

Отчёт по лабораторной работе № 10

дисциплина: Операционные системы

Андрианова Марина Георгиевна

Цель работы

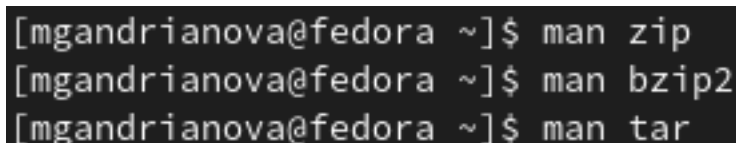
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

Задание

1. Сделать отчёт по лабораторной работе № 10 в формате Markdown.
2. Изучить основы программирования в оболочке ОС UNIX/Linux.

Выполнение лабораторной работы

1). Сначала я изучила команды архивации, используя команды “man zip”, “man bzip2”, “man tar” (рис.1, рис.2, рис.3, рис.4).



```
[mgandrianova@fedora ~]$ man zip  
[mgandrianova@fedora ~]$ man bzip2  
[mgandrianova@fedora ~]$ man tar
```

Рис.1: Работа с консолью

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]

```
mgandrianova@fedora:~ — man zip
ZIP(1L)
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(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 4.0), 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:Информация о zip

Синтаксис команды bzip2 для архивации файла:

bzip2 [опции] [имена файлов]

Синтаксис команды bzip2 для разархивации/распаковки файла:

bunzip2 [опции] [архивы.bz2]

```
mgandrianova@fedora:~ — man bzip2
bzip2(1)
General Commands Manual
bzip2(1)

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

SYNOPSIS
bzip2 [ -cdfkqstzVL123456789 ] [ filenames ... ]
bunzip2 [ -fkvsVL ] [ filenames ... ]
bzcat [ -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

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

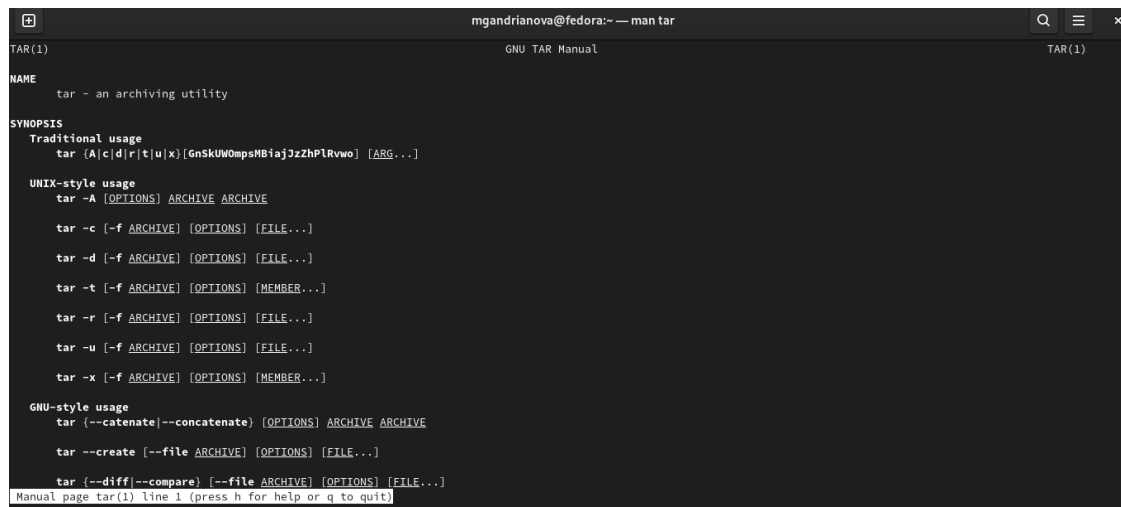
Рис.3:Информация о bzip2

Синтаксис команды tar для архивации файла:

tar [опции] [архив.tar][файлы_для_архивации]

Синтаксис команды bzip2 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}[GnSkUWompsMBiajJzZhPlRvwo] [ARG...]

