

# **Отчет по лабораторной работе №11**

**Дисциплина: Операционные системы**

Калистратова Ксения Евгеньевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задачи</b>	<b>5</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>17</b>
<b>5</b>	<b>Выводы</b>	<b>23</b>
<b>6</b>	<b>Библиография</b>	<b>24</b>

# List of Figures

3.1	Команда <code>man</code> . . . . .	6
3.2	Синтаксис команды <code>zip</code> . . . . .	7
3.3	Синтаксис команды <code>bzip2</code> . . . . .	8
3.4	Синтаксис команды <code>tar</code> . . . . .	9
3.5	Создание файла и открытие <code>emacs</code> . . . . .	9
3.6	Первый скрипт . . . . .	10
3.7	Проверка работы скрипта . . . . .	10
3.8	Проверка работы скрипта . . . . .	11
3.9	Создание файла и открытие <code>emacs</code> . . . . .	11
3.10	Второй скрипт . . . . .	12
3.11	Проверка работы скрипта . . . . .	13
3.12	Создание файла и открытие <code>emacs</code> . . . . .	13
3.13	Третий скрипт . . . . .	14
3.14	Проверка работы скрипта . . . . .	14
3.15	Создание файла и открытие <code>emacs</code> . . . . .	15
3.16	Четвертый скрипт . . . . .	15
3.17	Проверка работы скрипта . . . . .	16

# 1 Цель работы

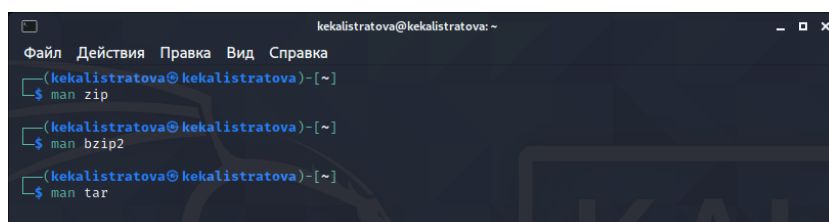
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

## 2 Задачи

1. Познакомиться с командными процессорами.
2. Изучить переменные, арифметические операторы в языке программирования `bash`.
3. Изучить операторы цикла `for`, `while` и `until`, оператор выбора `case`, условный оператор `if`.
4. В ходе работы написать 4 скрипта.
5. Выполнить отчет.

### 3 Выполнение лабораторной работы

- 1) Для начала я изучила команды архивации, используя команды «man zip», «man bzip2», «man tar» (рис. 3.1)

A screenshot of a terminal window with a dark background. The window title is 'kekalistratova@kekalistratova: ~'. The menu bar at the top shows 'Файл', 'Действия', 'Правка', 'Вид', and 'Справка'. The terminal shows three commands being entered: '\$ man zip', '\$ man bzip2', and '\$ man tar'. Each command is preceded by a prompt '(kekalistratova@kekalistratova)-[~]'.

```
kekalistratova@kekalistratova: ~
Файл Действия Правка Вид Справка
(kekalistratova@kekalistratova)-[~]
$ man zip
(kekalistratova@kekalistratova)-[~]
$ man bzip2
(kekalistratova@kekalistratova)-[~]
$ man tar
```

Figure 3.1: Команда man

Синтаксис команды zip для архивации файла (рис. 3.2):

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 [-aABcdDeEfFghjklLmoqrRSTuvVwXyzI@] [--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).

  A companion program (unzip(1L)) unpacks zip archives. The zip and unzip(1L) programs can work
  with archives produced by PKZIP (supporting most PKZIP features up to PKZIP version 4.6), and
  PKZIP and PKUNZIP can work with archives produced by zip (with some exceptions, notably
  streamed archives, but recent changes in the zip file standard may facilitate better compati-
  bility). zip version 3.0 is compatible with PKZIP 2.04 and also supports the Zip64 extensions
  of PKZIP 4.5 which allow archives as well as files to exceed the previous 2 GB limit (4 GB in
  some cases). zip also now supports bzip2 compression if the bzip2 library is included when zip
  is compiled. Note that PKUNZIP 1.10 cannot extract files produced by PKZIP 2.04 or zip 3.0.
  You must use PKUNZIP 2.04g or unzip 5.0p1 (or later versions) to extract them.

