

Отчёт по лабораторной работе №10

Операционные системы

Самсонова Мария Ильинична

Содержание

Цель работы	1
Задание	1
Выполнение лабораторной работы	2
Вывод.....	13
Ответы на контрольные вопросы	13

Цель работы

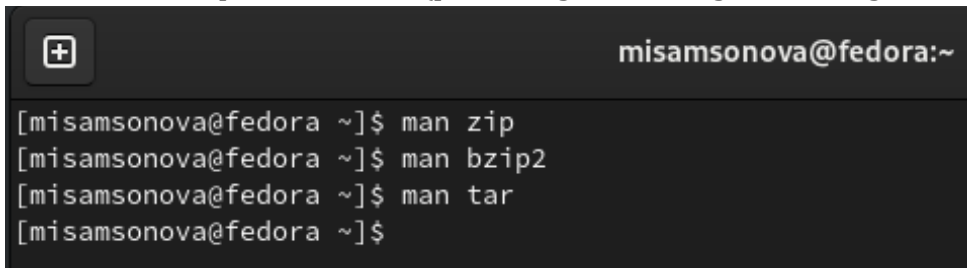
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

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.

Выполнение лабораторной работы

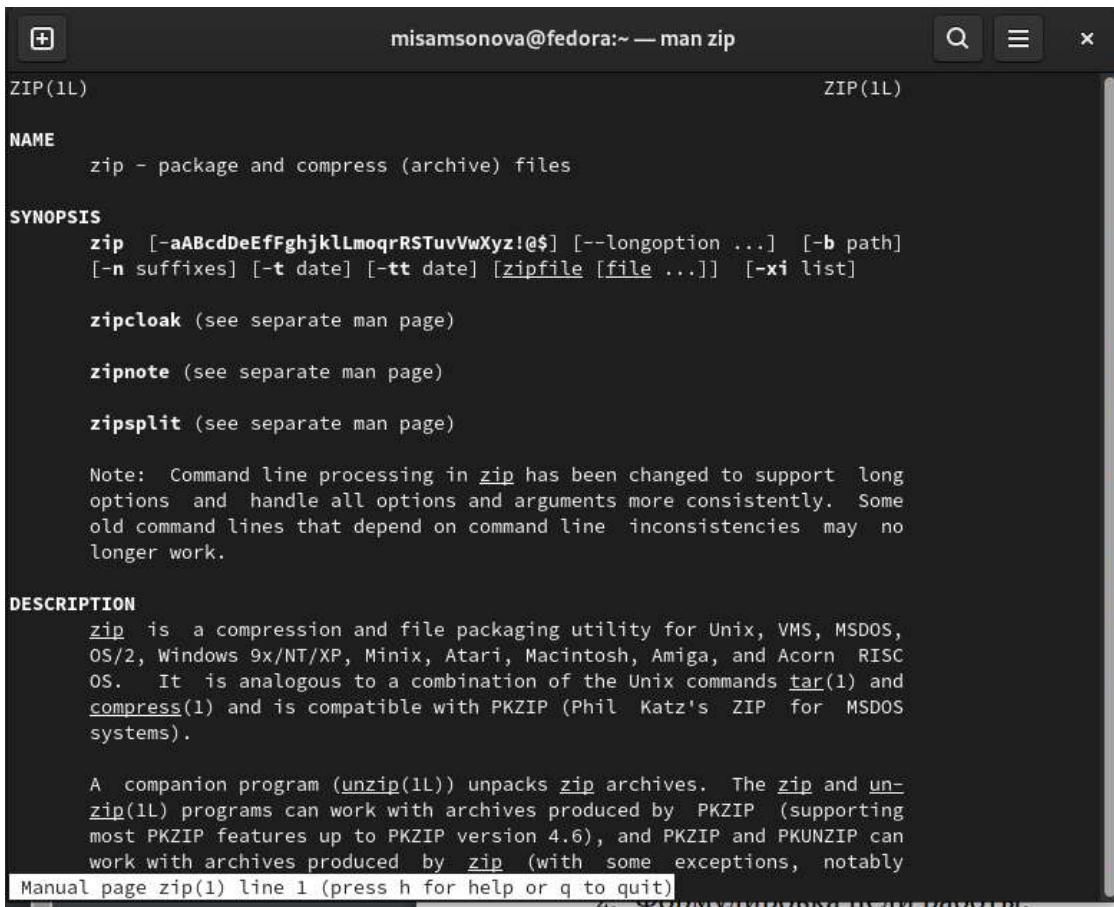
1. Для начала мы изучили команды архивации, используя команды «manzip», «manbzip2», «mantar» (рис. -@fig:001 , -@fig:002 , -@fig:003 , -@fig:004).



```
misamsonova@fedora:~  
[misamsonova@fedora ~]$ man zip  
[misamsonova@fedora ~]$ man bzip2  
[misamsonova@fedora ~]$ man tar  
[misamsonova@fedora ~]$
```