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

tar -c [-f ARCHIVE] [OPTIONS] [FILE...]

tar -d [-f ARCHIVE] [OPTIONS] [FILE...]

tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]

tar -r [-f ARCHIVE] [OPTIONS] [FILE...]

tar -u [-f ARCHIVE] [OPTIONS] [FILE...]

tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

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

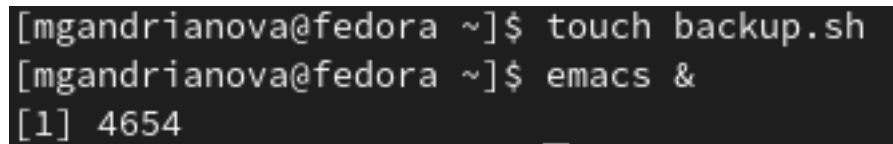
tar --create [--file ARCHIVE] [OPTIONS] [FILE...]

tar [--diff|--compare] [--file ARCHIVE] [OPTIONS] [FILE...]

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

Рис.4:Информация о tar

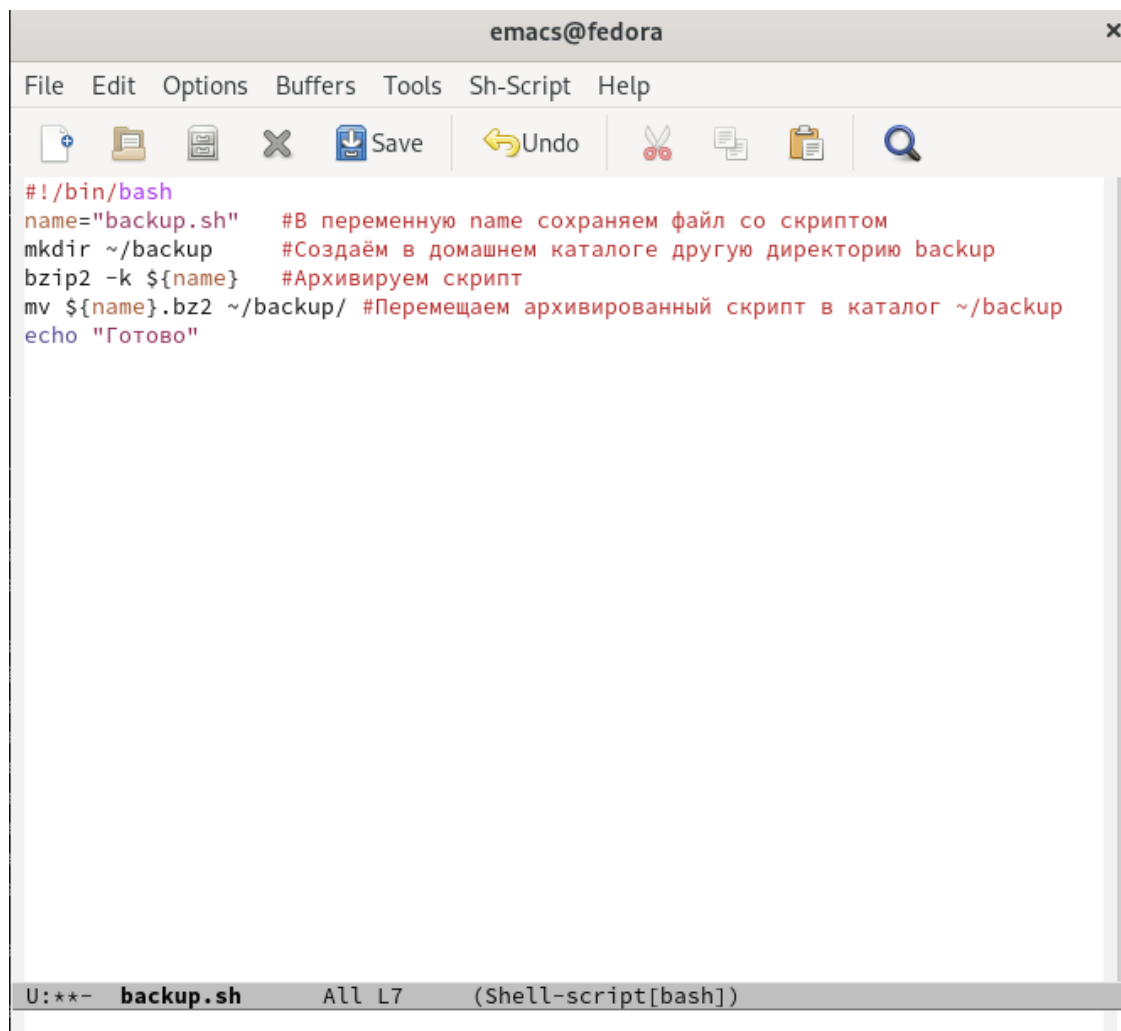
Создала файл backup.sh(команда “touch backup.sh”), в котором буду писать 1 скрипт, открыла редактор Emacs фоновом режиме(команда “emacs &”). Затем открыла созданный файл в редакторе, используя клавиши “Ctrl-x” и “Ctrl-f”(рис.5)



```
[mgandrianova@fedora ~]$ touch backup.sh
[mgandrianova@fedora ~]$ emacs &
[1] 4654
```

