

Отчёт по лабораторной работе №11

**Дисциплина: Операционные системы
Джеффри Родригес Сантос**

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Контрольные вопросы	17
5	Выводы	23

Список таблиц

1 Цель работы

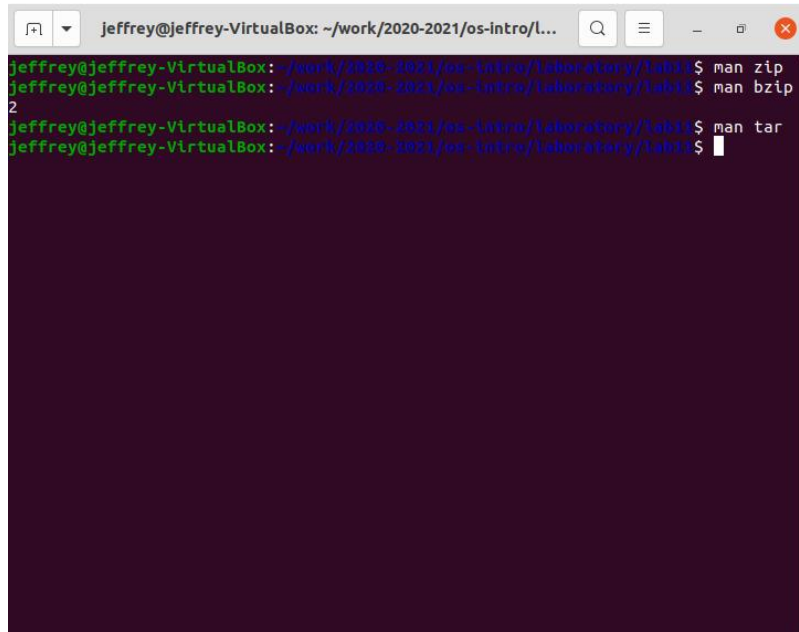
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

2 Задание

1. Написать 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`. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать 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.

3 Выполнение лабораторной работы

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



```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/l...
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ man zip
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ man bzip
2
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ man tar
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$
```

Синтаксис команды zip для архивации файла (рис. -fig. 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]

```
ZIP(1)                                General Commands Manual                                ZIP(1)

NAME
    zip - package and compress (archive) files

SYNOPSIS
    zip [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$] [--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).

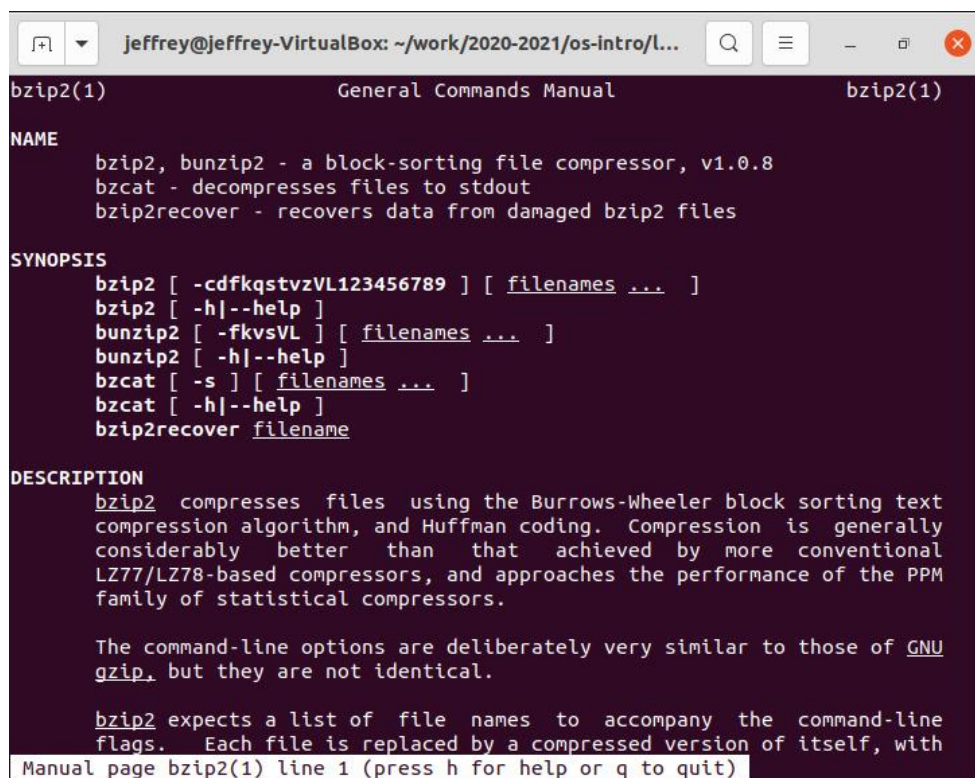
    A companion program (unzip(1)) unpacks zip archives. The zip and un-
    Manual page zip(1) line 1 (press h for help or q to quit)
```

