name[n]} –обращение к n-му элементу массива; 9. \${name[\*]} –перечисляет все элементы массива, разделённые пробелом; 10. \${name[@]} –то же самое, но позволяет учитывать символы пробелы в самих переменных; 11. \${name:-value} –если значение переменной name не определено,



то оно будет заменено на указанное value; 12. `${name:value}` –проверяется факт существования переменной; 13. `${name=value}` –если name не определено, то ему присваивается значение value; 14. `${name?value}` –останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке; 15. `${name+value}` –это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется value; 16. `${name#pattern}` –представляет значение переменной name с удалённым самым коротким левым образцом (pattern); 17. `${#name[*]}` и `${#name[@]}`–эти выражения возвращают количество элементов в массиве name.

## 4 Библиография

1. Программное обеспечение GNU/Linux. Лекция 3. FHS и процессы (Г. Курячий, МГУ);
2. Программное обеспечение GNU/Linux. Лекция 4. Права доступа (Е. Алёхова, МГУ);
3. Электронный ресурс: [https://ru.wikibooks.org/wiki %D0%92%D0%B2%D0%B5%D0%B4%D0%](https://ru.wikibooks.org/wiki/%D0%92%D0%B2%D0%B5%D0%B4%D0%)
4. Электронный ресурс: <http://fedoseev.net/materials/courses/admin/ch02.html>

## **5 Выводы**

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