Рис.5:Создание файла backup.sh

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 нашем домашнем каталоге(рис.6). При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Я использовала архиватор bzip2.



```
emacs@fedora
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
name="backup.sh" #В переменную name сохраняем файл со скриптом
mkdir ~/backup #Создаём в домашнем каталоге другую директорию backup
bzip2 -k ${name} #Архивируем скрипт
mv ${name}.bz2 ~/backup/ #Перемещаем архивированный скрипт в каталог ~/backup
echo "Готово"
U:***- backup.sh All L7 (Shell-script[bash])
```

Рис.6:Скрипт № 1

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

```

[mgandrianova@fedora ~]$ chmod +x *.sh
[1]+  Завершён      emacs
[mgandrianova@fedora ~]$ ./backup.sh
Готово
[mgandrianova@fedora ~]$ cd backup/
[mgandrianova@fedora backup]$ ls
backup.sh.bz2
[mgandrianova@fedora backup]$ bunzip2 -c backup.sh.bz2
#!/bin/bash
name="backup.sh"    #В переменную name сохраняем файл со скриптом
mkdir ~/backup      #Создаём в домашнем каталоге другую директорию backup
bzip2 -k ${name}     #Архивируем скрипт
mv ${name}.bz2 ~/backup/ #Перемещаем архивированный скрипт в каталог ~/backup
echo "Готово"

```

Рис.7:Проверка работы скрипта

2). Создала файл teamfile.sh(команда “touch teamfile.sh”), в котором буду писать 2 скрипт, и открыла его в редакторе emacs, используя клавиши “Ctrl-x” и “Ctrl-f”(рис.8).

```

[mgandrianova@fedora backup]$ cd
[mgandrianova@fedora ~]$ touch teamfile.sh
[mgandrianova@fedora ~]$ emacs &

```

Рис.8:Создание файла

Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов(рис.9).

