

# **Отчет по лабораторной работе 10**

**Лабораторная работа 10**

Куркина Евгения Вячеславовна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	21
	Список литературы	22

## Список иллюстраций

2.1	Справка zip . . . . .	6
2.2	Справка bzip . . . . .	7
2.3	Справка tar . . . . .	8
2.4	Команда man . . . . .	8
2.5	Создание файла . . . . .	9
2.6	переход в файл в emacs . . . . .	9
2.7	Текст скрипта . . . . .	9
2.8	Сохранение файла . . . . .	9
2.9	Наличие сохраненного файла . . . . .	9
2.10	Права для выполнения . . . . .	10
2.11	Проверка наличия . . . . .	10
2.12	Корктный результат . . . . .	10
2.13	Создание нового файла . . . . .	10
2.14	Переход в файл в emacs . . . . .	11
2.15	Текст скрипта . . . . .	11
2.16	Сохранение файла . . . . .	11
2.17	Права выполнения и наличие файла . . . . .	11
2.18	Команда проверки скрипта . . . . .	12
2.19	Результат . . . . .	12
2.20	Создание файла и переход в emacs . . . . .	12
2.21	Скрипт командного файла . . . . .	13
2.22	Проверка на наличие и предоставление прав выполнения . . . . .	13
2.23	Результат проверки работы скрипта . . . . .	14
2.24	Создание файла format.sh . . . . .	14
2.25	Введенный текст скрипта . . . . .	15
2.26	Права доступа,результат проверки работы скрипта . . . . .	15

## Список таблиц

# 1 Цель работы

Здесь приводится формулировка цели лабораторной работы. Формулировки цели для каждой лабораторной работы приведены в методических указаниях.

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

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

## 2 Выполнение лабораторной работы

- 1) Получила справки по командам `zip` `bzip2` `tar`, с помощью команды `man`. (рис. 2.1)(рис. 2.2)(рис. 2.3)(рис. 2.4)

```
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, 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 com-
  press(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 ar-
  chives 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 compatibility). 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 entries 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 be used to force zip to create PKZIP 2 compatible archives
  (as long as Zip64 extensions are not needed). You must use a PKZIP 4.5 compatible unzip, such as unzip 6.0
  or later, to extract files using the Zip64 extensions.

  In addition, streamed archives, entries encrypted with standard encryption, or split archives created with
  the pause option may not be compatible with PKZIP as data descriptors are used and PKZIP at the time of
  this writing does not support data descriptors (but recent changes in the PKWare published zip standard now
  include some support for the data descriptor format zip uses).

  Mac OS X. Though previous Mac versions had their own zip port, zip supports Mac OS X as part of the Unix
  port and most Unix features apply. References to "MacOS" below generally refer to MacOS versions older
  than OS X. Support for some Mac OS features in the Unix Mac OS X port, such as resource forks, is expected
  in the next zip release.

  For a brief help on zip and unzip, run each without specifying any parameters on the command line.
  Manual page zip(1) line 1 (press h for help or q to quit)
```

Рис. 2.1: Справка `zip`

```
Файл  Правка  Вид  Закладки  Настройка  Справка
bzip2(1)                                     General Commands Manual      bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
  bzip2 - decompresses files to stdout
  bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
  bzip2 [ -cdfkqstzVL123456789 ] [ filenames ... ]
  bunzip2 [ -fkvsVL ] [ filenames ... ]
  bzip2cat [ -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
      filename.bz becomes filename
      filename.tbz2 becomes filename.tar
      filename.tbz becomes filename.tar
      anyothername becomes anyothername.out

  If the file does not end in one of the recognised endings, .bz2, .bz, .tbz2 or .tbz, bzip2 complains that it cannot guess the name of the original file, and uses the original name with .out appended.

  As with compression, supplying no filenames causes decompression from standard input to standard output.

  bunzip2 will correctly decompress a file which is the concatenation of two or more compressed files. The result is the concatenation of the corresponding uncompressed files. Integrity testing (-t) of concatenated compressed files is also supported.

  You can also compress or decompress files to the standard output by giving the -c flag. Multiple files may
```