Рис.
3.2:

Синтаксис команды zip

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

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



```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/l...
bzip2(1)                                General Commands Manual                                bzip2(1)

NAME
    bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
    bzip2cat - decompresses files to stdout
    bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
    bzip2 [ -cdfkqstvzVL123456789 ] [ filenames ... ]
    bzip2 [ -h|--help ]
    bunzip2 [ -fkvsVL ] [ filenames ... ]
    bunzip2 [ -h|--help ]
    bzip2cat [ -s ] [ filenames ... ]
    bzip2cat [ -h|--help ]
    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

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

Рис. 3.3: Синтаксис команды bzip2

Синтаксис команды tar для архивации файла (рис. -fig. 3.4): tar [опции]
[архив.tar] [файлы_для_архивации]
Синтаксис команды tar для разархивации/распаковки файла: tar [опции]
[архив.tar].


```
TAR(1) GNU TAR Manual TAR(1)
NAME
tar - an archiving utility

SYNOPSIS
Traditional usage
tar {A|c|d|r|t|u|x}[GnSkUW0mpsMBiajJzZhPLRvwo] [ARG...]

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

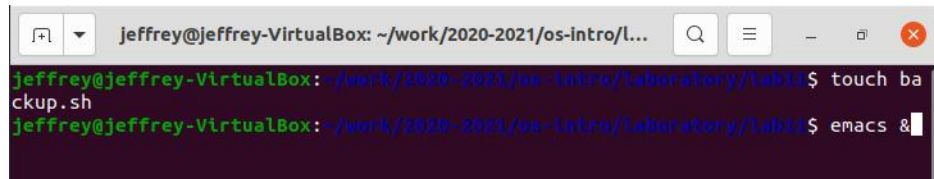
tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
tar -r [-f ARCHIVE] [OPTIONS] [FILE...]
tar -u [-f ARCHIVE] [OPTIONS] [FILE...]
tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

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

tar --create [--file ARCHIVE] [OPTIONS] [FILE...]
Manual page tar(1) line 1 (press h for help or q to quit)
```

Рис. 3.4: Синтаксис команды tar

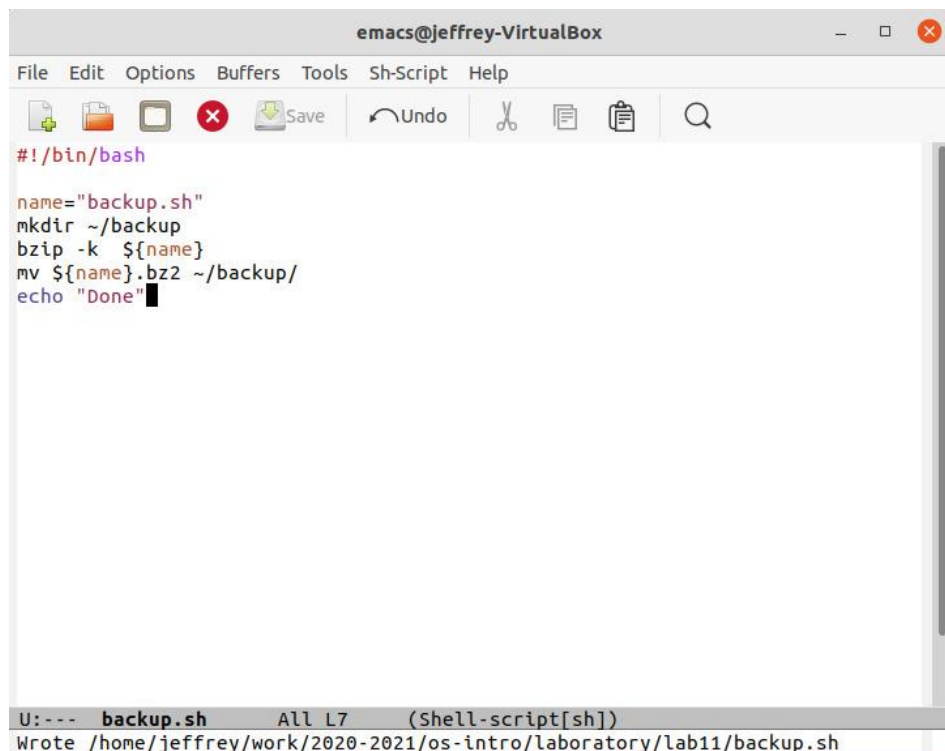
Далее я создал файл, в котором буду писать первый скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (рис. -fig. 3.5).

A terminal window titled 'jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/l...' with standard window controls. The terminal shows two commands: 'touch backup.sh' and 'emacs &'. The first command creates a file named 'backup.sh' in the current directory. The second command opens the file in the Emacs text editor, indicated by the '&' at the end of the command line.

