

Отчёт по лабораторной работе № 11

Перелыгин Сергей Викторович

Содержание

1 Цель работы	4
2 Выполнение лабораторной работы	5
3 Ответы на контрольные вопросы	16
4 Выводы	22
5 Библиография	23

Список иллюстраций

2.1	Изучаем справки команд zip, bzip2, tar	5
2.2	zip	6
2.3	bzip2	7
2.4	tar	8
2.5	Вызов emacs	9
2.6	Скрипт 1	9
2.7	Просмотр содержимого домашнего каталога	10
2.8	Просмотр содержимого backup	10
2.9	Просмотр содержимого архива	10
2.10	Создаем ex2.sh	11
2.11	Скрипт 2	11
2.12	Проверка работы скрипта 2	12
2.13	Создаем ex3.sh	12
2.14	Скрипт 3	13
2.15	Проверка работы скрипта 3	14
2.16	Создаем ex4.sh	14
2.17	Скрипт 4	15
2.18	Проверка работы скрипта 4	15

1 Цель работы

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

2 Выполнение лабораторной работы

1. Для начала я изучил команды архивации, используя команды «man zip», «man bzip2», «man tar»(Рисунки 1-4).

Синтаксис команды zip для архивации файла: 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]

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

Синтаксис команды tar для архивации файла:tar[опции][архив.tar][файлы _для _архивации].
Синтаксис команды tar для разархивации/распаковки файла: tar[опции][архив.tar]

```
sergperelihgin@sergperelihgin-VirtualBox:~$ man zip  
sergperelihgin@sergperelihgin-VirtualBox:~$ man bzip2  
sergperelihgin@sergperelihgin-VirtualBox:~$  
sergperelihgin@sergperelihgin-VirtualBox:~$ man tar
```

Рис. 2.1: Изучаем справки команд zip, bzip2, tar

```
sergperelihgin@sergperelihgin-VirtualBox: ~
```

ZIP(1) General Commands Manual **ZIP(1)**

NAME
zip - package and compress (archive) files

SYNOPSIS

```
zip [-aABCdDeEffFghjkLmoqrRSTuvVwXyz!@$] [--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, MS-DOS, 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 **unzip(1)** 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**.

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

Рис. 2.2: zip

```
sergperelihgın@sergperelihgın-VirtualBox: ~          Q  -  X
bzip2(1)           General Commands Manual          bzip2(1)

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

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

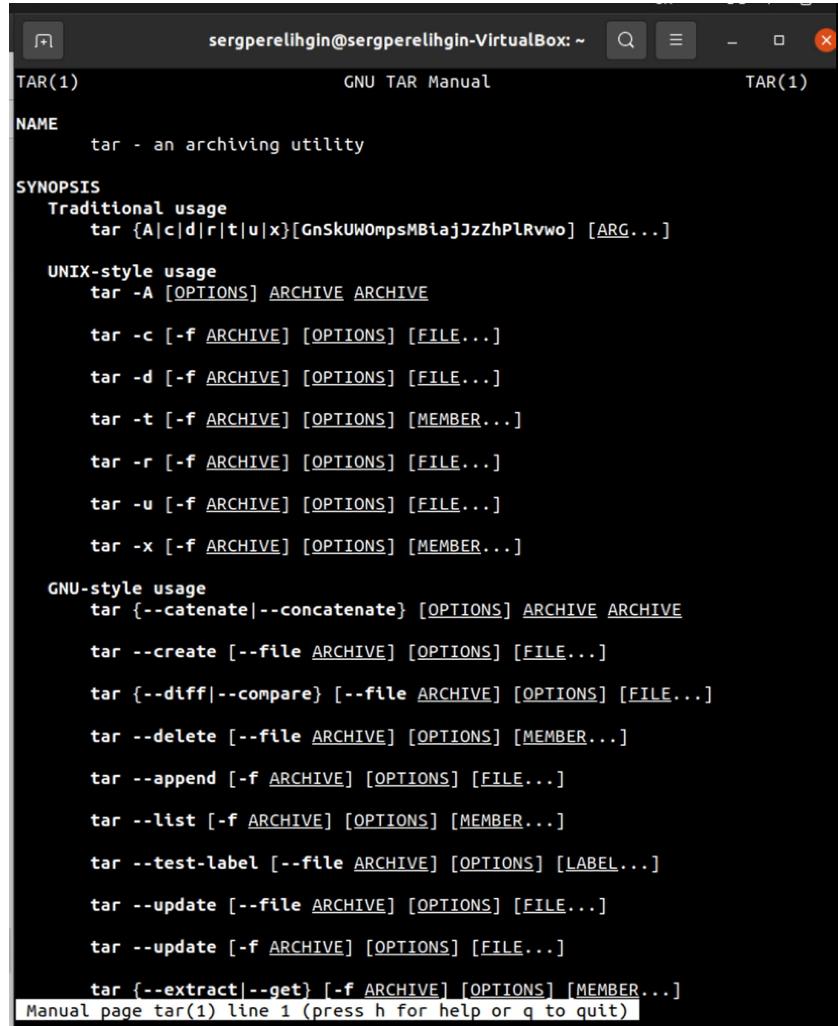
bzip2 expects a list of file names to accompany the command-line
flags. Each file is replaced by a compressed version of itself,
with the name "original_name.bz2". Each compressed file has the
same modification date, permissions, and, when possible, ownership
as the corresponding original, so that these 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 com-
pressed output to a terminal, as this would be entirely incomprehen-
sible and therefore pointless.