Рис. 2.2: Справка bzip

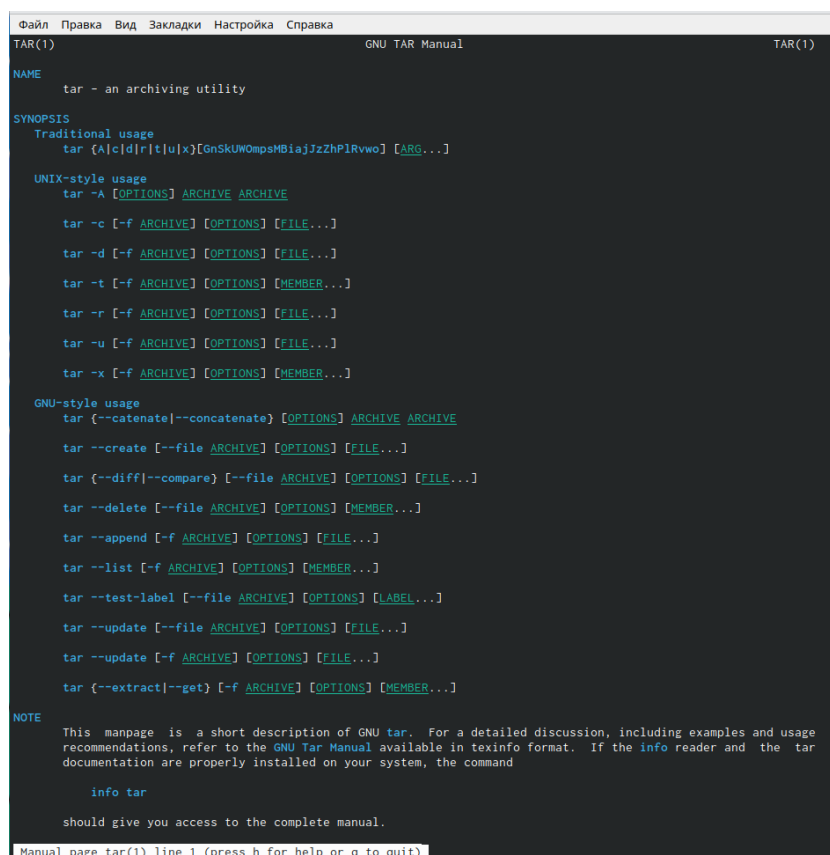


Рис. 2.3: Справка tar

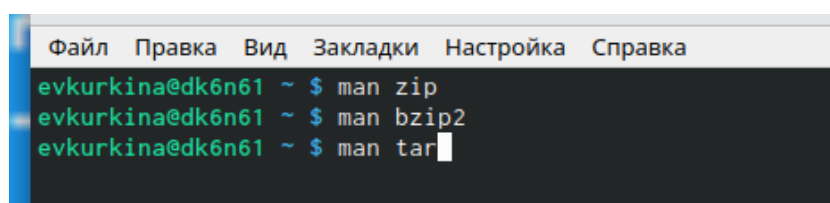


Рис. 2.4: Команда man

- 2) Создала файлы, в котором буду писать первый скрипт, назвала его backup.sh. Затем перешла в emacs(рис. 2.5), перешла в только что созданный файл(рис. 2.6). Написала необходимый скрипт(рис. 2.7), затем сохранила его(рис. 2.8). Перешла в консоль, дала право на выполнение, а после проверила работу скрипта. Убедились в том, что скрипт работает корректно.(рис. 2.9)(рис. 2.10)(рис. 2.11)(рис. 2.12)



```
evkurkina@dk6n61 ~ $ touch backup.sh
evkurkina@dk6n61 ~ $ emacs $
```

