

Отчёт по лабораторной работе №10

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Дарья Сергеевна Кочина

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Выводы	31

Список иллюстраций

4.1	Работа с терминалом	7
4.2	Информация о zip	8
4.3	Информация о bzip2	8
4.4	Информация о tar	9
4.5	Создание файла	10
4.6	Скрипт №1	11
4.7	Проверка работы скрипта	12
4.8	Проверка работы скрипта	12
4.9	Создание файла	13
4.10	Скрипт №2	14
4.11	Проверка работы скрипта	15
4.12	Проверка работы скрипта	15
4.13	Создание файла	16
4.14	Скрипт №3	17
4.15	Скрипт №3	18
4.16	Проверка работы скрипта	19
4.17	Проверка работы скрипта	20
4.18	Проверка работы скрипта	21
4.19	Создание файла	22
4.20	Скрипт №4	23
4.21	Проверка работы скрипта	24

1 Цель работы

Целью данной лабораторной работы является изучение основ программирования в оболочке ОС UNIX/Linux. А также приобретение практических навыков написания небольших командных файлов.

2 Задание

Изучить основы программирования в оболочке ОС UNIX/Linux. А также научиться писать небольшие командные файлы.

3 Теоретическое введение

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:

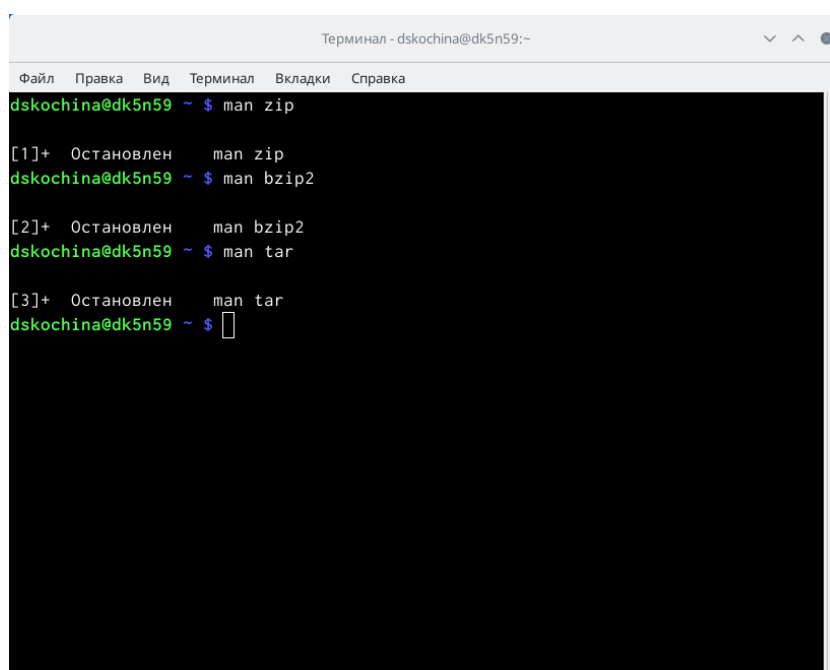
- оболочка Борна (*Bourne shell* или *sh*) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или *csh*) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или *ksh*) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- *BASH* — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

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.

4 Выполнение лабораторной работы

1. Для начала я изучила команды архивации, используя команды “manzip”, “manzip2”, “mantar”. (рис. [4.1], [4.2], [4.3], [4.4])



```
Терминал - dskochina@dk5n59:~  
Файл  Правка  Вид  Терминал  Вкладки  Справка  
dskochina@dk5n59 ~ $ man zip  
[1]+  Остановлен    man zip  
dskochina@dk5n59 ~ $ man bzip2  
[2]+  Остановлен    man bzip2  
dskochina@dk5n59 ~ $ man tar  
[3]+  Остановлен    man tar  
dskochina@dk5n59 ~ $
```

Рис. 4.1: Работа с терминалом

```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  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, MSDOS,
  OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC
  Manual page zip(1) line 1 (press h for help or q to quit)
```