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

Рис. 2.3: bzip2



The screenshot shows a terminal window with the title "sergperelihgin@sergperelihgin-VirtualBox: ~". The window displays the "GNU TAR Manual" page for the "tar" command. The page is organized into sections: NAME, SYNOPSIS, Traditional usage, UNIX-style usage, and GNU-style usage. The "NAME" section contains the command name "tar - an archiving utility". The "SYNOPSIS" section is divided into three parts: Traditional usage, UNIX-style usage, and GNU-style usage, each listing various options and arguments. The "Traditional usage" part includes commands like tar {A|c|d|r|t|u|x}[GnSkUW0mpsMBiajJzzhPlRvwo] [ARG...]. The "UNIX-style usage" part includes 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...], and tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]. The "GNU-style usage" part includes tar {--catenate|--concatenate} [OPTIONS] ARCHIVE ARCHIVE, tar --create [--file ARCHIVE] [OPTIONS] [FILE...], tar {--diff|--compare} [--file ARCHIVE] [OPTIONS] [FILE...], tar --delete [--file ARCHIVE] [OPTIONS] [MEMBER...], tar --append [-f ARCHIVE] [OPTIONS] [FILE...], tar --list [-f ARCHIVE] [OPTIONS] [MEMBER...], tar --test-label [--file ARCHIVE] [OPTIONS] [LABEL...], tar --update [--file ARCHIVE] [OPTIONS] [FILE...], tar --update [-f ARCHIVE] [OPTIONS] [FILE...], and tar {--extract|--get} [-f ARCHIVE] [OPTIONS] [MEMBER...]. At the bottom of the page, it says "Manual page tar(1) line 1 (press h for help or q to quit)".

Рис. 2.4: 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 &») (Рисунок 5).

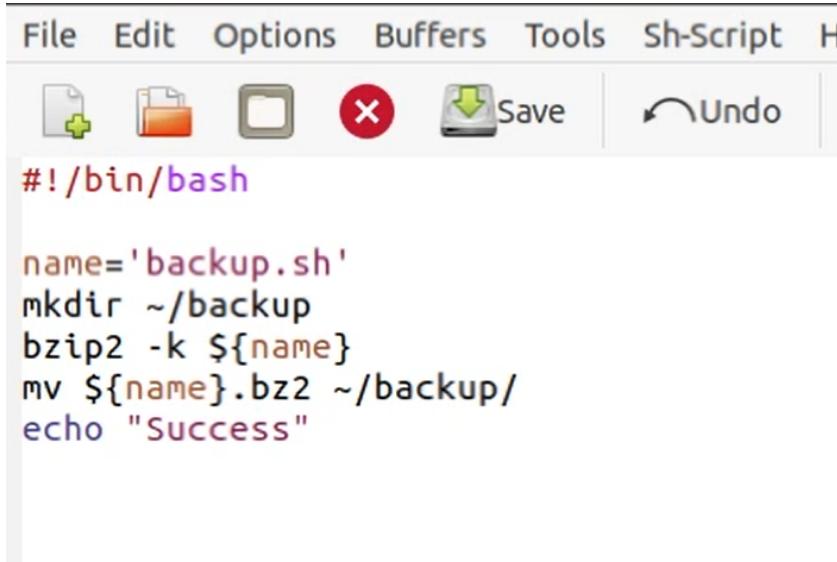
```

sergperelihgin@sergperelihgin-VirtualBox:~$ ls
abc1                               file3.txt      Pictures
academic-laboratory-report-template   file4.txt      play
academic-presentation-markdown-template file.txt      Public
australia                            '#lab07.sh#'  reports
conf.txt                             lab07.sh     ski.plases
Desktop                             lab2          snap
Documents                           may           Templates
Downloads                           monthly       test
equipment                           morefunnew  Videos
feathers                            Music         work
file1.txt                           my_os        маба10
file2.txt                           pandoc-crossref 'aa6a9 (1)'
sergperelihgin@sergperelihgin-VirtualBox:~$ touch backup.sh
sergperelihgin@sergperelihgin-VirtualBox:~$ emacs &