Рис. 2.5: Создание файла

```
U:--- $ All L1 (Fundamental) Ср мая 18 17:03 2.00
Welcome to GNU Emacs, one component of the GNU/Linux operating system.

[Emacs Tutorial] Learn basic keystroke commands (Учебник Emacs)
[Emacs Guided Tour] Overview of Emacs features at gnu.org
[View Emacs Manual] View the Emacs manual using Info
[Absence of Warranty] GNU Emacs comes with ABSOLUTELY NO WARRANTY
[Copying Conditions] Conditions for redistributing and changing Emacs
[Ordering Manuals] Purchasing printed copies of manuals
To quit a partially entered command, type Control-g.

This is GNU Emacs 27.2 (build 1, x86_64-pc-linux-gnu, GTK+ Version 3.24.29, cairo version 1.16.0)
of 2021-06-22
Copyright (C) 2021 Free Software Foundation, Inc.

[Dismiss this startup screen] ☐ Never show it again.

U:%%- *GNU Emacs* All L3 (Fundamental) Ср мая 18 17:03 2.00
Find file: ~/backup.sh
```

Рис. 2.6: переход в файл в emacs

```
File Edit Options Buffers Tools Sh-Script Help

#!/bin/bash

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

Рис. 2.7: Текст скрипта

```
U:%*- *Warnings* All L8 (Special) Ср мая 18 17:09 2.46
Wrote /afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/backup.sh
```

Рис. 2.8: Сохранение файла

```
evkurkina@dk6n61 ~ $ ls
abcl evkurkina.github.io feathers lab07.sh- presentation.md public_html Видео
Architecture_PC example2.txt file254 'Matrisa(54)' presentation.pdf reports Документы
australia example2.txt~ file254.txt may program1 ski.places Загрузки
backup.sh example3.txt file.txt monthly program12 temp Изображения
backup.sh~ example4.txt GNUstep my.es Programshad text.txt Музыка
bin example4.txt~ lab01 os-intro programsum tmp Общеизвестные
blog example.txt 'lab07.sh#' play program2 vozrast препрограмма
conf.txt example.txt~ lab07.sh poil public work
```

Рис. 2.9: Наличие сохраненного файла

```

evkurkina@dk6n61 ~ $ chmod +x *.sh
evkurkina@dk6n61 ~ $ ./backup.sh
Выполнено
evkurkina@dk6n61 ~ $ █

```

Рис. 2.10: Права для выполнения

```

evkurkina@dk6n61 ~ $ cd backup/
evkurkina@dk6n61 ~/backup $ ls
backup.sh.bz2
evkurkina@dk6n61 ~/backup $ █

```

Рис. 2.11: Проверка наличия

```

evkurkina@dk6n61 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
evkurkina@dk6n61 ~/backup $ █

```

Рис. 2.12: Корктный результат

- 3) Создала новый файл с названием prog2.sh, для второго скрипта, запустила emacs (рис. 2.13). Перешла в созданный файл (рис. 2.14), ввела текст скрипта (рис. 2.15), сохранила файл (рис. 2.16), затем проверила его наличие и дала права на выполнение (рис. 2.17), проверила работу скрипта(рис. 2.18)(рис. 2.19).

```

evkurkina@dk6n61 ~/backup $ cd
evkurkina@dk6n61 ~ $ touch prog2.sh
evkurkina@dk6n61 ~ $ emacs █

```

Рис. 2.13: Создание нового файла

```
U:%%- *GNU Emacs* Top L5 (Fundamental) Ср мая 18 17:15 1.20
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) Ср мая 18 17:15 1.20
Find file: ~/blog2.sh
```

Рис. 2.14: Переход в файл в emacs

```
emacs@dk6n61
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
echo "Аргументы"
for a in #@
do echo $a
done
```

Рис. 2.15: Текст скрипта

```
U:%%- *Warnings* All L8 (Special) Ср мая 18 17:17 1.37
Wrote /afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/blog2.sh
```

Рис. 2.16: Сохранение файла

```
Файл Правка Вид Закладки Настройка Справка
evkurkina@dk6n61 ~ $ chmod +x *.sh
evkurkina@dk6n61 ~ $ ls
abcl evkurkina.github.io file254.txt my_os programsum work
Architecture_PC example2.txt file.txt os-intro programm2 Видео
australia example2.txt~ GNUstep play public Документы
backup example3.txt lab01 poil public_html Загрузки
backup.sh example4.txt '#lab07.sh#' presentation.md reports Изображения
backup.sh~ example4.txt~ lab07.sh presentation.pdf ski.plases Музыка
bin example.txt lab07.sh~ prog2.sh program1 temp Общедоступные
blog example.txt~ 'Matrisa(54)' program12 text.txt препрограмма
blog2.sh feathers may Programploshad tmp
conf.txt file254 monthly
evkurkina@dk6n61 ~ $
```

Рис. 2.17: Права выполнения и наличие файла