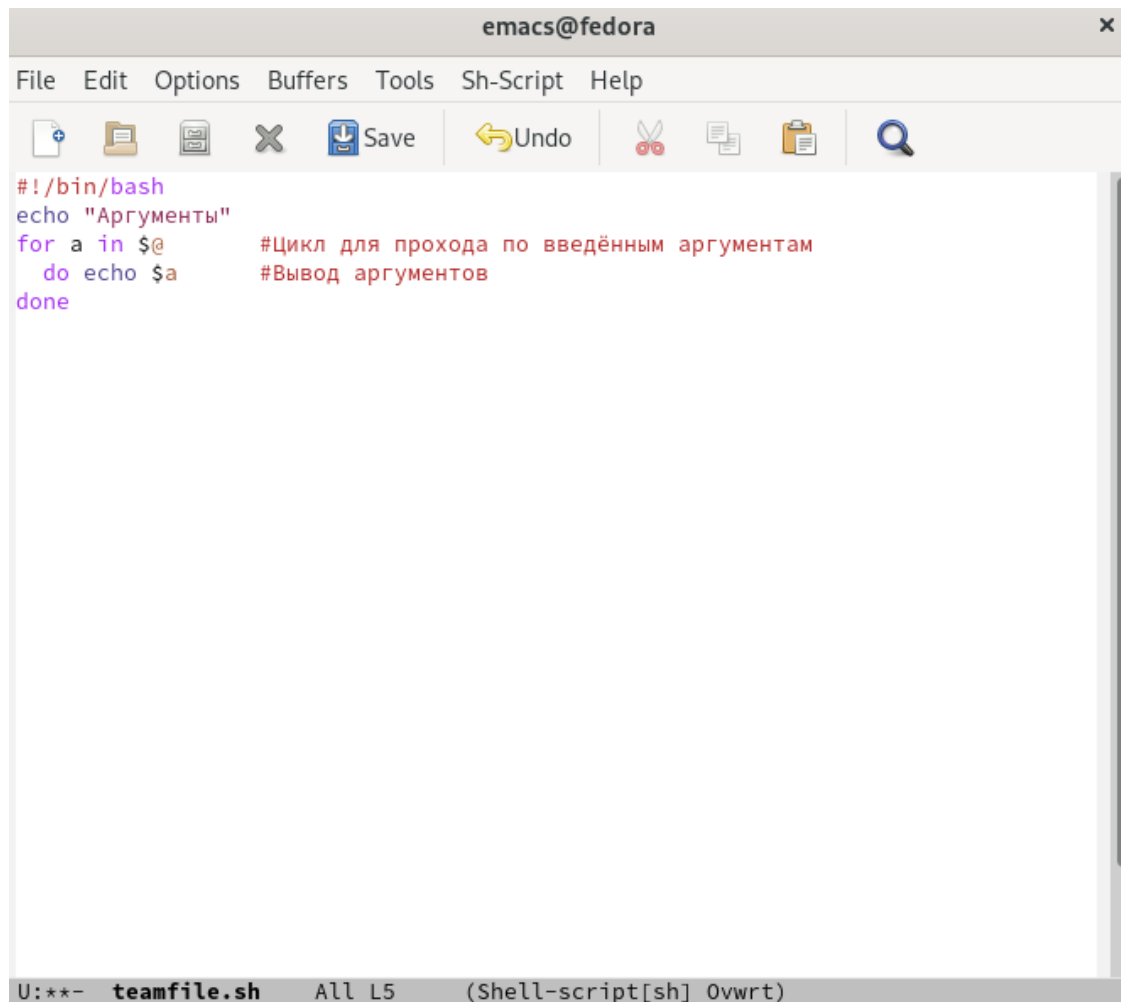


Рис.9:Скринш № 2

Проверила работу 2 скрипта (команды `./teamfile.sh 0 1 2 3 4 5` и `./teamfile.sh 0 1 2 3 4 5 6 7 8 9 10 11 12`), до этого добавив для него право на выполнение (команда `chmod +x *.sh`). Вводила аргументы, количество которых было меньше 10 и больше 10 (рис.10,рис.11). Скрипт работает корректно.

```
[mgandrianova@fedora ~]$ chmod +x *.sh
[2]+  Завершён      emacs
[mgandrianova@fedora ~]$ ./teamfile.sh 0 1 2 3 4 5
Аргументы
0
1
2
3
4
5
```

Рис.10:Проверка работы скрипта(кол-во аргументов меньше 10)

```
[mgandrianova@fedora ~]$ ./teamfile.sh 0 1 2 3 4 5 6 7 8 9 10 11 12
Аргументы
0
1
2
3
4
5
6
7
8
9
10
11
12
```