Рис. 4.2: Информация о zip

```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
LBZIP2(1) User commands LBZIP2(1)
NAME
  lbzip2 - parallel bzip2 utility
SYNOPSIS
  lbzip2|bzip2 [-n WTHRS] [-k|-c|-t] [-d] [-1 .. -9] [-f] [-s] [-u] [-v]
  [-S] [ FILE ... ]

  lbunzip2|bunzip2 [-n WTHRS] [-k|-c|-t] [-z] [-f] [-s] [-u] [-v] [-S] [
  FILE ... ]

  lbzcat|bzcat [-n WTHRS] [-z] [-f] [-s] [-u] [-v] [-S] [ FILE ... ]

  lbzip2|bzip2|lbunzip2|bunzip2|lbzcat|bzcat -h
DESCRIPTION
  Compress or decompress FILE operands or standard input to regular files
  or standard output using the Burrows-Wheeler block-sorting text com-
  pression algorithm. The lbzip2 utility employs multiple threads and an
  input-bound splitter even when decompressing .bz2 files created by
  standard bzip2.
  Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Рис. 4.3: Информация о bzip2


```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  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...]

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

Рис. 4.4: Информация о tar

2. Я создала файл, в котором буду писать первый скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f». (рис. [4.5])

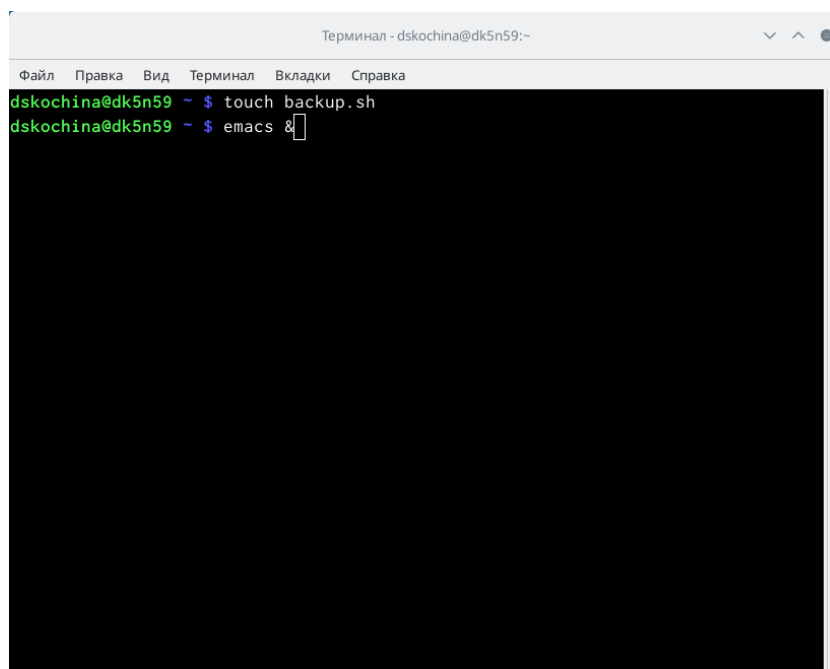


Рис. 4.5: Создание файла

3. Я 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 моём домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор `zip`, `bzip2` или `tar`. При написании скрипта использовала архиватор `bzip2`. (рис. [4.6])

```
backup.sh - GNU Emacs at dk5n59
File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help

#!/bin/bash

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


```

* 281 backup.sh Shell-script utf-8 | 8: 0 All

Warning (initialization): An error occurred while loading ‘/afs/.dk.sci.pfu.edu.~ru/home/d/s/dskochina/.config/emacs/init.el’:

File is missing: Cannot open load file, Нет такого файла или каталога, helm-conf~ig

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. [Disable showing Di](#)

[sable logging](#)

% 437 *Warnings* Special utf-8 | 8: 0 All

Рис. 4.6: Скрипт №1

4. 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»). (рис. [4.7], [4.8])

```
Терминал - dskochina@dk5n59:~/backup
Файл  Правка  Вид  Терминал  Вкладки  Справка
dskochina@dk5n59 ~ $ touch backup.sh
dskochina@dk5n59 ~ $ emacs &
[1] 4415
dskochina@dk5n59 ~ $ ls
backup.sh  GNUstep  public_html  work  Загрузки  Общедоступные
backup.sh~ my_os     ski.places   Видео  Изображения  'Рабочий стол'
bin        public    tmp          Документы  Музыка  Шаблоны
[1]+  Завершён      emacs
dskochina@dk5n59 ~ $ chmod +X *.sh
dskochina@dk5n59 ~ $ ./backup.sh
bash: ./backup.sh: Отказано в доступе
dskochina@dk5n59 ~ $ chmod +x *.sh
dskochina@dk5n59 ~ $ ./backup.sh
Выполнено
dskochina@dk5n59 ~ $ cd backup/
dskochina@dk5n59 ~/backup $ ls
backup.sh.bz2
dskochina@dk5n59 ~/backup $
```