```
evkurkina@dk6n61 ~ $ ./prog2.sh 0 1 2 3 4
```

Рис. 2.18: Команда проверки скрипта

```
evkurkina@dk6n61 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
evkurkina@dk6n61 ~ $
```

Рис. 2.19: Результат

- 4) Создала файл с названием `progl.sh` для третьего скрипта, открыла его в `emacs`(рис. 2.20). Написала командный файл, являющимся аналогом для команды `ls`, для вывода необходимой информации о нужном каталоге и возможностях доступа к файлам этого каталога(рис. 2.21), сохранила файл. Далее перешла в консоль, проверила наличие файла, дала права выполнения (рис. 2.22), убедилась в корректной работе скрипта(рис. 2.23).

```
evkurkina@dk6n61 ~ $ touch progl.sh
evkurkina@dk6n61 ~ $ emacs &
[2] 10007
evkurkina@dk6n61 ~ $
```

Рис. 2.20: Создание файла и переход в `emacs`

Рис. 2.21: Скрипт командного файла

```
evkurkina@dk6n61 ~ $ chmod +x *.sh
evkurkina@dk6n61 ~ $ ls
abc1      evkurkina.github.io  file.txt      play          program2      Документы
Architecture_PC  example2.txt~        GNUstep      poil          public        Загрузки
australia  example2.txt~        lab01        presentation.md public_html   Изображения
backup     example3.txt~        'lab07.sh#'  presentation.pdf public_html   Музыка
backup.sh  example4.txt~        lab07.sh     prog2.sh      reports       Обще-
backup.sh~ example4.txt~        lab07.sh~    '#progals.sh#' ski.plases    дост-
bin        example.txt~         'Matrisa(54)' program1       temp          пные
blog       example.txt~         may          program12     text.txt      пре-
blog2.sh   feathers            monthly      Programploshad tmp            про-
blog2.sh~  file254            my_os        programsum    vozrast       про-
conf.txt   file254.txt        os-intro     proglis.sh    work          гра-
evkurkina@dk6n61 ~ $ ./proglis.sh
```

Рис. 2.22: Проверка на наличие и предоставление прав выполнения

```

evkurkina@dk6n61 ~ $ ./progals.sh ~
/afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/abc1
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/Architecture_PC
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/australia
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/backup.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/backup.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/e/v/evkurkina/bin

```

Рис. 2.23: Результат проверки работы скрипта

- 5) Для последнего скрипта создала файл с названием format.sh(рис. 2.24). Перешла в emacs, в созданный файл, написала скрипт, который получает на входе формат файла, а затем считает количество таких файлов в указанной директории (рис. 2.25). Затем вернулась в консоль, дала права выполнения и проверила наличие, затем проверила исправную работу скрипта (рис. 2.26).

```

evkurkina@dk6n61 ~ $ touch format.sh
evkurkina@dk6n61 ~ $ emacs &

```

Рис. 2.24: Создание файла format.sh

```
Приложения Места GNU Emacs
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 разришением $a"
done
```

Рис. 2.25: Введенный текст скрипта

```
evkurkina@dk6n61 ~ $ chmod +x *.sh
evkurkina@dk6n61 ~ $ touch file.pdf file.doc file2.doc
evkurkina@dk6n61 ~ $ ls
abcl          conf.txt      file254      'lab07.sh#'  presentation.md  public      Документы
Architecture_PC evkurkina.github.io file254.txt  lab07.sh     presentation.pdf  public_html  Загрузки
australia     example2.txt  file2.doc   lab07.sh~    prog2.sh         reports     Изображения
backup        example2.txt~ file.doc     'Matrisa(54)' progals.sh       ski_places   Музыка
backup.sh     example3.txt  file.pdf    may          program1         temp        Общедоступны
backup.sh~    example4.txt  file.txt    monthly      program12       text.txt    tmp
bin           example4.txt~ format.sh~   my_os        Programploshad  tmp         возраст
blog          example.txt   format.sh   os-intro     programsum       vozrast
blog2.sh     example.txt~ GNUstep     play         progls.sh      work
blog2.sh~    feathers      lab01       poil         program2        Video