  See the EXAMPLES section at the bottom of this page for examples of some typical uses of zip.

Large Archives and Zip64. zip automatically uses the Zip64 extensions when files larger than 4
GB are added to an archive, an archive containing Zip64 entries is updated (if the resulting
archive still needs Zip64), the size of the archive will exceed 4 GB, or when the number of en-
tries in the archive will exceed about 64K. Zip64 is also used for archives streamed from
standard input as the size of such archives are not known in advance, but the option -fz- can

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

Figure 3.2: Синтаксис команды zip

Синтаксис команды bzip2 для архивации файла (рис. 3.3):

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
    bzip2 - decompresses files to stdout
    bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
    bzip2 [ -cdfkqstvzVL123456789 ] [ filenames ... ]
    bunzip2 [ -fkvsVL ] [ filenames ... ]
    bzipcat [ -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, with the name "original_name.bz2". Each compressed file has the same modification date, permissions, and, when possible, ownership as the corresponding original, so that these properties can be correctly restored at decompression time. File name handling is naive in the sense that there is no mechanism for preserving original file names, permissions, ownerships or dates in filesystems which lack these concepts, or have serious file name length restrictions, such as MS-DOS.

    bzip2 and bunzip2 will by default not overwrite existing files. If you want this to happen, specify the -f flag.

    If no file names are specified, bzip2 compresses from standard input to standard output. In this case, bzip2 will decline to write compressed output to a terminal, as this would be entirely incomprehensible and therefore pointless.

    bunzip2 (or bzip2 -d) decompresses all specified files. Files which were not created by bzip2 will be detected and ignored, and a warning issued. bzip2 attempts to guess the filename for the decompressed file from that of the compressed file as follows:

        filename.bz2 becomes filename

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

Figure 3.3: Синтаксис команды bzip2

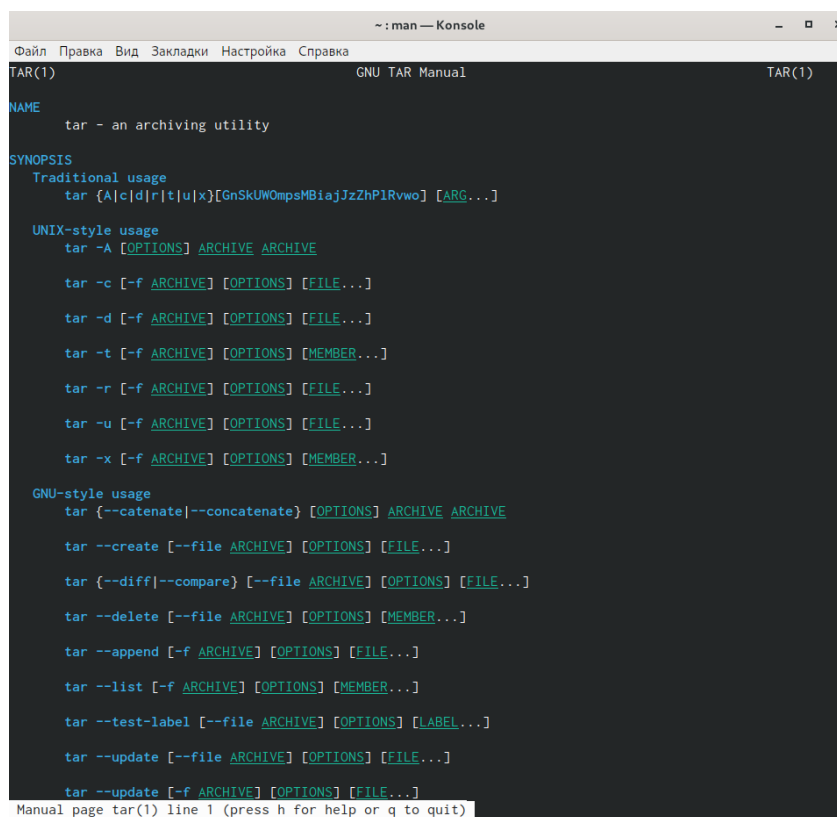
Синтаксис команды tar для архивации файла (рис. 3.4):

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}[GnSKUWmpsMBiaJzZhPlRvwo] [ARG...]