```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/l...$ touch backup.sh
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ emacs &
```

Рис. 3.5: создал файл для первого скрипта

После написал 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 (рис. -fig. 3.6). При написании скрипта использовал архиватор bzip2.



```
#!/bin/bash

name="backup.sh"
mkdir ~/backup
bzip -k ${name}
mv ${name}.bz2 ~/backup/
echo "Done"
```

U:--- backup.sh All L7 (Shell-script[sh])
Wrote /home/jeffrey/work/2020-2021/os-intro/laboratory/lab11/backup.sh

Рис. 3.6: Написал первый скрипт

Проверил работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod +x .sh»)*. Проверил, появился ли каталог backup/, перейдя в него (команда «cd backup/»), посмотрел его содержимое (команда «ls») и просмотрел содержимое архива (команда «bunzip2 -c backup.sh.bz2») (рис. -fig. 3.7) (рис. -fig. 3.8). Скрипт работает корректно.

```

jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ ls -l
total 4
-rwxrwxr-x 1 jeffrey jeffrey 99 мая 29 17:59 backup.sh
-rw-rw-r-- 1 jeffrey jeffrey  0 мая 29 17:56 backup.sh~
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ ls -l
total 4
-rwxrwxr-x 1 jeffrey jeffrey 99 мая 29 17:59 backup.sh
-rw-rw-r-- 1 jeffrey jeffrey  0 мая 29 17:56 backup.sh~
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ ./backup
.sh
mkdir: cannot create directory '/home/jeffrey/backup': File exists
./backup.sh: line 5: bzip: command not found
mv: cannot stat 'backup.sh.bz2': No such file or directory
Done
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$

```

Рис. 3.7: Проверил первый скрипт

1. Создал файл, в котором буду писать второй скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и «emacs &») (рис. -fig. 3.9).

```

jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ touch pr
og2.sh
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab1$ emacs &

```

Рис. 3.9: Создал файл для второго скрипта

Написал 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. -fig. 3.10).

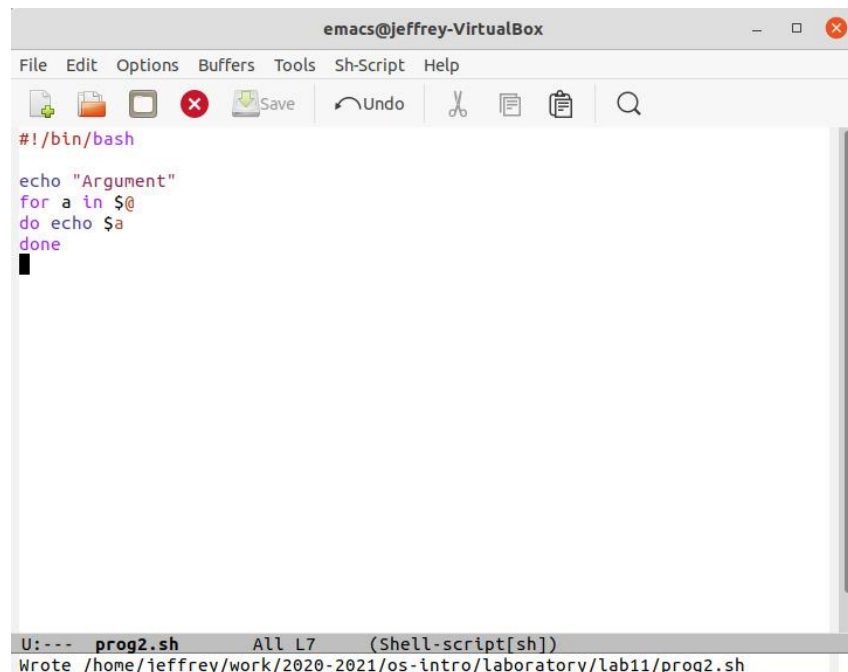


Рис. 3.10: Написал второй скрипт

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