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]



```
misamsonova@fedora:~ — man zip  
ZIP(1L) ZIP(1L)  
NAME  
zip - package and compress (archive) files  
SYNOPSIS  
zip [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$] [--longoption ...] [-b path]  
[-n suffixes] [-t date] [-tt date] [zipfile [file ...]] [-xi list]  
zipcloak (see separate man page)  
zipnote (see separate man page)  
zipsplit (see separate man page)  
Note: Command line processing in zip has been changed to support long  
options and handle all options and arguments more consistently. Some  
old command lines that depend on command line inconsistencies may no  
longer work.  
DESCRIPTION  
zip is a compression and file packaging utility for Unix, VMS, MSDOS,  
OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC  
OS. It is analogous to a combination of the Unix commands tar(1) and  
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 un-  
zip(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  
Manual page zip(1) line 1 (press h for help or q to quit)
```

Информация о zip

- Синтаксис команды bzip2 для архивации файла: bzip2 [опции] [имена файлов]

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

```

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

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

SYNOPSIS
    bzip2 [ -cdfkqstvvVL123456789 ] [ filenames ... ]
    bunzip2 [ -fkvsVL ] [ filenames ... ]
    bzip2cat [ -s ] [ filenames ... ]
    bzip2recover filename

DESCRIPTION
    bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and approaches the performance of the PPM family of statistical compressors.

    The command-line options are deliberately very similar to those of GNU gzip, but they are not identical.

    bzip2 expects a list of file names to accompany the command-line flags. Each file is replaced by a compressed version of itself, with the name "original_name.bz2". Each compressed file has the same modification date, permissions, and, when possible, ownership as the corresponding original, so that these properties can be correctly restored at decompression time. File name handling is naive in the sense that there is no mechanism for preserving original file names, permissions, ownerships or dates in filesystems which lack these concepts, or have serious file name length restrictions, such as MS-DOS.

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

```

Информация о `bzip2`

- Синтаксис команды `tar` для архивации файла:
`tar[опции][архив.tar][файлы_для_архивации]`
- Синтаксис команды `tar` для разархивации/распаковки файла:
`tar[опции][архив.tar]`

```
misamsonova@fedora:~ — man 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 {--catenate|--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)
```

