

# Лабораторная работа 10

## Отчет по лабораторной работе 10

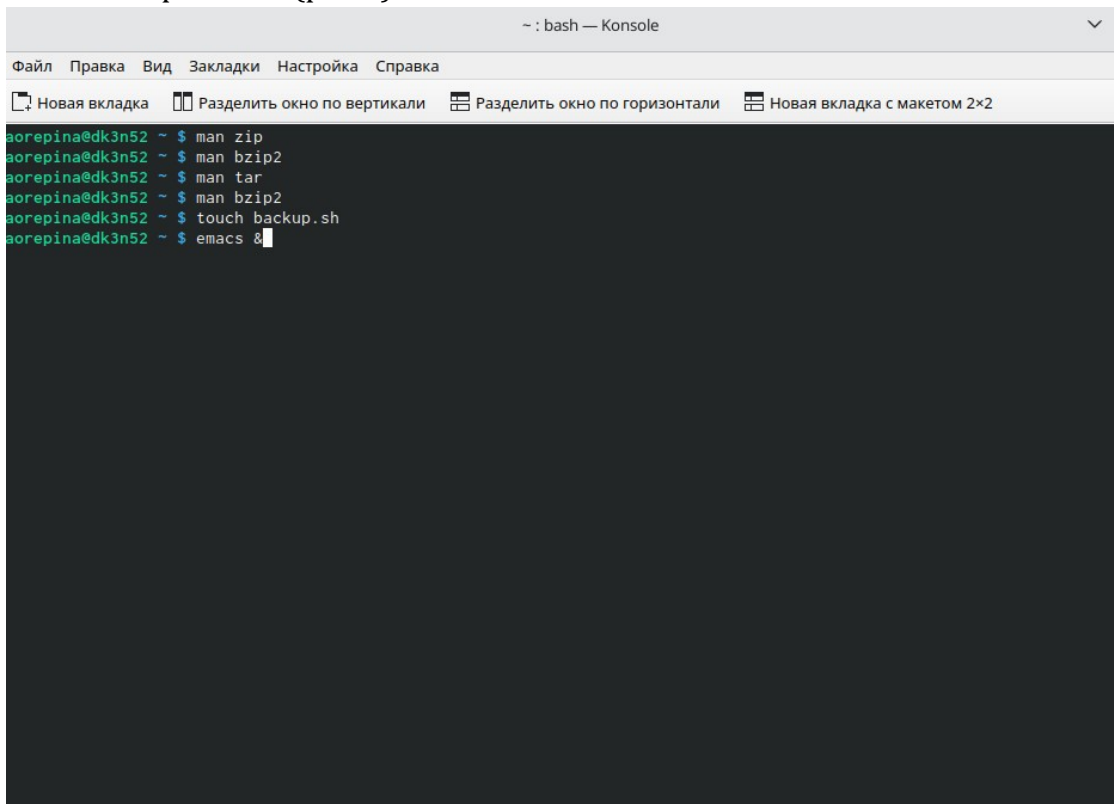
Репина Ангелина Олеговна

### Цель работы

Цель работы - изучение основ программирования в оболочке Linux и получение навыков по созданию командных файлов

### Выполнение лабораторной работы

- 1) Для начала я изучила команды архивации, используя команды `man bzip2` `man zip` `man tar` (рис 1)



```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
Новая вкладка  Разделить окно по вертикали  Разделить окно по горизонтали  Новая вкладка с макетом 2x2
aorepina@dk3n52 ~ $ man zip
aorepina@dk3n52 ~ $ man bzip2
aorepina@dk3n52 ~ $ man tar
aorepina@dk3n52 ~ $ man bzip2
aorepina@dk3n52 ~ $ touch backup.sh
aorepina@dk3n52 ~ $ emacs &
```