```

jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ chmod +x
*.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ ls -l
total 8
-rwxrwxr-x 1 jeffrey jeffrey 101 мая 29 18:05 backup.sh
-rw-rw-r-- 1 jeffrey jeffrey  0 мая 29 17:56 backup.sh~
-rwxrwxr-x 1 jeffrey jeffrey  57 мая 29 18:10 prog2.sh
-rw-rw-r-- 1 jeffrey jeffrey  0 мая 29 18:09 prog2.sh~
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$

```

```

jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ ./prog2.
sh 1 2 3 4 5
Argument
1
2
3
4
5
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$

```

```

jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ ./prog2.
sh 1 2 3 4 5 6 7 8 9 10 11 12
Argument
1
2
3
4
5
6
7
8
9
10
11
12
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$

```

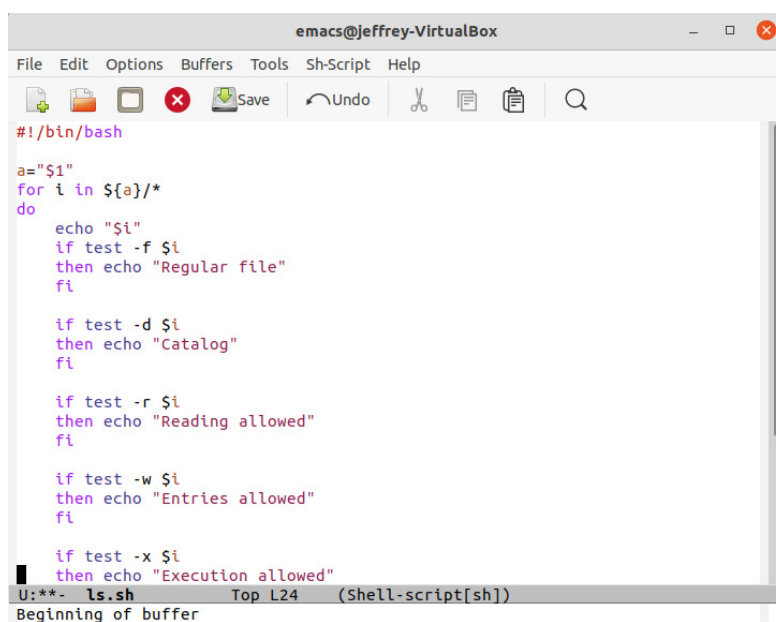
Рис. 3.11: проверил второй скрипт

2. Создал файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch ls.sh» и «emacs &») (рис. -fig. 3.12).

```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab11$ touch ls.sh
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab11$ emacs &
```

Рис. 3.12: Создал файл для третьего скрипта

Написал 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.-fig. 3.13).



The screenshot shows the Emacs editor window titled 'emacs@jeffrey-VirtualBox'. The menu bar includes File, Edit, Options, Buffers, Tools, Sh-Script, and Help. The toolbar contains icons for saving, undo, redo, and search. The main text area contains a shell script for a custom 'ls' command. The script starts with a shebang line '#!/bin/bash' and a function definition 'a="\$1"'. It then uses a 'for' loop to iterate over files in the directory 'a'. For each file 'i', it checks if it is a regular file, a directory, and then checks permissions (read, write, execute) using 'test' and 'echo' statements. The status bar at the bottom shows 'U:**- ls.sh Top L24 (Shell-script[sh])' and 'Beginning of buffer'.