Рис. 4.7: Проверка работы скрипта

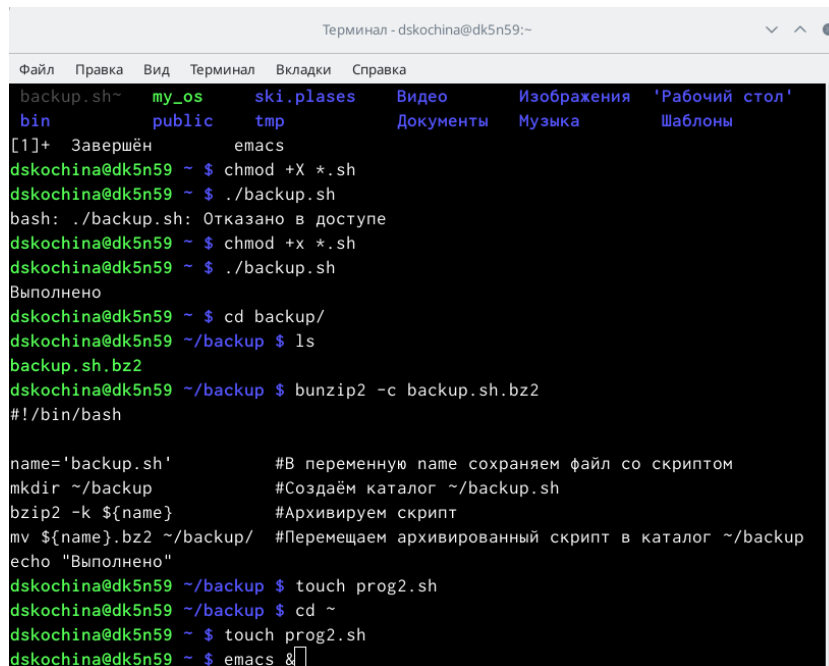
```
Терминал - dskochina@dk5n59:~/backup
Файл  Правка  Вид  Терминал  Вкладки  Справка
[1] 4415
dskochina@dk5n59 ~ $ ls
backup.sh  GNUstep  public_html  work  Загрузки  Общедоступные
backup.sh~ my_os     ski.places   Видео  Изображения  'Рабочий стол'
bin        public    tmp          Документы  Музыка  Шаблоны
[1]+  Завершён      emacs
dskochina@dk5n59 ~ $ chmod +X *.sh
dskochina@dk5n59 ~ $ ./backup.sh
bash: ./backup.sh: Отказано в доступе
dskochina@dk5n59 ~ $ chmod +x *.sh
dskochina@dk5n59 ~ $ ./backup.sh
Выполнено
dskochina@dk5n59 ~ $ cd backup/
dskochina@dk5n59 ~/backup $ ls
backup.sh.bz2
dskochina@dk5n59 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'      #В переменную name сохраняем файл со скриптом
mkdir ~/backup        #Создаём каталог ~/backup.sh
bzip2 -k ${name}       #Архивируем скрипт
mv ${name}.bz2 ~/backup/ #Перемещаем архивированный скрипт в каталог ~/backup
echo "Выполнено"
dskochina@dk5n59 ~/backup $
```

Рис. 4.8: Проверка работы скрипта

5. Создала файл, в котором буду писать второй скрипт, и открыла его в редак-

торе emacs, используя клавиши «Ctrl-x» и «Ctrl-f». (рис. [4.9])