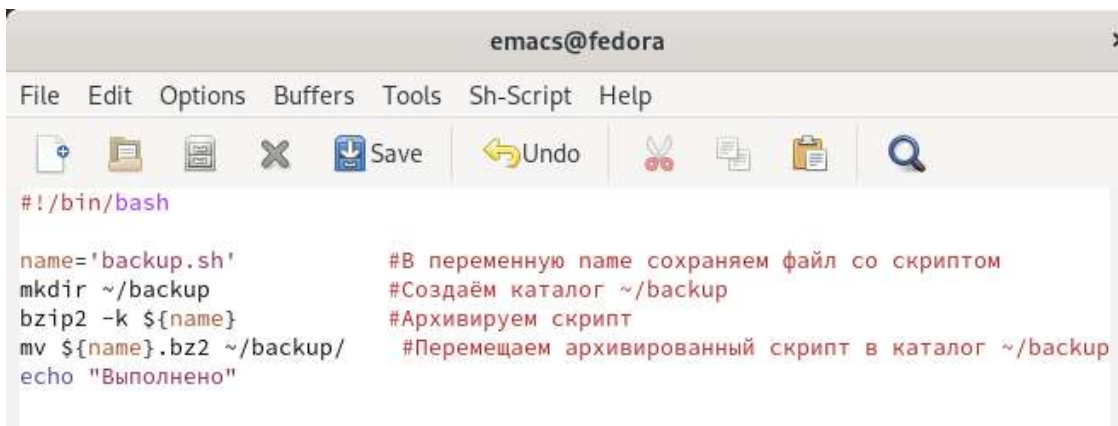
Информация о tar

- Далее создали файл, в котором будем писать первый скрипт, и открыли его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (рис. -@fig:005).

```
misamsonova@fedora:~
[misamsonova@fedora ~]$ touch backup.sh
[misamsonova@fedora ~]$ emacs &
```

Создание файла

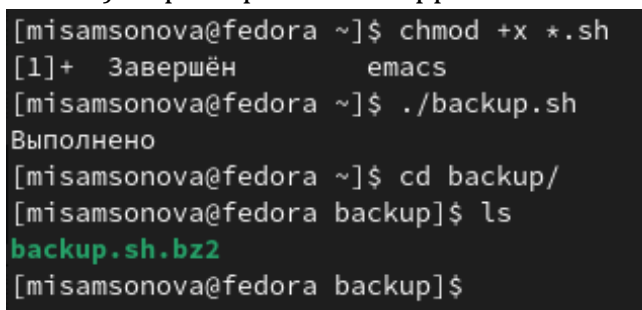
- После этого написали скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию back up в нашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar (рис. -@fig:006). При написании скрипта 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 каталог ~/backup
bzip2 -k ${name}           #Архивируем скрипт
mv ${name}.bz2 ~/backup/   #Перемещаем архивированный скрипт в каталог ~/backup
echo "Выполнено"
```

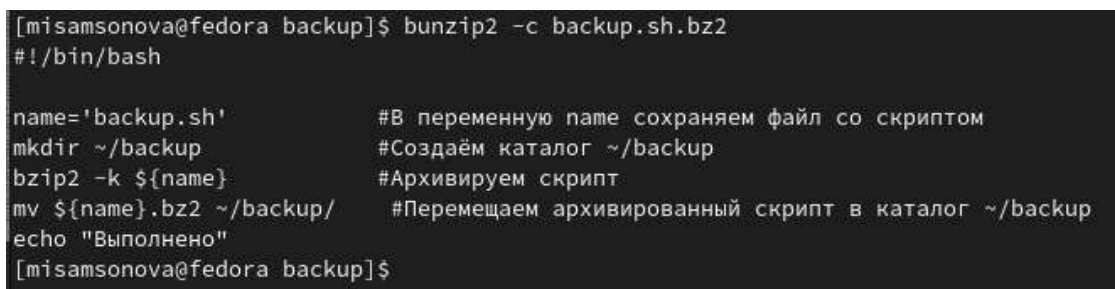
Скрипт №1

- Проверили работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod+x*.sh»). Проверили, появился ли каталог backup/, перейдя в него (команда «cd backup/»), просмотрели содержимое архива (команда «bunzip2 -c backup.sh.bz2») (рис. -@fig:007 , -@fig:008). Скрипт работает корректно.



```
[misamsonova@fedora ~]$ chmod +x *.sh
[1]+  Завершён      emacs
[misamsonova@fedora ~]$ ./backup.sh
Выполнено
[misamsonova@fedora ~]$ cd backup/
[misamsonova@fedora backup]$ ls
backup.sh.bz2
[misamsonova@fedora backup]$
```


Проверка работы скрипта



```
[misamsonova@fedora backup]$ bunzip2 -c backup.sh.bz2
#!/bin/bash
name='backup.sh'          #В переменную name сохраняем файл со скриптом
mkdir ~/backup             #Создаём каталог ~/backup
bzip2 -k ${name}           #Архивируем скрипт
mv ${name}.bz2 ~/backup/   #Перемещаем архивированный скрипт в каталог ~/backup
echo "Выполнено"
[misamsonova@fedora backup]$
```

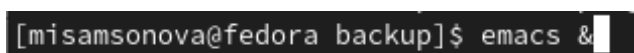
Проверка работы скрипта

2. Теперь создали файл, в котором будем писать второй скрипт, и открыли его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и «emacs &») (рис. -@fig:009 , -@fig:010).