```

Рис. 2.5: Вызов emacs

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 вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar (Рисунок 6). При написании скрипта использовал архиватор bzip2.



```

File Edit Options Buffers Tools Sh-Script H
[Icons for New, Open, Save, Undo]
#!/bin/bash

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

```

Рис. 2.6: Скрипт 1

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/»), посмотрела его содержимое (ко-

манда «ls») и просмотрел содержимое архива (команда «bunzip2 -c backup.sh.bz2») (Рисунки 7, 8, 9). Скрипт работает корректно.

```
sergperelihgın@sergperelihgın-VirtualBox:~$ ./backup.sh
Success
sergperelihgın@sergperelihgın-VirtualBox:~$ chmod +x *.sh
sergperelihgın@sergperelihgın-VirtualBox:~$ ./backup.sh
mkdir: cannot create directory '/home/sergperelihgın/backup': File exists
Success
sergperelihgın@sergperelihgın-VirtualBox:~$ ls
abc1                               file1.txt      pandoc-crossref
academic-laboratory-report-template   file2.txt      Pictures
academic-presentation-markdown-template   file3.txt      play
australia                            file4.txt      Public
backup                               file.txt       reports
backup.sh                            '#lab07.sh#'  skl.plases
backup.sh~                           lab07.sh     snap
conf.txt                            lab2          Templates
Desktop                             may           test
Documents                           monthly        Videos
Downloads                           morefunnew    work
equipment                           Music          магазин
feathers                            my_os         'магазин (1)'
sergperelihgın@sergperelihgın-VirtualBox:~$
```

Рис. 2.7: Просмотр содержимого домашнего каталога

```
sergperelihgın@sergperelihgın-VirtualBox:~/backup$ ls
backup.sh.bz2
sergperelihgın@sergperelihgın-VirtualBox:~/backup$
```

Рис. 2.8: Просмотр содержимого backup

```
sergperelihgın@sergperelihgın-VirtualBox:~/backup$ bunzip2 -c backup.sh.bz2
#!/bin/bash
=====
name='backup.sh'
mkdir ~/backup
bztp2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Success"
sergperelihgın@sergperelihgın-VirtualBox:~/backup$
```

Рис. 2.9: Просмотр содержимого архива

2. Создал файл, в котором буду писать второй скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch ex2.sh» и «emacs &») (Рисунок 10).

```
sergperelihgin@sergperelihgin-VirtualBox:~$ touch ex2.sh  
sergperelihgin@sergperelihgin-VirtualBox:~$ emacs &
```

Рис. 2.10: Создаем ex2.sh

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 (Рисунок 11).

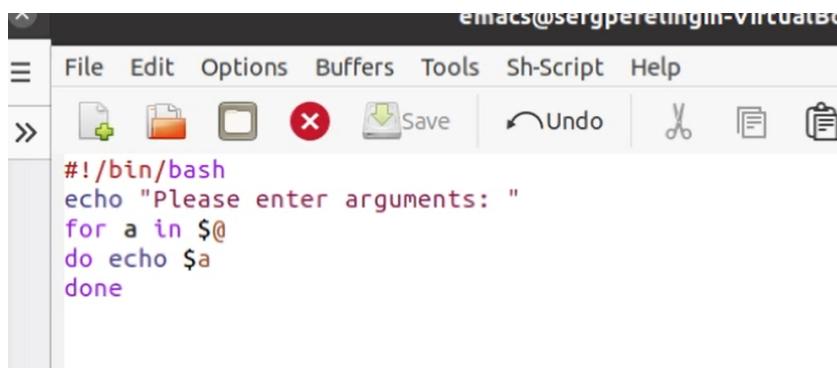


Рис. 2.11: Скрипт 2

Проверил работу написанного скрипта, предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Вводил аргументы, количество которых меньше 10 и больше 10 (Рисунок 12). Скрипт работает корректно.

```
sergperelihgin@sergperelihgin-VirtualBox:~$ chmod +x *.sh
sergperelihgin@sergperelihgin-VirtualBox:~$ ./ex2.sh 1 2 3
Please enter arguments:
a
a
a
sergperelihgin@sergperelihgin-VirtualBox:~$ emacs
sergperelihgin@sergperelihgin-VirtualBox:~$ ./ex2.sh 1 2 3
Please enter arguments:
1
2
3
sergperelihgin@sergperelihgin-VirtualBox:~$ ./ex2.sh 1 2 3 4 5 6 7 8 9 10
11 12 13 14
Please enter arguments:
1
2
3
4
5
6
7
8
9
10
11
12
13
14
sergperelihgin@sergperelihgin-VirtualBox:~$
```