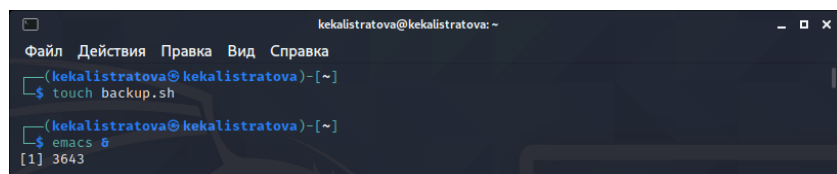
UNIX-style usage
tar -A [OPTIONS] ARCHIVE ARCHIVE

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

GNU-style usage
tar {--catenate|--concatenate} [OPTIONS] ARCHIVE ARCHIVE
tar --create [--file ARCHIVE] [OPTIONS] [FILE...]
tar {--diff|--compare} [--file ARCHIVE] [OPTIONS] [FILE...]
tar --delete [--file ARCHIVE] [OPTIONS] [MEMBER...]
tar --append [-f ARCHIVE] [OPTIONS] [FILE...]
tar --list [-f ARCHIVE] [OPTIONS] [MEMBER...]
tar --test-label [--file ARCHIVE] [OPTIONS] [LABEL...]
tar --update [--file ARCHIVE] [OPTIONS] [FILE...]
tar --update [-f ARCHIVE] [OPTIONS] [FILE...]
Manual page tar(1) line 1 (press h for help or q to quit)
```

Figure 3.4: Синтаксис команды tar

Далее я 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 &»). (рис. 3.5)



```
kekalistratova@kekalistratova: ~
Файл Действия Правка Вид Справка
(kekalistratova@kekalistratova)-[~]
$ touch backup.sh
(kekalistratova@kekalistratova)-[~]
$ emacs &
[1] 3643
```

Figure 3.5: Создание файла и открытие emacs

После 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 вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. При написании скрипта использовала архиватор bzip2. (рис. 3.6)