1

Синтаксис команды `zip` для архивации файла: `zip [опции] [имя файла.zip] [файлы ил папки, которые будем архивировать]` Синтаксис команды `zip` для разархивации файла: `unzip [опции] [файл архива.zip] [файлы] -x [исключить] -d [папка]` (рис 2)

```
~: man — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
[иконка] Новая вкладка  [иконка] Разделить окно по вертикали  [иконка] Разделить окно по горизонтали  [иконка] Новая вкладка с макетом 2x2  >> >>

ZIP(1L)
NAME
    zip - package and compress (archive) files

SYNOPSIS
    zip [-aABcdDeEffGghjklLmoqrRSTuvVwXyz!@&] [--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 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 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),

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

2

Синтаксис [bzip2](#) для архивации файла: [bzip2](#) [опции][имена файлов] Синтаксис команды [bzip2](#) для разархивации файла: [bunzip2](#)[опции][архивы.bz2] (рис 3)

```
~: man — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
[icon] Новая вкладка  [icon] Разделить окно по вертикали  [icon] Разделить окно по горизонтали  [icon] Новая вкладка с макетом 2x2  » »
bzip2(1)                                     General Commands Manual                                     bzip2(1)

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

SYNOPSIS
bzip2 [ -cdkqstvwVL123456789 ] [ filenames ... ]
bunzip2 [ -fkvsVL ] [ filenames ... ]
bzip2recover [ -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 compres-
sors, 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 cor-
rectly 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.

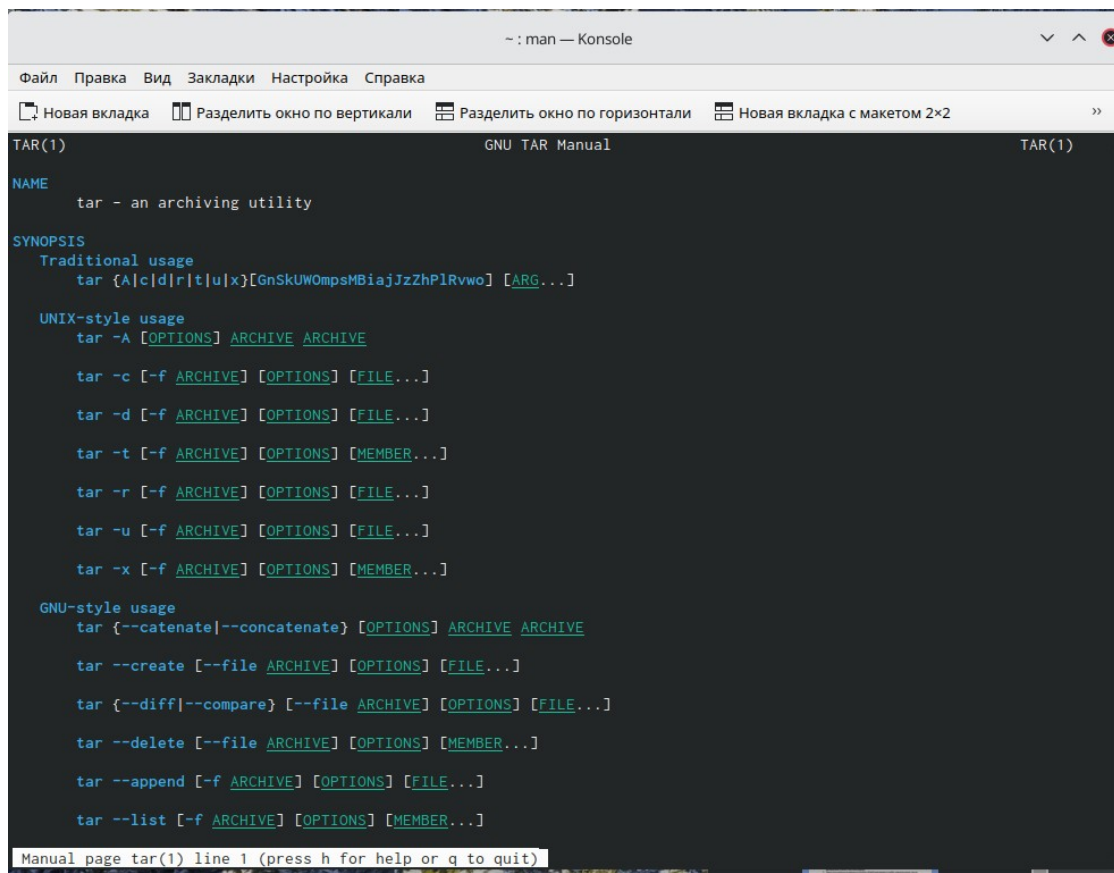
If no file names are specified, bzip2 compresses from standard input to standard output. In this case, bzip2
will decline to write compressed output to a terminal, as this would be entirely incomprehensible and there-
fore pointless.