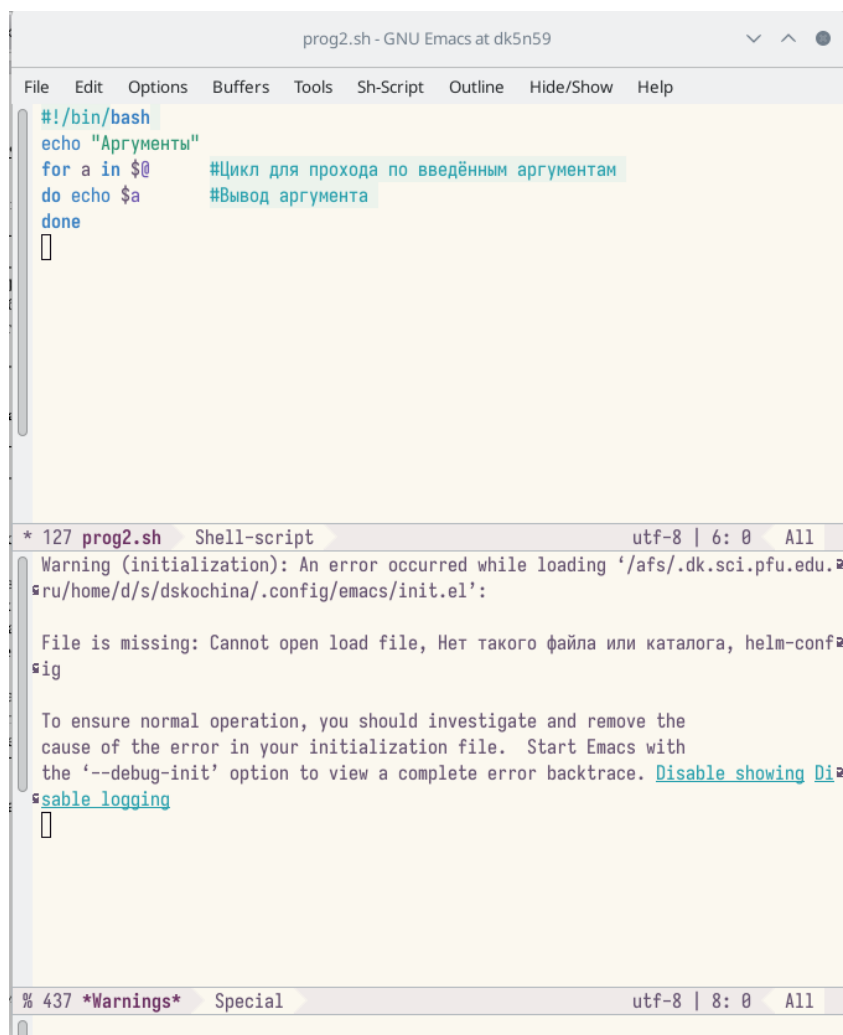
```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
backup.sh~  my_os  ski.plases  Видео  Изображения  'Рабочий стол'
bin         public  tmp         Документы  Музыка  Шаблоны
[1]+  Завершён      emacs
dskochina@dk5n59 ~ $ chmod +X *.sh
dskochina@dk5n59 ~ $ ./backup.sh
bash: ./backup.sh: Отказано в доступе
dskochina@dk5n59 ~ $ chmod +x *.sh
dskochina@dk5n59 ~ $ ./backup.sh
Выполнено
dskochina@dk5n59 ~ $ cd backup/
dskochina@dk5n59 ~/backup $ ls
backup.sh.bz2
dskochina@dk5n59 ~/backup $ bzip2 -c backup.sh.bz2
#!/bin/bash

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

dskochina@dk5n59 ~/backup $ touch prog2.sh
dskochina@dk5n59 ~/backup $ cd ~
dskochina@dk5n59 ~ $ touch prog2.sh
dskochina@dk5n59 ~ $ emacs &
```

Рис. 4.9: Создание файла

6. 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. [4.10])



```
prog2.sh - GNU Emacs at dk5n59
File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help
#!/bin/bash
echo "Аргументы"
for a in $@ #Цикл для прохода по введённым аргументам
do echo $a #Вывод аргумента
done

```

* 127 prog2.sh Shell-script utf-8 | 6: 0 All

Warning (initialization): An error occurred while loading ‘/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/.config/emacs/init.el’:

File is missing: Cannot open load file, Нет такого файла или каталога, helm-confaig

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. [Disable showing Di](#)

[sable logging](#)