```
em:
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 3.6: Первый скрипт

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. 3.7) (рис. 3.8)

```
kekalistratova@kekalistratova: ~
Файл Действия Правка Вид Справка
$ ls
abc1 backup.sh~ example3.txt lab07.sh~ reports Загрузки
abc11 conf.txt example3.txt~ may ski.plases Изображения
abcd 'example1.txt#' example4.txt monthly Музыка
abcl example1.txt example4.txt~ morefun1 work Общедоступные
australia example2.txt feather my_os Видео 'Рабочий стол'
backup.sh example2.txt~ lab07.sh play Документы Шаблоны

$ chmod +x *.sh

$ ./backup.sh
Выполнено

$ cd backup/

$ ls
backup.sh.bz2

$ bunzip2 -c backup.sh.bz2
#!/bin/bash

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

Figure 3.7: Проверка работы скрипта

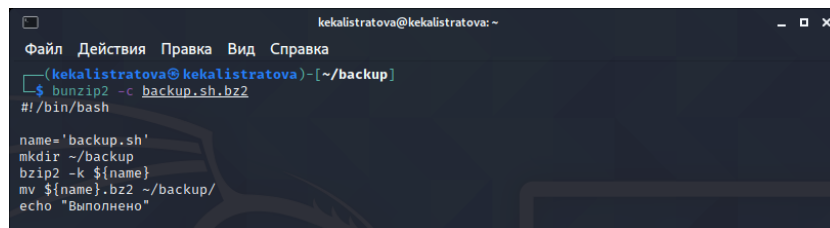
A terminal window titled 'kekalistratova@kekalistratova: ~' with a menu bar (Файл, Действия, Правка, Вид, Справка). The prompt is '(kekalistratova@ kekalistratova)-[~/backup]'. The user enters '\$ bunzip2 -c backup.sh.bz2', followed by '#!/bin/bash'. The script output is: 'name=\'backup.sh\'', 'mkdir ~/backup', 'bzip2 -k \${name}', 'mv \${name}.bz2 ~/backup/', and 'echo "Выполнено"'.

Figure 3.8: Проверка работы скрипта

- 2) Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touchprog2.sh» и «emacs &»). (рис. 3.9)

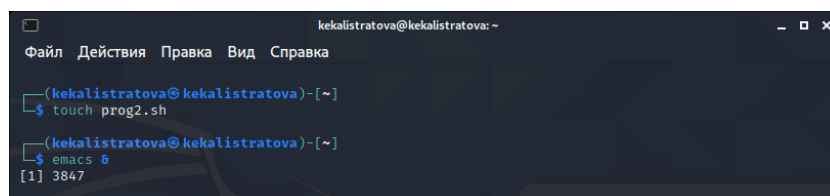
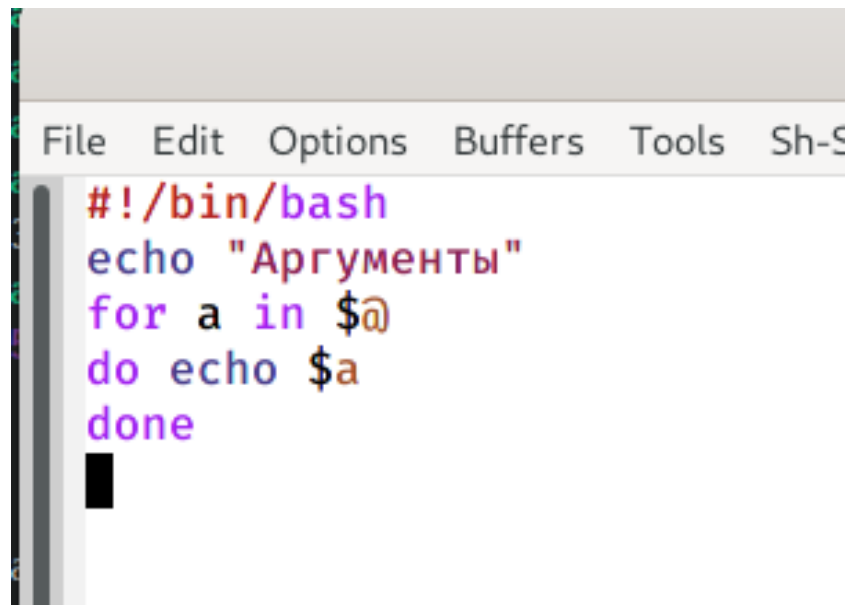
A terminal window titled 'kekalistratova@kekalistratova: ~' with a menu bar (Файл, Действия, Правка, Вид, Справка). The prompt is '(kekalistratova@ kekalistratova)-[~]'. The user enters '\$ touch prog2.sh', followed by '\$ emacs &'. The prompt changes to '[1] 3847'.

Figure 3.9: Создание файла и открытие 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. (рис. 3.10)

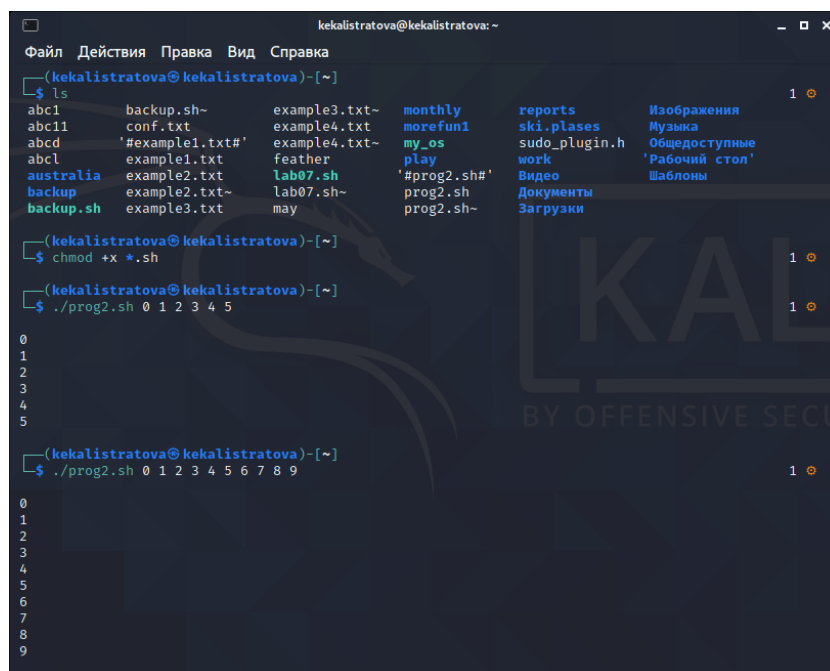
A screenshot of a text editor window with a menu bar containing 'File', 'Edit', 'Options', 'Buffers', 'Tools', and 'Sh-S'. The editor displays a shell script with the following content: 