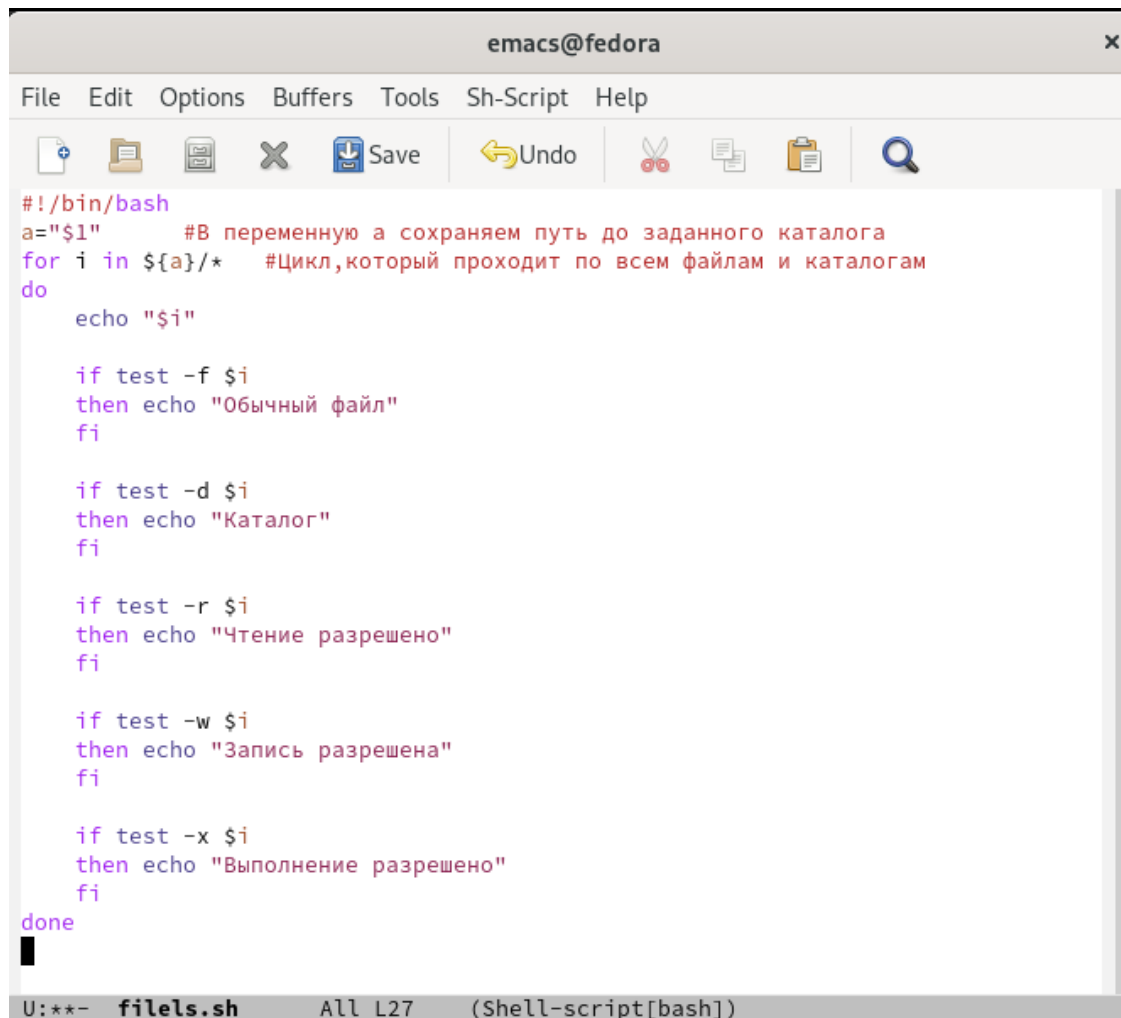
Рис.11:Проверка работы скрипта(кол-во аргументов больше 10)

3). Создала файл filels.sh(команда “touch filels.sh”), в котором буду писать 3 скрипт, и открыла его в редакторе emacs, используя клавиши “Ctrl-x” и “Ctrl-f”(рис.12).

```
[mgandrianova@fedora ~]$ touch filels.sh
[mgandrianova@fedora ~]$ emacs &
```

Рис.12:Создание файла

Написала командный файл — аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога(рис.13).



```
#!/bin/bash
a="$1" #В переменную а сохраняем путь до заданного каталога
for i in ${a}/* #Цикл, который проходит по всем файлам и каталогам
do
    echo "$i"

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

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

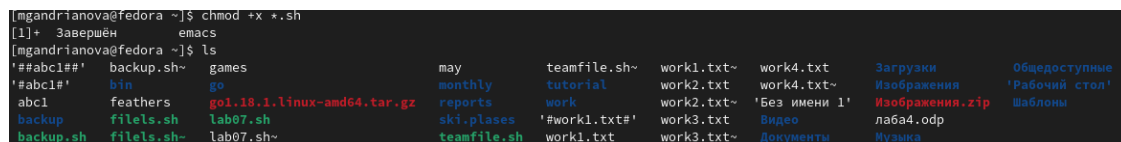
    if test -r $i
    then echo "Чтение разрешено"
    fi

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

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

Рис.13:Скринт № 3

Проверила работу 3 скрипта (команда “./filels.sh ~”), до этого добавив для него право на выполнение (команда “chmod +x *.sh”).(рис.14,рис.15,рис.16). Скрипт работает корректно.