```
#!/bin/bash
a="$1"
for i in ${a}/*
do
    echo "$i"
    if test -f $i
    then echo "Regular file"
    fi

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

    if test -r $i
    then echo "Reading allowed"
    fi

    if test -w $i
    then echo "Entries allowed"
    fi

    if test -x $i
    then echo "Execution allowed"
    fi
done
```

U:**- ls.sh Top L24 (Shell-script[sh])
Beginning of buffer



```
if test -d $i
then echo "Catalog"
fi

if test -r $i
then echo "Reading allowed"
fi

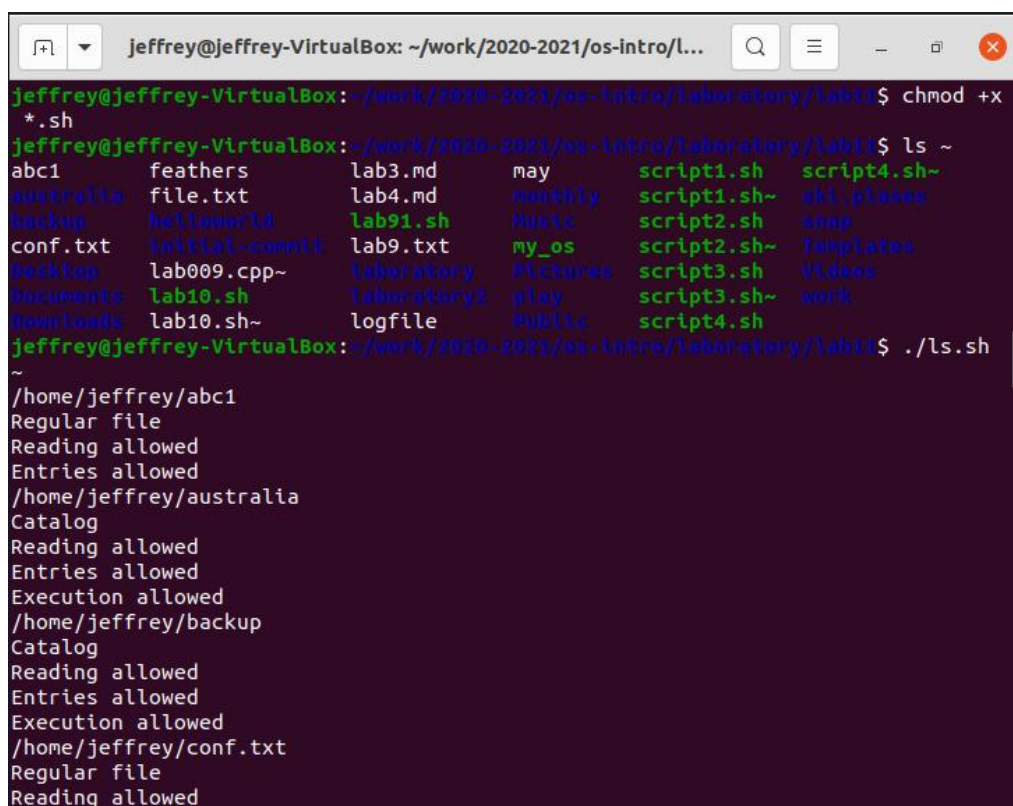
if test -w $i
then echo "Entries allowed"
fi

if test -x $i
then echo "Execution allowed"
fi

done

U:**- ls.sh Bot L26 (Shell-script[sh])
```

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



```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/l...$ chmod +x *.sh
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab11$ ls ~
abc1      feathers      lab3.md      may          script1.sh  script4.sh~
australia file.txt      lab4.md      monthly     script1.sh~ shi.placed
backup    helloworld   lab91.sh     Rustic      script2.sh  shop
conf.txt  initial-commi lab9.txt     my_os       script2.sh~ templates
desktop  lab009.cpp~   laboratory  Pictures    script3.sh  videos
documents lab10.sh      laboratory2  play        script3.sh~ work
downloads lab10.sh~     logfile     Public      script4.sh

jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab11$ ./ls.sh
~/home/jeffrey/abc1
Regular file
Reading allowed
Entries allowed
~/home/jeffrey/australia
Catalog
Reading allowed
Entries allowed
Execution allowed
~/home/jeffrey/backup
Catalog
Reading allowed
Entries allowed
Execution allowed
~/home/jeffrey/conf.txt
Regular file
Reading allowed
```

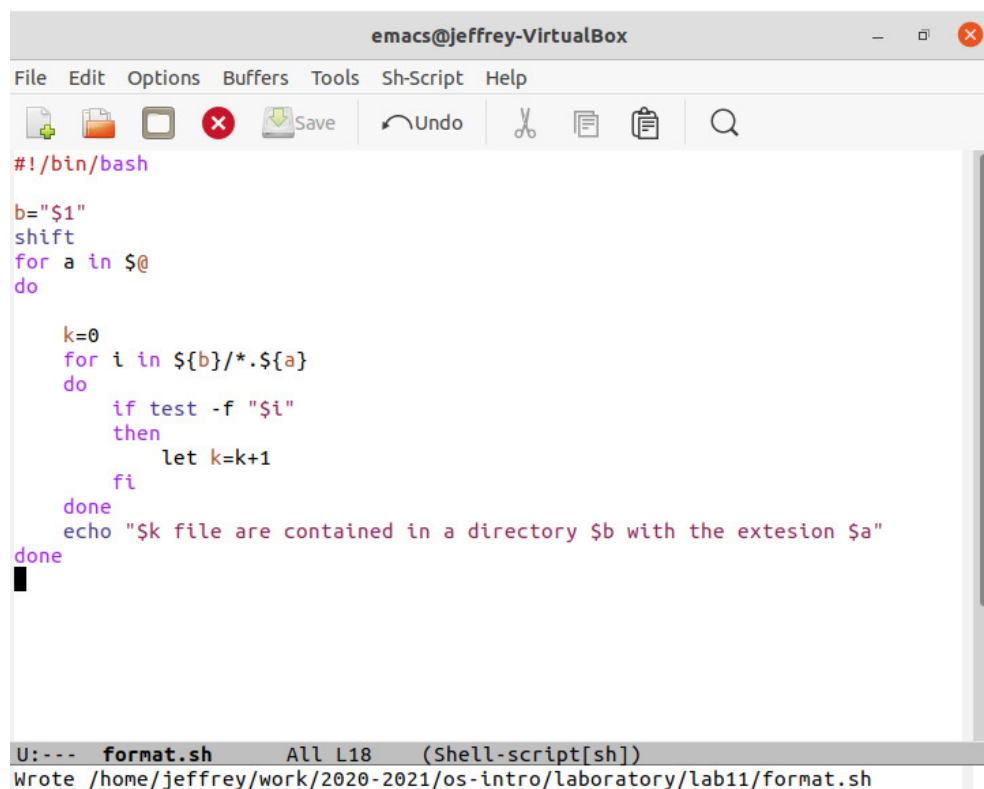

Рис. 3.14: Проверил третий скрипт

3. Для четвертого скрипта также создал файл (команда «touch format.sh») и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (рис. -fig. 3.15).

```
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ touch format.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ emacs &
```

Рис. 3.15: Создал файл для четвёртого скрипта

Написал 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.-fig. 3.16).



```
emacs@jeffrey-VirtualBox
File Edit Options Buffers Tools Sh-Script Help
+ Save Undo Cut Copy Paste Find
#!/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 file are contained in a directory $b with the extesion $a"
done
```

U:--- format.sh All L18 (Shell-script[sh])
Wrote /home/jeffrey/work/2020-2021/os-intro/laboratory/lab11/format.sh

Рис. 3.16: Написал четвёртый скрипт

Проверил 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») (рис.-fig. 3.17). Скрипт работает корректно.

```
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ ./format
.sh ~ pdf doc sh txt
./format.sh: line 17: syntax error: unexpected end of file
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ chmod +x
*.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ ls ~
abc1      feathers  lab3.md   may       script1.sh  script4.sh~
australia file.txt  lab4.md   monthly  script1.sh~ ski_places
backup    helloworld lab91.sh  Music     script2.sh  snap
conf.txt  initial-commit lab9.txt  my_os     script2.sh~ Templates
desktop   lab009.cpp~ laboratory Pictures  script3.sh  Videos
documents lab10.sh  laboratory2 play     script3.sh~ work
downloads lab10.sh~ logfile  Public   script4.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$ ./format
.sh ~ pdf doc sh txt
0 file are contained in a directory /home/jeffrey with the extesion pdf
0 file are contained in a directory /home/jeffrey with the extesion doc
6 file are contained in a directory /home/jeffrey with the extesion sh
3 file are contained in a directory /home/jeffrey with the extesion txt
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab11$
```

Рис. 3.17: Проверил четвёртый скрипт

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:

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

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 `mv afile ${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:

- `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 (new line).
- `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 You have mail (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 метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который, в свою очередь, является метасимволом.

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,
- `echo ab'|'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 осуществит её интерпретацию.

11. Группу команд можно объединить в функцию. Для этого существует ключе-

вое 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

`$` является метасимволом процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов `$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:

- $\$ \square$ – 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 изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.