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

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

3

Синтаксис команды tar для архивации: tar [опции][архив.tar][файлы для архивации]  
Синтаксис команды для разархивации: tar [опции] [архив.tar] (рис 4)



The image shows a terminal window titled "man — Konsole". The menu bar includes "Файл", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The toolbar has icons for "Новая вкладка", "Разделить окно по вертикали", "Разделить окно по горизонтали", and "Новая вкладка с макетом 2x2". The main content area displays the manual page for "tar(1)".

```
TAR(1) GNU TAR Manual TAR(1)

NAME
    tar - an archiving utility

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

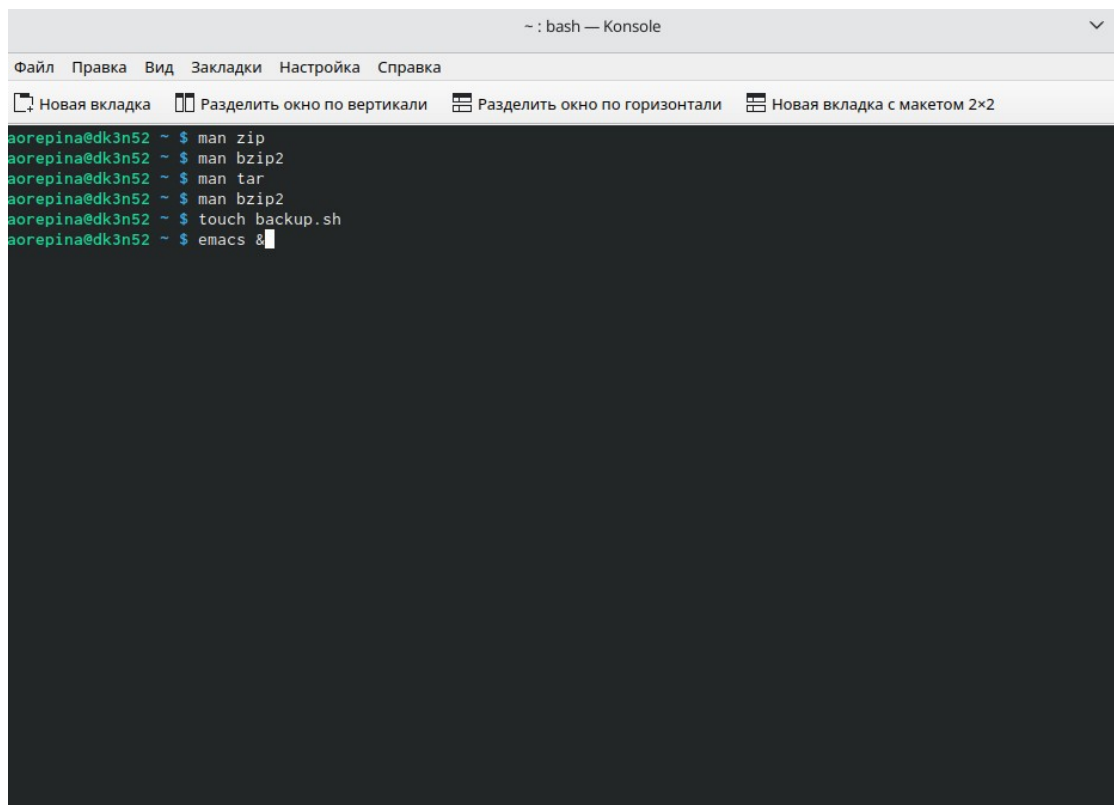
    GNU-style usage
    tar [--catenate|--concatenate] [OPTIONS] ARCHIVE ARCHIVE

    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...]
```