```
#!/bin/bash
echo "Аргументы"
for a in $@
do echo $a
done
```

 The script is written in a monospaced font with syntax highlighting: the shebang is red, 'echo' is blue, 'for' is purple, 'do' is purple, and 'done' is purple. The loop body 'echo \$a' is also in a mix of colors. A black cursor is visible at the end of the 'done' line.

Figure 3.10: Второй скрипт

Проверила работу написанного скрипта (команды «./prog2.sh 0 1 2 3 4 5» и «./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12»), предварительно добавив для него право на выполнение (команда «chmod +x \*.sh»). Вводила аргументы, количество которых меньше 10 и больше 10. Скрипт работает корректно. (рис. 3.11) (рис. ??)

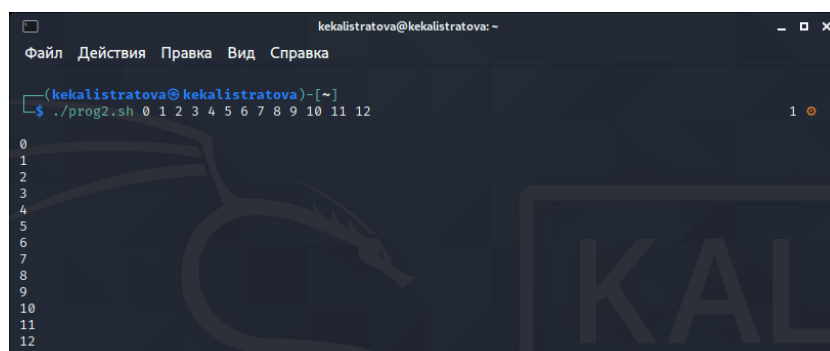


```
kekalistratova@kekalistratova: ~  
Файл Действия Правка Вид Справка  
$ ls  
abc1 backup.sh~ example3.txt~ monthly reports  
abc11 conf.txt example4.txt~ morefun1 ski.plases  
abcd 'example1.txt#' example4.txt~ my_os sudo_plugin.h  
abcl example1.txt feather play work  
australia example2.txt lab07.sh '#prog2.sh#' prog2.sh prog2.sh  
backup example2.txt~ lab07.sh~ prog2.sh  
backup.sh example3.txt may prog2.sh~  
$ chmod +x *.sh  
$ ./prog2.sh 0 1 2 3 4 5  
0  
1  
2  
3  
4  
5  
$ ./prog2.sh 0 1 2 3 4 5 6 7 8 9  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Figure 3.11: Проверка работы скрипта

### 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 «touch prls.sh» и «emacs &») (рис. 3.12)

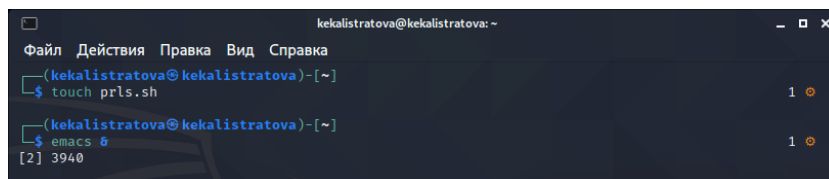


```
kekalistratova@kekalistratova: ~  
Файл Действия Правка Вид Справка  
$ ./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12
```

Figure 3.12: Создание файла и открытие emacs

Написала командный файл–аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информацию о нужном каталоге и