```
[mgandrianova@fedora ~]$ chmod +x *.sh
[1]+  Завершён      emacs
[mgandrianova@fedora ~]$ ls
'##abc1#*'  backup.sh~  games      may      teamfile.sh~  work1.txt~  work4.txt~  Загрузки  Общедоступные
'##abc1#*'  bin         go         monthly  tutorial      work2.txt~  work4.txt~  Изображения  'Рабочий стол'
abc1       feathers    gol.18.1.linux-amd64.tar.gz  reports   work        work2.txt~  'Без имени 1'  Изображения.zip  Шаблоны
backup     filels.sh~  lab07.sh~  ski.plases  '#work1.txt#'  work3.txt~  Видео        лаба4.odp
backup.sh  filels.sh~  lab07.sh~  teamfile.sh  work1.txt      work3.txt~  Документы    Музыка
```

Рис.14:Проверка работы скрипта

```
[mgandrianova@fedora ~]$ ./filels.sh ~  
/home/mgandrianova/##abc1##  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/mgandrianova/#abc1#  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/mgandrianova/abc1  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/mgandrianova/backup  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mgandrianova/backup.sh  
Обычный файл  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mgandrianova/backup.sh~  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/mgandrianova/bin  
Каталог  
Чтение разрешено  
Запись разрешена
```

Рис.15:Проверка работы скрипта

```

/home/mgandrianova/Изображения.zip
Обычный файл
Чтение разрешено
Запись разрешена
/home/mgandrianova/лаба4.odp
Обычный файл
Чтение разрешено
Запись разрешена
/home/mgandrianova/Музыка
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/mgandrianova/Общедоступные
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/mgandrianova/Рабочий стол
./filels.sh: строка 7: test: /home/mgandrianova/Рабочий: ожидается бинарный оператор
./filels.sh: строка 11: test: /home/mgandrianova/Рабочий: ожидается бинарный оператор
./filels.sh: строка 15: test: /home/mgandrianova/Рабочий: ожидается бинарный оператор
./filels.sh: строка 19: test: /home/mgandrianova/Рабочий: ожидается бинарный оператор
./filels.sh: строка 23: test: /home/mgandrianova/Рабочий: ожидается бинарный оператор
/home/mgandrianova/Шаблоны
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено

```

Рис.16:Проверка работы скрипта

4). Создала файл findformat.sh(команда “touch findformat.sh”), в котором буду писать 4 скрипт, и открыла его в редакторе emacs, используя клавиши “Ctrl-x” и “Ctrl-f”(рис.17).