```
[misamsonova@fedora backup]$ touch prog2.sh
```

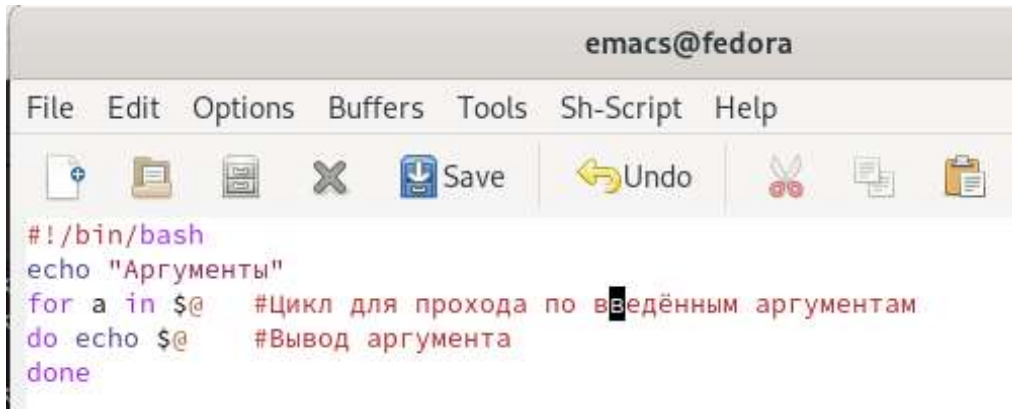
Создание файла



```
[misamsonova@fedora backup]$ emacs &
```

Открываем emacs

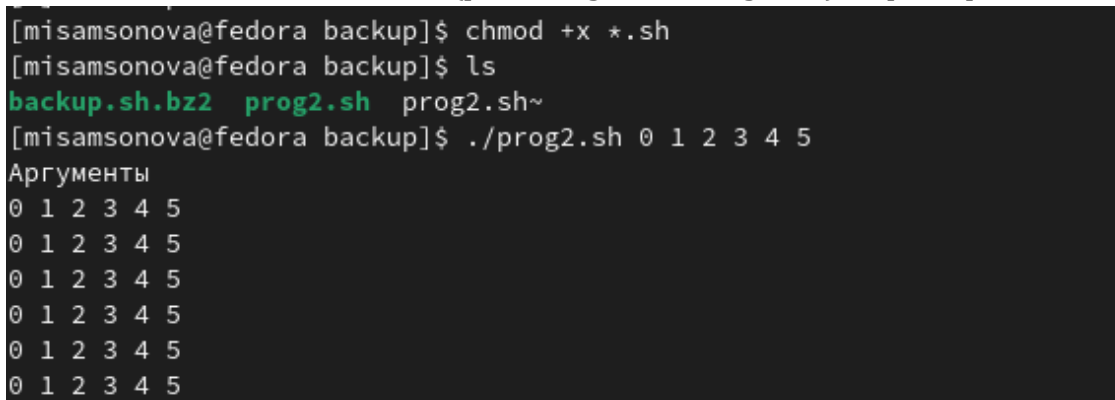
- Написали пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов (рис. -@fig:011).