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

4

Создала файл, в котором в будущем буду писать первый скрипт, и открыла его в редакторе emacs, используя клавиши Ctrl-x Ctrl-f.(рис 5)



```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
Новая вкладка  Разделить окно по вертикали  Разделить окно по горизонтали  Новая вкладка с макетом 2x2
aorepina@dk3n52 ~ $ man zip
aorepina@dk3n52 ~ $ man bzip2
aorepina@dk3n52 ~ $ man tar
aorepina@dk3n52 ~ $ man bzip2
aorepina@dk3n52 ~ $ touch backup.sh
aorepina@dk3n52 ~ $ emacs &
```

5

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 моем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор ( я выбрала при написании bzip2) (рис 6)

```
emacs@dk3n52
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

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

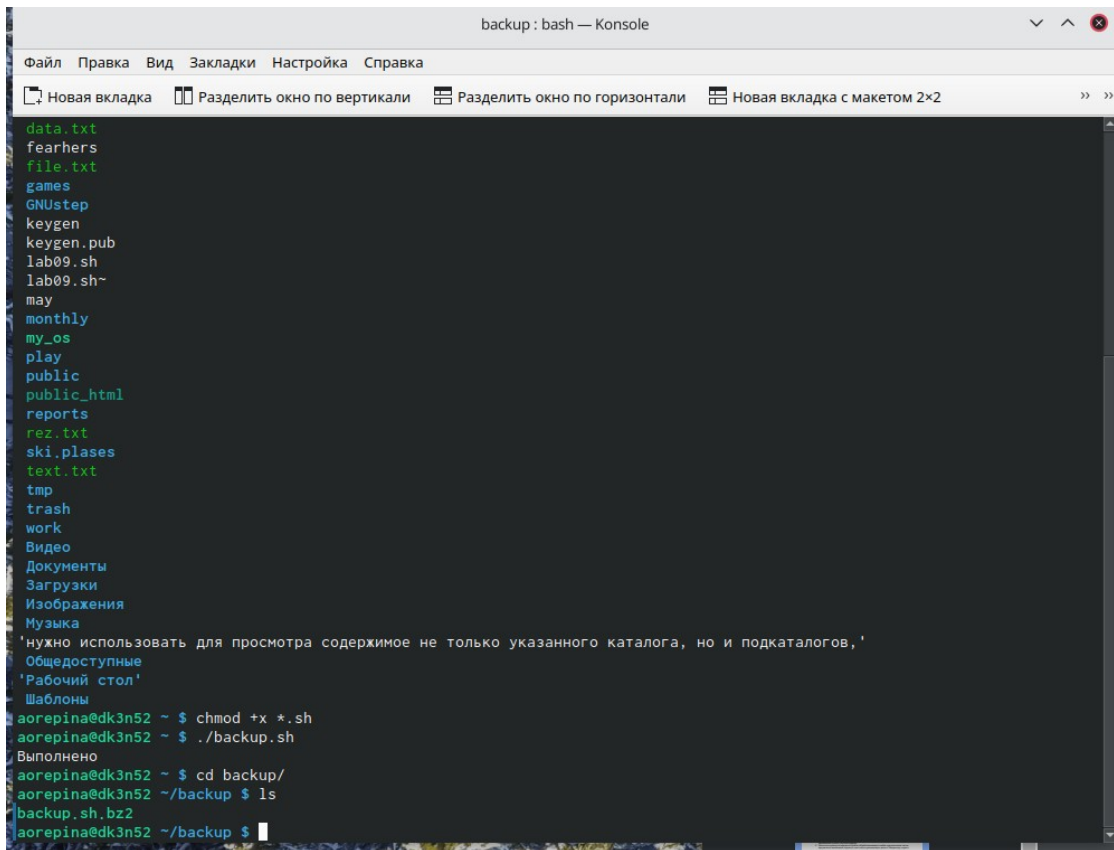
U:--- backup.sh All L8 (Shell-script[sh]) Чт мая 5 13:56 0.97
Warning (initialization): An error occurred while loading '~/.emacs':

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.
```