```

[mgandrianova@fedora ~]$ touch findformat.sh
[mgandrianova@fedora ~]$ emacs &

```

Рис.17:Создание файла

Написала командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки(рис.18).

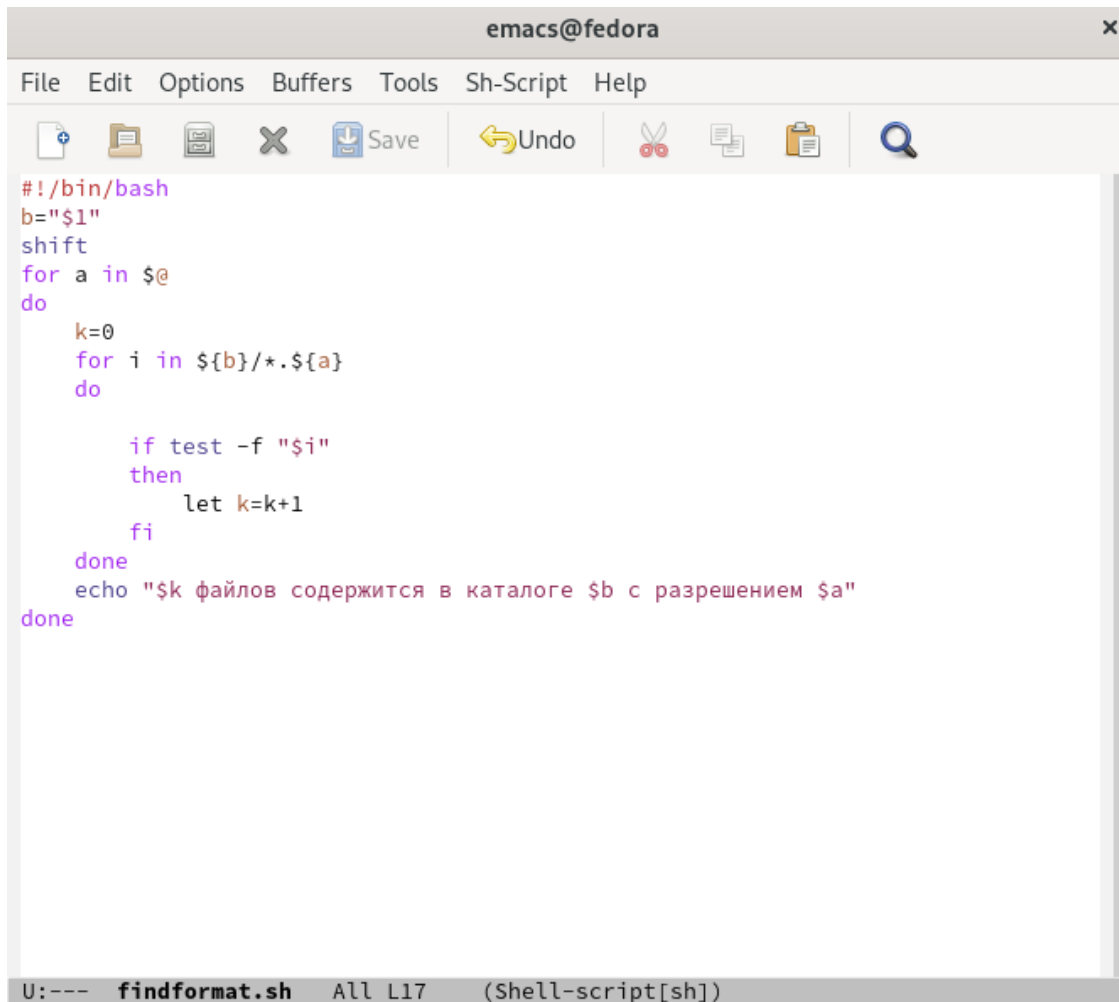


Рис.18:Скринш № 4

Проверила работу 4 скрипта (команда “./findformat.sh ~ txt doc jpg pdf sh”), до этого добавив для него право на выполнение (команда “chmod +x *.sh”).(рис.19). Скрипт работает корректно.

```

[mgandrianova@fedora ~]$ chmod +x *.sh
[1]+  Завершён          emacs
[mgandrianova@fedora ~]$ ./findformat.sh ~ txt doc jpg pdf sh
4 файлов содержится в каталоге /home/mgandrianova с разрешением txt
0 файлов содержится в каталоге /home/mgandrianova с разрешением doc
0 файлов содержится в каталоге /home/mgandrianova с разрешением jpg
0 файлов содержится в каталоге /home/mgandrianova с разрешением pdf
5 файлов содержится в каталоге /home/mgandrianova с разрешением sh

```

Рис.19:Проверка работы скрипта

Выводы

Я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.

Контрольные вопросы

1). Командный процессор (командная оболочка, интерпретатор команд shell) – это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

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

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

3). Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда «mark=/usr/andy/bin» присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол «\$». Например команда «

{mark}» переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin. Оболочка bash позволяет работать с массивами. Для создания массива используется команда set 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, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать её.

5). В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).

6). В (()) можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.

7). Стандартные переменные:

PATH: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной **PATH**, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.

PS1 и **PS2:** эти переменные предназначены для отображения промптера командного процессора. **PS1** – это промптер командного процессора, по умолчанию его значение равно символу \$ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер **PS2**. Он по умолчанию имеет значение символа >.

HOME: имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.

IFS: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (newline).

MAIL: командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение `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 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 unsetсфлагом -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.