```
#!/bin/bash
echo "Аргументы"
for a in $@    #Цикл для прохода по введённым аргументам
do echo $a    #Вывод аргумента
done
```

Скринш №2

- Теперь проверили работу написанного скрипта (команды «./prog2.sh 0 1 2 3 4» и «./prog2.sh 0 1 2 3 45 6 7 8 9 10 11»), предварительно добавив для него право на выполнение (команда «chmod+x *.sh»). Ввели аргументы, количество которых меньше 10 и больше 10 (рис. -@fig:012 , -@fig:013). Скрипт работает корректно.



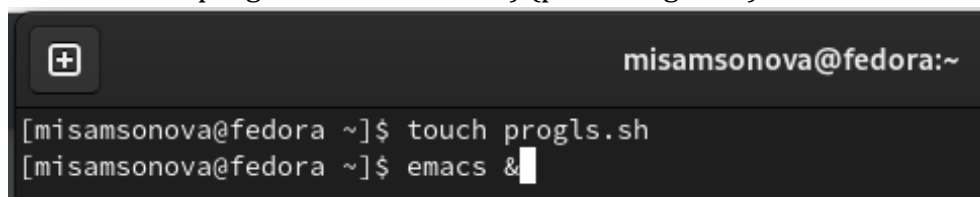
```
[misamsonova@fedora backup]$ chmod +x *.sh
[misamsonova@fedora backup]$ ls
backup.sh.bz2  prog2.sh  prog2.sh~
[misamsonova@fedora backup]$ ./prog2.sh 0 1 2 3 4 5
Аргументы
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
```

Проверка работы скрипта

```
[misamsonova@fedora backup]$ ./prog2.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
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12
[misamsonova@fedora backup]$
```

Проверка работы скрипта

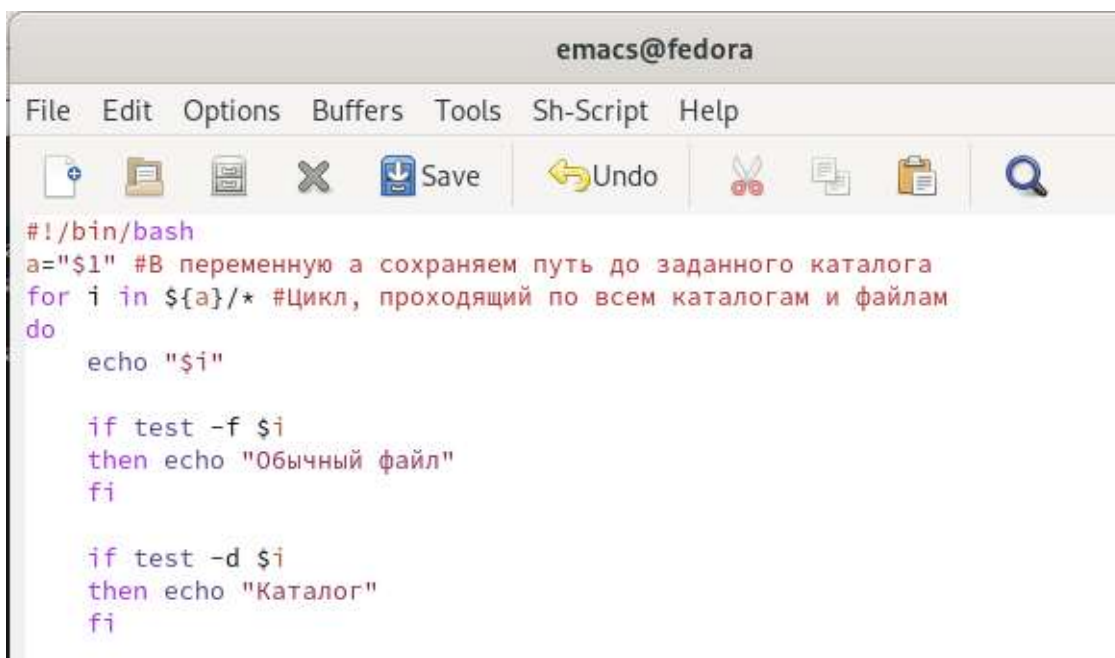
3. После этого создали файл, в котором будем писать третий скрипт, и открыли его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touchprogl.sh» и «emacs&») (рис. -@fig:014).