6

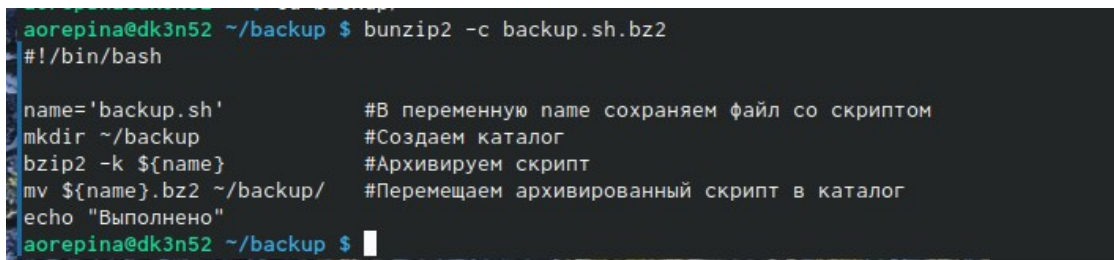
Проверила работу скрипта (команда `./backup.sh`), предварительно добавив для него права на выполнение (`chmod +x *.sh`). Проверила, появился ли каталог `backup/`, перейдя в него (команда `cd backup/`), посмотрела его содержимое (`ls`) и просмотрела содержимое архива (`bunzip2 -backup.sh.bz2`). Скрипт работает корректно (рис 7,8)



```
backup : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
Новая вкладка  Разделить окно по вертикали  Разделить окно по горизонтали  Новая вкладка с макетом 2x2

data.txt
fearhers
file.txt
games
GNUstep
keygen
keygen.pub
lab09.sh
lab09.sh~
may
monthly
my_os
play
public
public_html
reports
rez.txt
ski.places
text.txt
tmp
trash
work
Видео
Документы
Загрузки
Изображения
Музыка
'нужно использовать для просмотра содержимое не только указанного каталога, но и подкаталогов,'
Общедоступные
'Рабочий стол'
Шаблоны
aorepina@dk3n52 ~ $ chmod +x *.sh
aorepina@dk3n52 ~ $ ./backup.sh
Выполнено
aorepina@dk3n52 ~ $ cd backup/
aorepina@dk3n52 ~/backup $ ls
backup.sh.bz2
aorepina@dk3n52 ~/backup $
```