% 437 *Warnings* Special utf-8 | 8: 0 All

Рис. 4.10: Скрипт №2

7. Проверила работу написанного скрипта (команды «./prog2.sh 0 1 2 3 4» и «./prog2.sh 0 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. Скрипт работает корректно. (рис. [4.11], [4.12])

```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
dskochina@dk5n59 ~ $ chmod +x *.sh
dskochina@dk5n59 ~ $ ls
backup      GNUstep     public      work        Изображения  Шаблоны
backup.sh   my_os       public_html Видео        Музыка
backup.sh~ prog2.sh    ski.plases  Документы   Общедоступные
bin         prog2.sh~   tmp         Загрузки    'Рабочий стол'
dskochina@dk5n59 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
dskochina@dk5n59 ~ $ ./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11
Аргументы
0
1
2
3
4
5
6
7
8
```

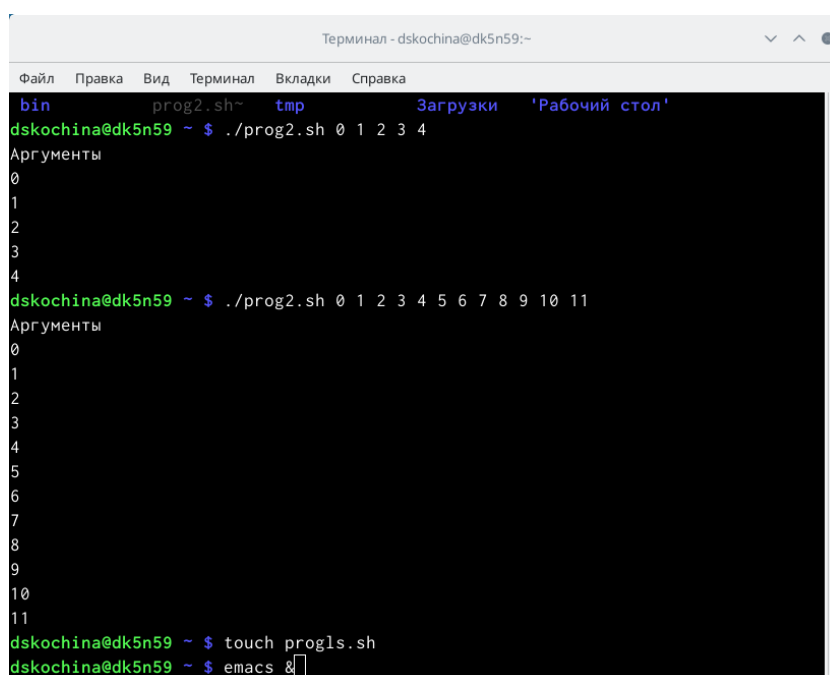
Рис. 4.11: Проверка работы скрипта

```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
backup.sh~ prog2.sh    ski.plases  Документы   Общедоступные
bin         prog2.sh~   tmp         Загрузки    'Рабочий стол'
dskochina@dk5n59 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
dskochina@dk5n59 ~ $ ./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
dskochina@dk5n59 ~ $
```

Рис. 4.12: Проверка работы скрипта

8. Создала файл, в котором буду писать третий скрипт, и открыла его в редак-

топе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touchprogl.sh» и «emacs&»). (рис. [4.13])



```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
bin    prog2.sh~  tmp      Загрузки  'Рабочий стол'
dskochina@dk5n59 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
dskochina@dk5n59 ~ $ ./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
dskochina@dk5n59 ~ $ touch progl.sh
dskochina@dk5n59 ~ $ emacs &
```

Рис. 4.13: Создание файла

9. 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. [4.14], [4.15])

The screenshot shows a GNU Emacs editor window titled 'progl.s.sh - GNU Emacs at dk5n59'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', 'Outline', 'Hide/Show', and 'Help'. The main text area contains a shell script with the following content:

```
#!/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
done
```

Below the script, a status bar indicates '* 461 progl.s.sh Shell-script utf-8 | 16: 0 Top'. The next section displays a warning message:

```
Warning (initialization): An error occurred while loading '/afs/.dk.sci.pfu.edu.
ru/home/d/s/dskochina/.config/emacs/init.el':

File is missing: Cannot open load file, Нет такого файла или каталога, helm-confi
g

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. Disable showing Di
sable logging
```

The status bar at the bottom shows '% 437 *Warnings* Special utf-8 | 8: 0 All' and 'Beginning of buffer'.