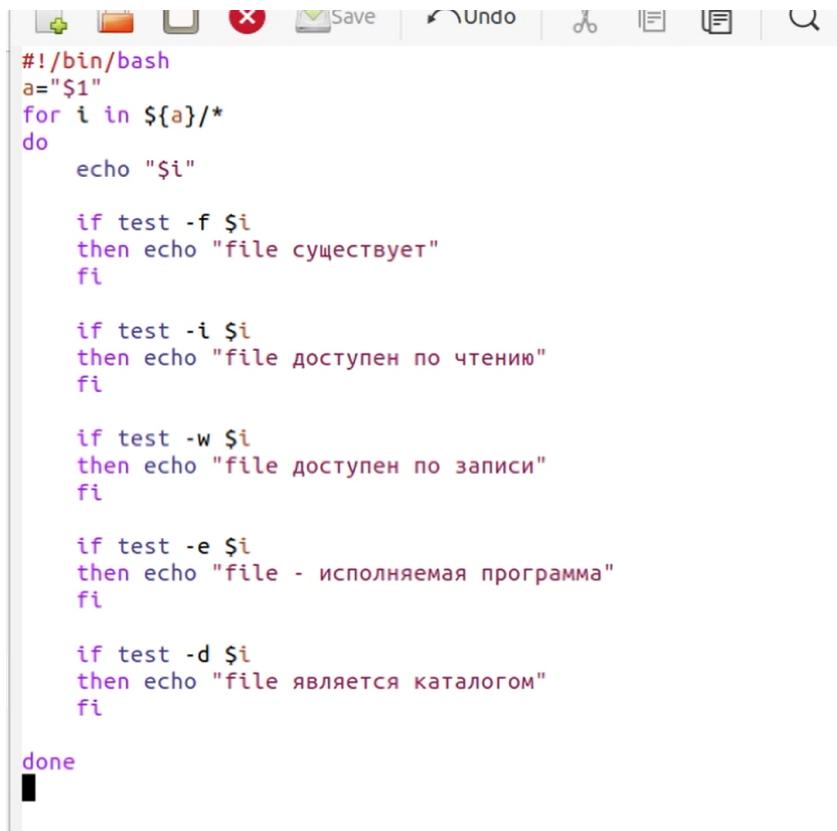
Рис. 2.12: Проверка работы скрипта 2

- Создал файл, в котором буду писать третий скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch ex3.sh» и «emacs &») (Рисунок 13).

```
13
14
sergperelihgin@sergperelihgin-VirtualBox:~$ touch ex3.sh
sergperelihgin@sergperelihgin-VirtualBox:~$ emacs
□
```

Рис. 2.13: Создаем ex3.sh

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 (Рисунок 14).



```
#!/bin/bash
a="$1"
for i in ${a}/*
do
    echo "$i"

    if test -f $i
    then echo "file существует"
    fi

    if test -i $i
    then echo "file доступен по чтению"
    fi

    if test -w $i
    then echo "file доступен по записи"
    fi

    if test -e $i
    then echo "file - исполняемая программа"
    fi

    if test -d $i
    then echo "file является каталогом"
    fi

done
```

Рис. 2.14: Скрипт 3

Далее проверил работу скрипта (команда «./ex3.sh ~»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh») (Рисунок 15). Скрипт работает корректно.

```
sergperelihgin@sergperelihgin-VirtualBox:~$ ./ex3.sh ~
/home/sergperelihgin/abc1
file существует
./ex3.sh: line 11: test: -i: unary operator expected
file доступен по записи
file - исполняемая программа
/home/sergperelihgin/academic-laboratory-report-template
./ex3.sh: line 11: test: -i: unary operator expected
file доступен по записи
file - исполняемая программа
file является каталогом
/home/sergperelihgin/academic-presentation-markdown-template
./ex3.sh: line 11: test: -i: unary operator expected
file доступен по записи
file - исполняемая программа
file является каталогом
/home/sergperelihgin/australia
./ex3.sh: line 11: test: -i: unary operator expected
file доступен по записи
file - исполняемая программа
file является каталогом
/home/sergperelihgin/backup
./ex3.sh: line 11: test: -i: unary operator expected
file доступен по записи
file - исполняемая программа
file является каталогом
/home/sergperelihgin/backup.sh
file существует
./ex3.sh: line 11: test: -i: unary operator expected
file доступен по записи
file - исполняемая программа
/home/sergperelihgin/backup.sh~
file существует
```