7

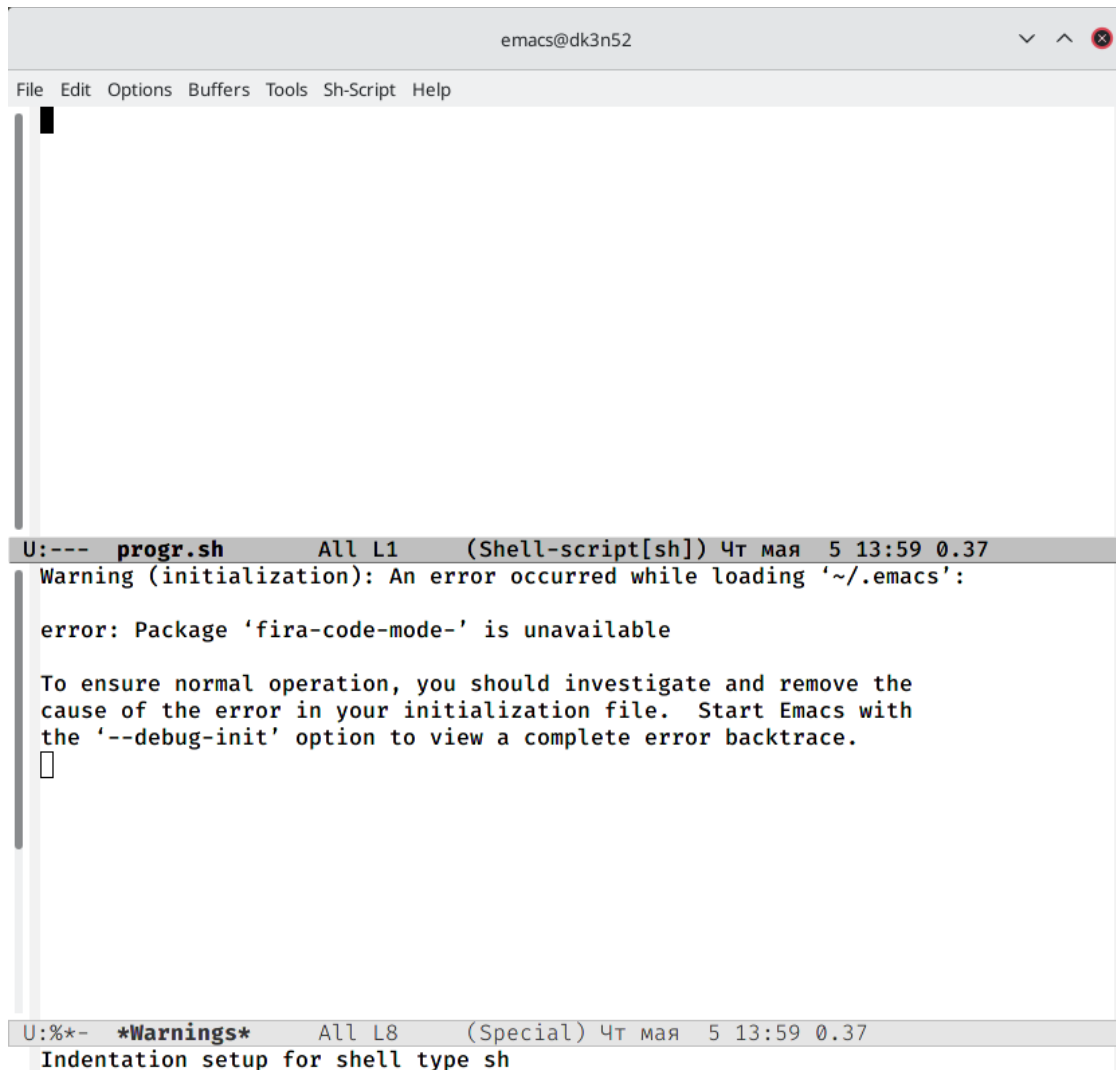


```
aorepina@dk3n52 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

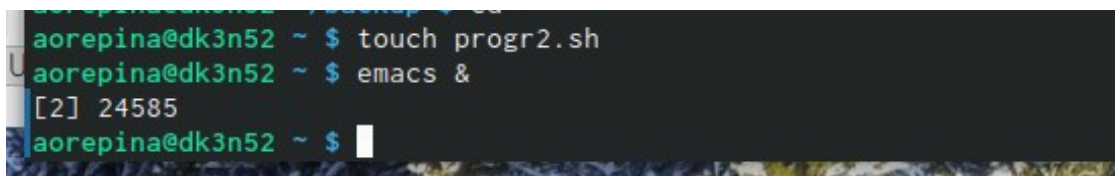
name='backup.sh'           #В переменную name сохраняем файл со скриптом
mkdir ~/backup             #Создаем каталог
bzip2 -k ${name}           #Архивируем скрипт
mv ${name}.bz2 ~/backup/   #Перемещаем архивированный скрипт в каталог
echo "Выполнено"
aorepina@dk3n52 ~/backup $
```

8

- 2) Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs, используя клавиши Ctrl-x Ctrl-f (touch progr2.sh и emacs &) (рис 9, 10)