```
misamsonova@fedora:~
[misamsonova@fedora ~]$ touch progl.sh
[misamsonova@fedora ~]$ emacs &
```

Создание файла

- Написали 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:015 , -@fig:016).

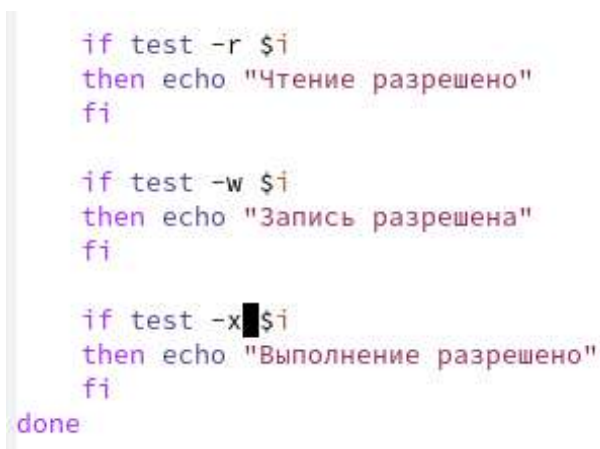


```
#!/bin/bash
a="$1" #В переменную a сохраняем путь до заданного каталога
for i in ${a}/* #Цикл, проходящий по всем каталогам и файлам
do
    echo "$i"

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

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

Скринш №3



```
    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
```

Скринш №3

- Далее проверили работу скрипта (команда «./progl.sh~»), предварительно добавив для него право на выполнение (команда «chmod+x*.sh») (рис. -@fig:017 , -@fig:018 , -@fig:019). Скрипт работает корректно.


```
misamsonova@fedora:~  
[misamsonova@fedora ~]$ chmod +x *.sh  
[misamsonova@fedora ~]$ ls  
australia      example1.txt~  example4.txt  monthlt      Видео  
backup         '#example2.txt#' example4.txt~  monthly      Документы  
backup.sh      example2.txt   feathers      my_os        Загрузки  
backup.sh~     example2.txt~  file.txt      proglis.sh   Изображения  
bin            '#example3.txt#' '#lab07.sh#'  proglis.sh~  Музыка  
blog           '#example3.txt#~' lab07.sh      reports      Общедоступные  
conf.txt       example3.txt   lab08.sh      ski.places   'Рабочий стол'  
'#example1.txt#~' example3.txt~  may           text.txt     Шаблоны  
example1.txt   '#example4.txt#' misamsonova.github.io work  
[misamsonova@fedora ~]$ ./proglis.sh ~  
/home/misamsonova/australia  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/backup  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/backup.sh  
Обычный файл  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/backup.sh~  
Обычный файл  
Чтение разрешено  
Запись разрешена
```

Проверка работы скрипта

```
misamsonova@fedora:~  
/home/misamsonova/backup.sh~  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/misamsonova/bin  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/blog  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/conf.txt  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/misamsonova/#example1.txt#~  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/misamsonova/example1.txt  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/misamsonova/example1.txt~  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/misamsonova/#example2.txt#
```

Проверка работы скрипта

```
misamsonova@fedora:~  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/Изображения  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/Музыка  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/Общедоступные  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/misamsonova/Рабочий стол  
./progl.sh: строка 7: test: /home/misamsonova/Рабочий: ожидается бинарный оператор  
./progl.sh: строка 11: test: /home/misamsonova/Рабочий: ожидается бинарный оператор  
./progl.sh: строка 15: test: /home/misamsonova/Рабочий: ожидается бинарный оператор  
./progl.sh: строка 19: test: /home/misamsonova/Рабочий: ожидается бинарный оператор  
./progl.sh: строка 23: test: /home/misamsonova/Рабочий: ожидается бинарный оператор  
/home/misamsonova/Шаблоны  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
[misamsonova@fedora ~]$
```

Проверка работы скрипта

4. Для четвертого скрипта создали файл (команда «touch format.sh») и открыли его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (рис. -@fig:020).

```
misamsonova@fedora:~  
[misamsonova@fedora ~]$ touch format.sh  
[misamsonova@fedora ~]$ emacs &
```

Создание файла

- После чего написали 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:021).

Скринш №4

- Наконец, проверили 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») (Рисунок -@fig:022). Скрипт работает корректно.

Проверка работы скрипта

Вывод

В процессе выполнения лабораторной работы №10 мы изучили основы программирования в оболочке ОС UNIX/Linux и научились писать небольшие командные файлы.

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:
 - 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 `.` 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` 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:
 - 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 `unsetс`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 –количеств 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.