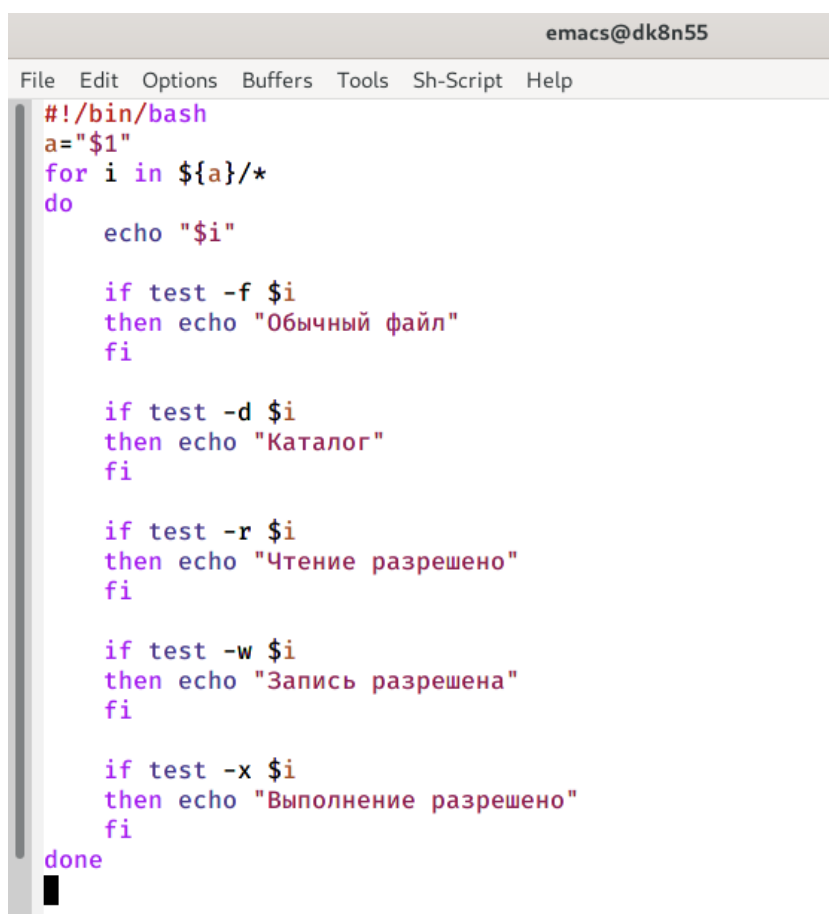
выводить информацию о возможностях доступа к файлам этого каталога. (рис. 3.13)



```
kekalistratova@kekalistratova: ~  
Файл Действия Правка Вид Справка  
(kekalistratova@kekalistratova)-[~]  
$ touch prls.sh  
(kekalistratova@kekalistratova)-[~]  
$ emacs &  
[2] 3940
```

Figure 3.13: Третий скрипт

Далее проверила работу скрипта (команда «./prls.sh~»), предварительно добавив для него право на выполнение (команда «chmod +x \*.sh»). Скрипт работает корректно. (рис. 3.14)

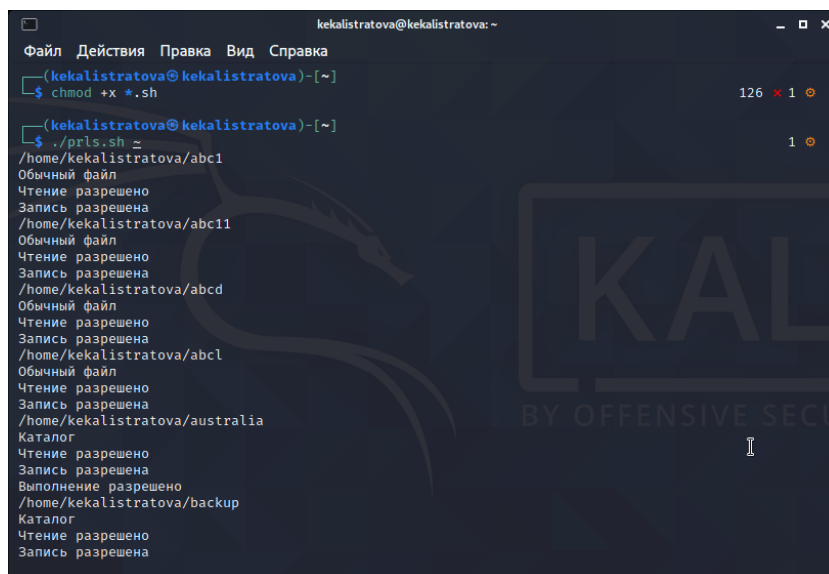


```
emacs@dk8n55  
File Edit Options Buffers Tools Sh-Script Help  
#!/bin/bash  
a="$1"  
for i in ${a}/*  
do  
    echo "$i"  
  
    if test -f $i  
    then echo "Обычный файл"  
    fi  
  
    if test -d $i  
    then echo "Каталог"  
    fi  
  
    if test -r $i  
    then echo "Чтение разрешено"  
    fi  
  
    if test -w $i  
    then echo "Запись разрешена"  
    fi  
  
    if test -x $i  
    then echo "Выполнение разрешено"  
    fi  
done
```