9



10

Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее 10. Например, скрипт может последовательно распечатать значения всех переданных аргументов (рис 11, 12, 13,14)



```
emacs@dk3n52
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
echo "Аргументы"
for a in $@      #Цикл для прохода по введенным аргументам
do echo $a      #Вывод аргумента
done

U:--- progr.sh      All L6      (Shell-script[sh]) Чт мая 5 14:01 1.06
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
[]

U:%*- *Warnings*      All L8      (Special) Чт мая 5 14:01 1.06
Wrote /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/progr.sh
```

```

aorepina@dk3n52 ~ $ touch progr2.sh
aorepina@dk3n52 ~ $ emacs &
[2] 24585
aorepina@dk3n52 ~ $ chmod +x *.sh
aorepina@dk3n52 ~ $ ls
-
'2022-04-21 10-37-53.mkv'
abc
abc1
australia
backup
backup.sh
backup.sh~
c.cpp
conf.txt
course-directory-student-template
data.txt

```

12

```

aorepina@dk3n52 ~ $ ./progr.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
aorepina@dk3n52 ~ $

```

13

```

aorepina@dk3n52 ~ $ ./progr.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
aorepina@dk3n52 ~ $

```

14

Проверила работу написанного скрипта, предварительно добавив для него право на выполнение. Вводила аргументы количество которых меньше 10 и больше 10. Скрипт работает корректно (рис 15,16)

```
aorepina@dk3n52 ~ $ ./progr.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
aorepina@dk3n52 ~ $
```

15

```
aorepina@dk3n52 ~ $ ./progr.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
aorepina@dk3n52 ~ $
```

16

- 3) Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs. (рис 17)

```
aorepina@dk3n52 ~ $ touch proglis.sh
[2]+  Завершён      emacs
aorepina@dk3n52 ~ $ emacs &
```

17

Написала командный файл - аналог команды ls без использования этой самой команды и команды dir). Он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога (рис 18,19)

```
emacs@dk3n52
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
a="$1"          #В переменную а сохраняем путь дол заданного каталога
for i in ${a}/*  #Цикл, который проходит по всем каталогам и файлам
do
    echo "$i"

    if test -f $i
    then echo "Обычный файл"
    fi

    if test -d $i
    then echo "Каталог"
    fi

    if test -r $i
    then echo "Чтение разрешено"
    fi

    if test -w $i
    then echo "Запись разрешена"
    fi

U:**~ progls.sh Top L21 (Shell-script[sh]) Чт мая 5 14:12 0.53
Warning (initialization): An error occurred while loading '~/.emacs':
```

18

```
emacs
File Edit Options Buffers Tools Sh-Script Help
    then echo "Запись разрешена"
    fi

    if test -x $i
    then echo "Выполнение разрешено"
    fi
done
```

19

Далее проверила работу скрипта, предварительно добавив для него право на выполнение. Скрипт работает корректно (рис 20, 21)

```
aorepina@dk3n52 ~ $ chmod +x *.sh
aorepina@dk3n52 ~ $ ls
```

20

```

aorepina@dk3n52 ~ $ ./proglis.sh ~
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/-
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/2022-04-21 10-37-53.mkv
./proglis.sh: строка 7: test: /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/2022-04-21: ожидается бинарный оператор
./proglis.sh: строка 11: test: /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/2022-04-21: ожидается бинарный оператор
./proglis.sh: строка 15: test: /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/2022-04-21: ожидается бинарный оператор
./proglis.sh: строка 19: test: /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/2022-04-21: ожидается бинарный оператор
./proglis.sh: строка 23: test: /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/2022-04-21: ожидается бинарный оператор
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/abc
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/abc1
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/australia
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/backup.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/backup.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/c.cpp
Обычный файл
Чтение разрешено

```

21

- 4) Для четвертого скрипта создала файл и открыла его в редакторе emacs (рис 22)

```

Выполнение разрешено
aorepina@dk3n52 ~ $ touch format.sh
[1]- Завершён      emacs
aorepina@dk3n52 ~ $ emacs &