evkurkina@dk6n61 ~ $ ./format.sh ~ pdf sh txt doc
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
0 файлов содержится в каталоге /afs/dk.sci.pfu.edu.ru/home/e/v/evkurkina с разришением pdf
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
0 файлов содержится в каталоге /afs/dk.sci.pfu.edu.ru/home/e/v/evkurkina с разришением sh
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
0 файлов содержится в каталоге /afs/dk.sci.pfu.edu.ru/home/e/v/evkurkina с разришением txt
./format.sh: строка 10: test -f: команда не найдена
./format.sh: строка 10: test -f: команда не найдена
0 файлов содержится в каталоге /afs/dk.sci.pfu.edu.ru/home/e/v/evkurkina с разришением doc
evkurkina@dk6n61 ~ $
```

Рис. 2.26: Права доступа, результат проверки работы скрипта

## 6) 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, с

C-оболочка (или `csh`) –настройка на оболочкой Борна, использующая Подобный синтаксис  
Оболочка Корна (или `ksh`) – напоминает оболочку C, но операторы управления программами  
BASH – сокращение от BourneAgainShell(опять оболочка Борна), в основе своей совместимая

2). POSIX (Portable Operating System Interface for Computer Environments ) – набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX - совместимые оболочки разработаны на базе оболочки Корна.

3). Командный процессор `bash` обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда «`mark=/usr/andy/bin`» присваивает значение строки символов `/usr/andy/bin` переменной `mark` типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол `.`, «`mva file{mark}`» переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Оболочка `bash` позволяет работать с массивами. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «`set -A states Delaware Michigan "New Jersey"`». Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента.

4). Оболочка `bash` поддерживает встроенные арифметические функции. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение – это единственный терм (`term`), обычно целочисленный. Команда `let` берет два операнда



и присваивает их переменной. Команда `read` позволяет читать значения переменных со стандартного ввода: «`echo "Please enter Month and Day of Birth ?"`» «`read mon day trash`». В переменные `mon` и `day` будут считаны соответствующие значения, 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

`PS1` и `PS2`: эти переменные предназначены для отображения промптера командного про-  
то интерактивная программа, запущенная командным процессором, требует ввода, то и

`HOME`: имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов

`IFS`: последовательность символов, являющихся разделителями в командной строке, нап

`MAIL`: командный процессор каждый раз перед выводом на экран промптера проверяет со

`TERM`: тип используемого терминала.

`LOGNAME`: содержит регистрационное имя пользователя, которое устанавливается автом

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 предшеству-

ющего 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 пр

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

--значение флагов командного процессора;

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

\${#name} –возвращает целое значение длины строки в переменной name;

\${name[n]} –обращение к n-му элементу массива;

\${name[\*]} –перечисляет все элементы массива, разделённые пробелом;

\${name[@]} –то же самое, но позволяет учитывать символы пробелы в самих переменных

\${name:-value} –если значение переменной name не определено, то оно будет заменен

\${name:value} –проверяется факт существования переменной;

\${name=value} –если name не определено, то ему присваивается значение value;

\${name?value} –останавливает выполнение, если имя переменной не определено, и выв

`${name+value}` -это выражение работает противоположно `${name-value}`. Если переменная `name` имеет значение, то `value` добавляется к значению `name`.  
`${name#pattern}` -представляет значение переменной `name` с удалённым самым коротким подстроком, соответствующим `pattern`.  
`${#name[*]}` и `${#name[@]}`-эти выражения возвращают количество элементов в массиве `name`.

## **3 Выводы**

Во время выполнения данной лабораторной работы, я изучила основы программирования в оболчке ОС UNIX/Linux, найчилась писать небольшие командные файлы.

## **Список литературы**