Рис. 4.14: Скрипт №3

```
progl.sh - GNU Emacs at dk5n59
File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help

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
[]

* 462 progl.sh Shell-script utf-8 | 27: 0 Bottom
Warning (initialization): An error occurred while loading '/afs/.dk.sci.pfu.edu.
ru/home/d/s/dskochina/.config/emacs/init.el':

File is missing: Cannot open load file, Нет такого файла или каталога, helm-confi
g

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. Disable showing Di
sable logging
[]

% 437 *Warnings* Special utf-8 | 8: 0 All
```

Рис. 4.15: Скрипт №3

10. Далее проверила работу скрипта (команда «./progl.sh~»), предварительно добавив для него право на выполнение (команда «chmod+x*.sh»). Скрипт работает корректно. (рис. [4.16], [4.17], [4.18])

```

dskochina@dk5n59 ~ $ chmod +x *.sh
dskochina@dk5n59 ~ $ ls
backup      my_os      public      Видео      Общедоступные
backup.sh   prog2.sh   public_html Документы   'Рабочий стол'
backup.sh~  prog2.sh~  ski.plases  Загрузки   Шаблоны
bin         proglis.sh tmp         Изображения
GNUstep     proglis.sh~ work        Музыка
dskochina@dk5n59 ~ $ ./proglis.sh ~
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/backup.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/backup.sh~
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/bin
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/GNUstep
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/my_os
Обычный файл
Чтение разрешено
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/prog2.sh
Обычный файл
Чтение разрешено

```

Рис. 4.16: Проверка работы скрипта

```
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/prog2.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/progls.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/progls.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/public
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/public_html
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/ski.plases
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/tmp
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/work
Каталог
Чтение разрешено
Запись разрешена
```

Рис. 4.17: Проверка работы скрипта

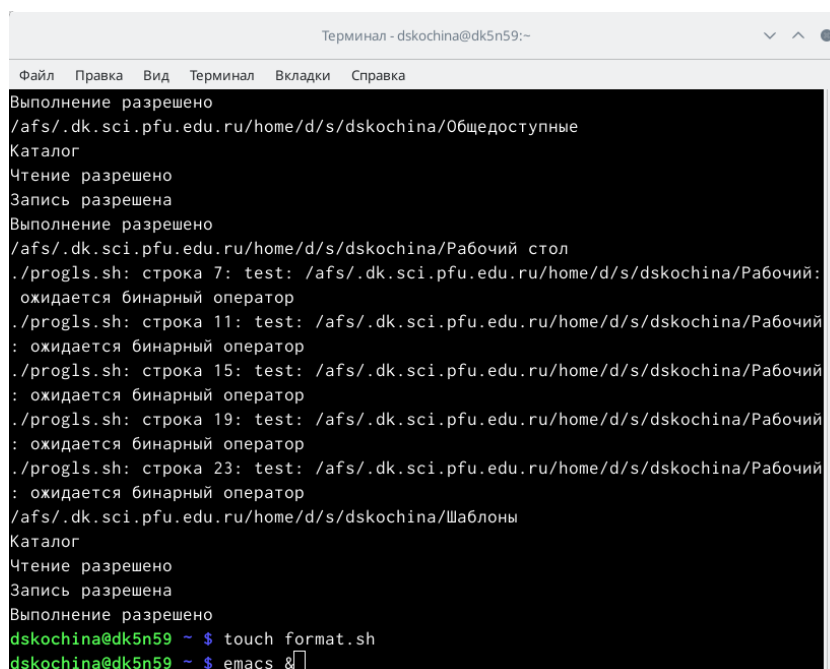
```

Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Документы
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Загрузки
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Изображения
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Музыка
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Общедоступные
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий стол
./proglis.sh: строка 7: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий: ожидается бинарный оператор
./proglis.sh: строка 11: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий: ожидается бинарный оператор
./proglis.sh: строка 15: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий: ожидается бинарный оператор
./proglis.sh: строка 19: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий: ожидается бинарный оператор
./proglis.sh: строка 23: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий: ожидается бинарный оператор
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Шаблоны
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
dskochina@dk5n59 ~ $