```

22

Написала командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории. Путь к директории также передается в виде аргумента командной строки (рис 23)

The screenshot shows an Emacs editor window titled 'emacs@dk3n52'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The main text area contains a shell script named 'format.sh' with the following content:

```
#!/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
```

Below the script, the output of running the script is shown in a buffer titled 'U:--- format.sh All L16 (Shell-script[sh]) Чт мая 5 14:32 1.84'. The output includes a warning and an error message:

```
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.
```

At the bottom, another buffer titled 'U:%\*- \*Warnings\* All L8 (Special) Чт мая 5 14:32 1.84' shows the command executed:

```
Wrote /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina/format.sh
```

The status bar at the very bottom indicates the user is 'aorepina@dk3n52' in a shell.

23

Проверила работу написанного скрипта, предварительно добавив для него право на выполнение, а также создав дополнительные файлы с разными расширениями. Скрипт работает корректно (рис 24,25)

```
aorepina@dk3n52 ~ $ chmod +x *.sh
aorepina@dk3n52 ~ $ touch file.pdf file.docx file2.docx
aorepina@dk3n52 ~ $ ls
-
'2022-04-21 10-37-53.mkv'
abc
abc1
australia
backup
backup.sh
backup.sh~
C CDD
```

24

```
aorepina@dk3n52 ~ $ ./format.sh ~ pdf txt docx
1 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina с разрешением pdf
5 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina с разрешением txt
2 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/o/aorepina с разрешением docx
aorepina@dk3n52 ~ $
```

25

## Выводы

В ходе выполнения лабораторной работы я изучила основы программирования в оболочке Linux и научилась писать небольшие командные файлы

## Итоговые 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 `{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 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, 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 `cd` вводится без аргументов, то происходит переход в каталог, указанный в этой переменной. 4. `IFS`: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (`newline`). 5. `MAIL`: командный



процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение `Youhavemail` (у Вас есть почта). 6. `TERM`: тип используемого терминала. 7. `LOGNAME`: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему. 8). Такие символы, как `' < > ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл. 9). Снятие специального смысла с метасимвола называется экранированием мета символа. Экранирование может быть осуществлено с помощью предшествующего мета символу символа `\`, который, в свою очередь, является мета символом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме `$, ', , "`. Например, `-echo*` выведет на экран символ `*`, `-echoab'|'cd` выведет на экран строку `ab|cd`. 10). Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: `«bash командный_файл [аргументы]»`. Чтобы не вводить каждый раз последовательности символов `bash`, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `«chmod +x имя_файла»`. Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществить её интерпретацию. 11). Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды `unset` с 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: 1. \$ –отображается вся командная строка или параметры оболочки; 2. \$? –код завершения последней выполненной команды; 3. \$\$ –уникальный идентификатор процесса, в рамках которого выполняется командный процессор; 4. \$! –номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; 5. \$–значение флагов командного процессора; 6. \${#} –возвращает целое число –количество слов, которые были результатом \$; 7. \${#name} –возвращает целое значение длины строки в переменной name; 8. \${name[n]} –обращение к n-му элементу массива; 9. \${name[\*]} –перечисляет все элементы массива, разделённые пробелом; 10. \${name[@]} –то же самое, но позволяет учитывать символы пробелы в самих переменных; 11. \${name:-value} –если значение переменной name не определено, то оно будет заменено на указанное value; 12. \${name:value} –проверяется факт существования переменной; 13. \${name=value} –если name не определено, то ему присваивается значение value; 14. \${name?value} –останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке; 15. \${name+value} –это выражение работает противоположно \${name-value}. Если переменная определена, то подставляется value; 16. \${name#pattern} –представляет значение переменной name с удалённым самым коротким левым образцом (pattern); 17. \${#name[\*]} и \${#name[@]} –эти выражения возвращают количество элементов в массиве name.