Рис. 2.15: Проверка работы скрипта 3

4. Для четвертого скрипта также создал файл (команда «touch ex4.sh») и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (Рисунок 16).

```
sergperelihgin@sergperelihgin-VirtualBox:~$ touch ex4.sh
sergperelihgin@sergperelihgin-VirtualBox:~$ emacs
```

Рис. 2.16: Создаем ex4.sh

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(Рисунок 17).

```

#!/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 files in the catalog $b with the file extension $a"
done

```

Рис. 2.17: Скрипт 4

Проверил работу написанного скрипта, предварительно добавив для него право на выполнение (команда «chmod +x *.sh»), а также создав дополнительные файлы с разными расширениями (Рисунок 18). Скрипт работает корректно.

```

sergperelihgin@sergperelihgin-VirtualBox:~$ chmod +x *.sh
[1]+ Done                  emacs
sergperelihgin@sergperelihgin-VirtualBox:~$ ./ex4.sh ~ sh
5 files in the catalog /home/sergperelihgin with the file extension sh
sergperelihgin@sergperelihgin-VirtualBox:~$ ./ex4.sh ~ sh pdf doc docx jpg txt
5 files in the catalog /home/sergperelihgin with the file extension sh
1 files in the catalog /home/sergperelihgin with the file extension pdf
1 files in the catalog /home/sergperelihgin with the file extension doc
0 files in the catalog /home/sergperelihgin with the file extension docx
0 files in the catalog /home/sergperelihgin with the file extension jpg
7 files in the catalog /home/sergperelihgin with the file extension txt
sergperelihgin@sergperelihgin-VirtualBox:~$ 

```

Рис. 2.18: Проверка работы скрипта 4

3 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 (Bourneshell или sh) - стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций.
 - С-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 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 `$`. 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 monday trash`». В переменные `mon` и `day` будут считаны соответствующие значения, введённые с клавиатуры, а переменная `trash` нужна для того, чтобы отбрасывать 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 (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 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. Стока, 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 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:

- \$* - 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.

4 Выводы

В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux, а также научился писать небольшие командные файлы.

5 Библиография

- Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: <http://lib.rudn.ru/MegaPro2/Download/MObject/6118>.
- Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немлюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ET 60)
- Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ET 50)
- Ван Стеен М., Эндрю Таненбаум Распределенные системы. Принципы и парадигмы [Текст] / Э. Таненбаум, в.М. Стеен. — СПб. : Питер, 2003. — 877 с. : ил. — (Классика Computer science). — ISBN 5-272-00053-6 : 377.52. (ET 50)
- Сафонов, В.О. Основы современных операционных систем : учебное пособие / В.О. Сафонов. — Москва : Интернет-Университет Информационных Технологий, 2011. — 584 с. — (Основы информационных технологий). — ISBN 978-5-9963-0495-0 ; То же [Электронный ресурс]. — URL: <http://biblioclub.ru/index.php?page=book&id=233210>.
- Немет Эви. UNIX — руководство системного администратора [Текст] / Э. Немет, Г. Снайдер, С. Сибасс; Э.Немет, Г.Снайдер, С.Сибасс, Х.Р.Трент. — 3-е изд. — СПб. : Питер, 2004. — 925 с. : ил. — (Для профессионалов). — ISBN

0-13-020601-6. — ISBN 5-318-00754-6 : 280.00. (ET 30)

- Бек Л. Введение в системное программирование [Текст] / Л. Бек; Пер. с англ. Н.А.Богомолова, В.М.Вязовского и С.Е.Морковина; Под ред. Л.Н.Королева. — М. : Мир, 1988. — 448 с. : ил. — ISBN 5-03-000011-9 : 2.60. (ET 3)
- Дьяконов Владимир Юрьевич. Системное программирование [Текст] : Учебное пособие для втузов / В.Ю. Дьяконов, В.А. Китов, И.А. Калинчев; Под ред. А.Л.Горелика. — М. : Высшая школа, 1990. — 221 с. : ил. — ISBN 5-06-000732-4 : 0.55.