```

Рис. 4.18: Проверка работы скрипта

11. Для четвертого скрипта создала файл (команда «touch format.sh») и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &»). (рис. [4.19])



```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Общедоступные
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий стол
./progl.s.sh: строка 7: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий:
: ожидается бинарный оператор
./progl.s.sh: строка 11: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий
: ожидается бинарный оператор
./progl.s.sh: строка 15: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий
: ожидается бинарный оператор
./progl.s.sh: строка 19: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий
: ожидается бинарный оператор
./progl.s.sh: строка 23: test: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Рабочий
: ожидается бинарный оператор
/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/Шаблоны
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
dskochina@dk5n59 ~ $ touch format.sh
dskochina@dk5n59 ~ $ emacs &
```

Рис. 4.19: Создание файла

12. 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. [4.20])

```
format.sh - GNU Emacs at dk5n59
File Edit Options Buffers Tools Sh-Script Outline Hide/Show 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 разрешением $a"
done
```

* 219 **format.sh** Shell-script unix | 16: 0 All

Warning (initialization): An error occurred while loading '/afs/.dk.sci.pfu.edu.*/ru/home/d/s/dskochina/.config/emacs/init.el':

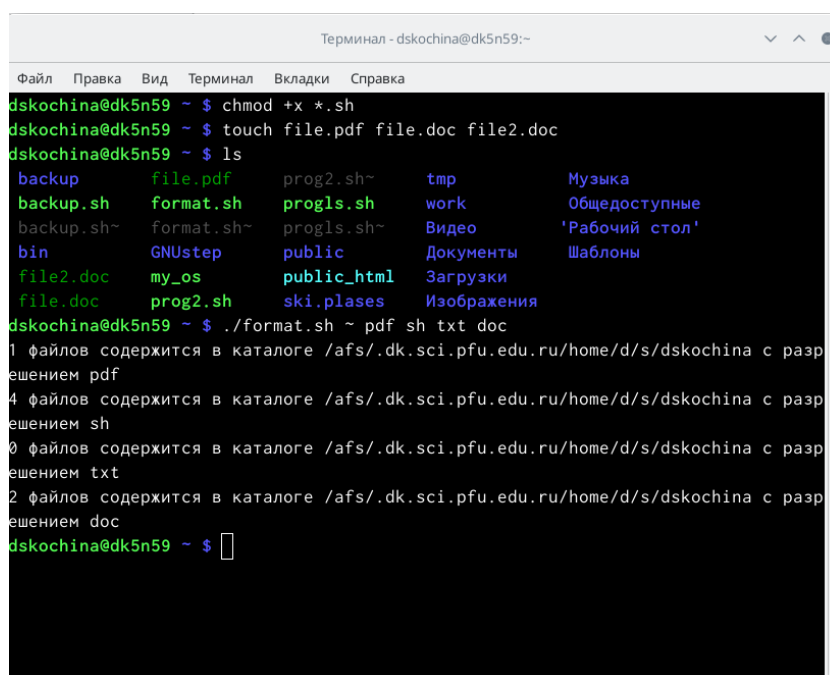
File is missing: Cannot open load file, Нет такого файла или каталога, helm-conf*
sig

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. [Disable showing Di*](#)
[sable logging](#)

% 437 *Warnings* Special utf-8 | 8: 0 All

Рис. 4.20: Скрипт №4

13. 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»). Скрипт работает корректно. (рис. [4.21])



```
Терминал - dskochina@dk5n59:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
dskochina@dk5n59 ~ $ chmod +x *.sh
dskochina@dk5n59 ~ $ touch file.pdf file.doc file2.doc
dskochina@dk5n59 ~ $ ls
backup      file.pdf    prog2.sh~   tmp         Музыка
backup.sh   format.sh  progl.s~   work        Общедоступные
backup.sh~  format.sh~ progl.s~    Видео       'Рабочий стол'
bin         GNUstep    public      Документы   Шаблоны
file2.doc   my_os      public_html Загрузки
file.doc    prog2.sh   ski.plases  Изображения
dskochina@dk5n59 ~ $ ./format.sh ~ pdf sh txt doc
1 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina с разр
ешением pdf
4 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina с разр
ешением sh
0 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina с разр
ешением txt
2 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina с разр
ешением doc
dskochina@dk5n59 ~ $
```

Рис. 4.21: 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:
 - 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

FreeSoftwareFoundation).

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. Например команда «`{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` 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 mon day trash»`. В переменные `mon` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать 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 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 `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 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`.

5 Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux. А также приобрела практические навыки написания небольших командных файлов.