Figure 3.14: Проверка работы скрипта

4) Для четвертого скрипта также создала файл (команда «touch format.sh»)

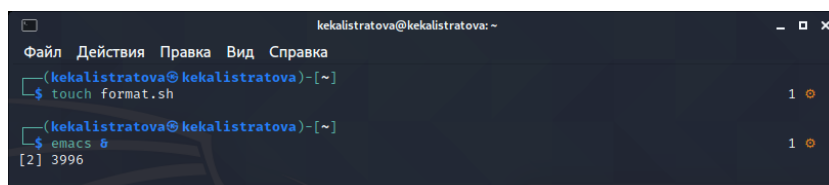
и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &»). (рис. 3.15)



```
kekalistratova@kekalistratova: ~  
Файл Действия Правка Вид Справка  
$ chmod +x *.sh  
$ ./prls.sh ~  
/home/kekalistratova/abc1  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/kekalistratova/abc11  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/kekalistratova/abcd  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/kekalistratova/abcl  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/kekalistratova/australia  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/kekalistratova/backup  
Каталог  
Чтение разрешено  
Запись разрешена
```

Figure 3.15: Создание файла и открытие emacs

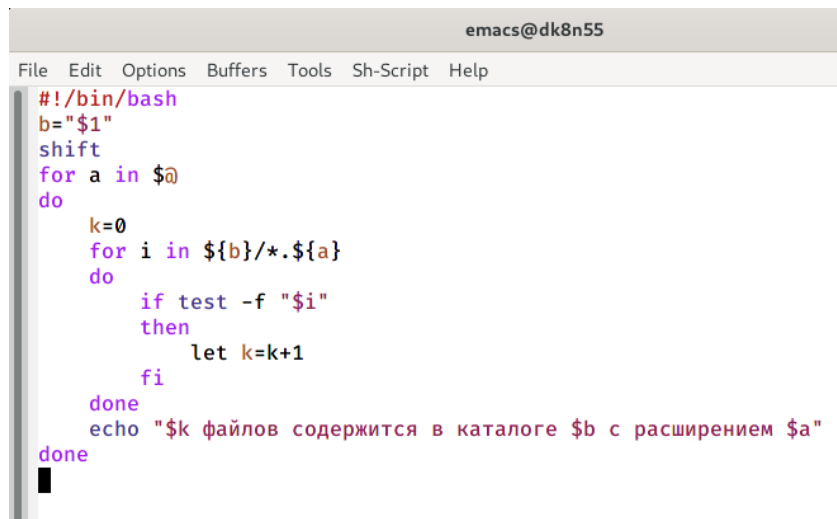
Написала командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки. (рис. 3.16)



```
kekalistratova@kekalistratova: ~  
Файл Действия Правка Вид Справка  
$ touch format.sh  
$ emacs &  
[2] 3996
```

Figure 3.16: Четвертый скрипт

Проверила работу написанного скрипта (команда «./format.sh ~ pdf jpg doc txt png»), предварительно добавив для него право на выполнение (команда «chmod +x \*.sh»). Скрипт работает корректно. (рис. 3.17)



```
emacs@dk8n55
File Edit Options Buffers Tools Sh-Script Help
#!/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 3.17: Проверка работы скрипта



## 4 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 `setc` 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:

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

## **5 Выводы**

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

## 6 Библиография

1. [https://esystem.rudn.ru/pluginfile.php/1142090/mod\\_resource/content/2/008-lab\\_shell\\_prog\\_1.pdf](https://esystem.rudn.ru/pluginfile.php/1142090/mod_resource/content/2/008-lab_shell_prog_1.pdf)
2. Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: <http://lib.rudn.ru/MegaPro2/Download/MObject/6118>.
3. Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немнюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ЕТ 60)
4. Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ЕТ 50)