

Отчет к лабораторной работе №10

Операционные системы

Газизова Регина

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	13
4	Контрольные вопросы	14

Список иллюстраций

2.1	Команда man	6
2.2	Архив zip	6
2.3	Архив bzip2	7
2.4	Архив tar	7
2.5	Создание файла	8
2.6	Программа	8
2.7	Права доступа	8
2.8	Просмотр каталога	8
2.9	Просмотр содержимого	9
2.10	Создание файла	9
2.11	Программа	9
2.12	Выполнение	10
2.13	Код программы	10
2.14	Запуск файла	11
2.15	Создание файла	11
2.16	Код программы	12
2.17	Запуск файла	12

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1. Для начала изучим команды архивации с помощью команд `man bzip2`, `man zip`, `man tar` (рис.1).

```
rigazizova@dk8n74 ~ $ man zip
rigazizova@dk8n74 ~ $ man bzip2
rigazizova@dk8n74 ~ $ man tar
rigazizova@dk8n74 ~ $ touch backup.sh
rigazizova@dk8n74 ~ $ emacs $
```

Рис. 2.1: Команда man

Синтаксис команды `zip` для архивации файлов: `zip [опции] [имя_файла.zip]` [файлы или папки для архивации]. Синтаксис для разархивации файла: `unzip [опции] [файл_архива.zip] [файлы] -x [исключить] -d [папки]` (рис.2).

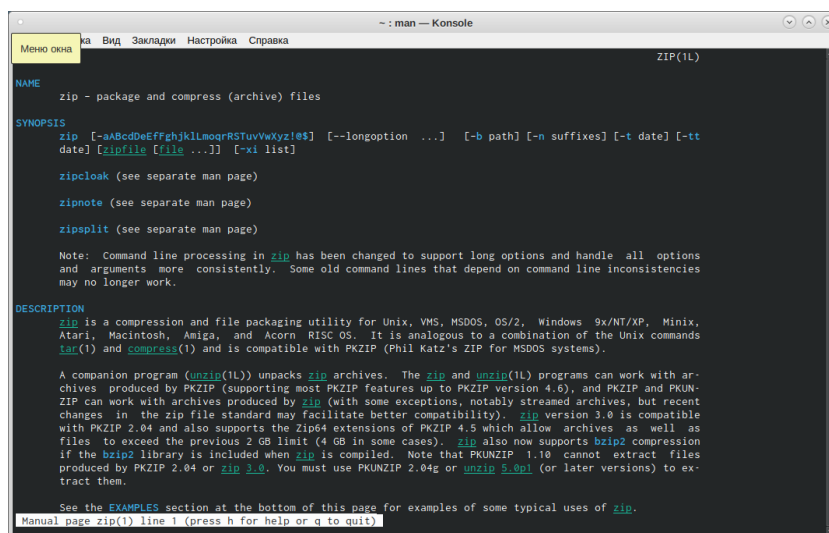
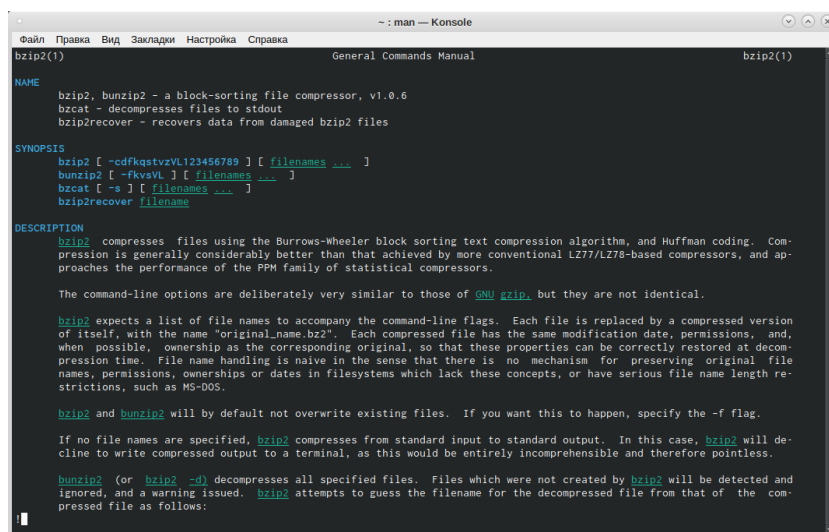


Рис. 2.2: Архив zip

Синтаксис команды bzip2 для архивации файлов: bzip2 [опции] [имя_файла].
Синтаксис для разархивации файла: bunzip2 [опции] [файл_архива.bz2] (рис.3).



```
~: man -- Konsole
bzip2(1)                                     General Commands Manual      bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.6
  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.

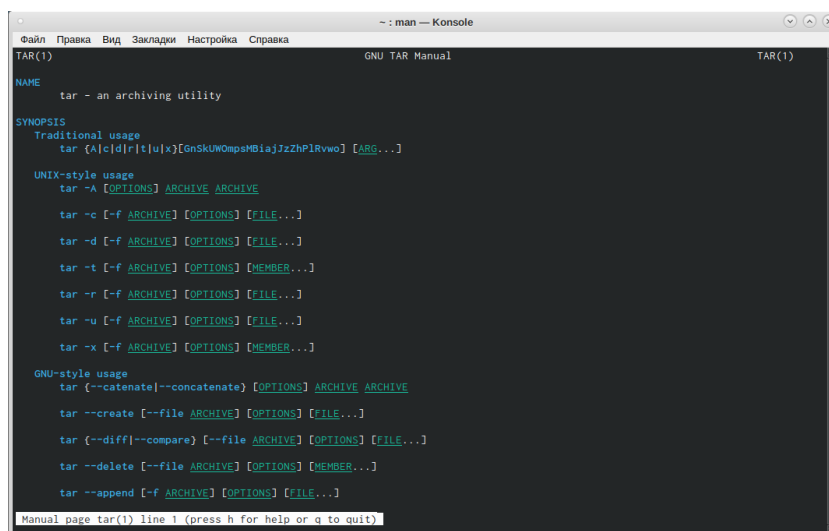
  bzip2 and bunzip2 will by default not overwrite existing files. If you want this to happen, specify the -f flag.

  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 attempts to guess the filename for the decompressed file from that of the compressed file as follows:
```

Рис. 2.3: Архив bzip2

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



```
~: man -- Konsole
TAR(1)                                     GNU TAR Manual              TAR(1)

NAME
  tar - an archiving utility

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

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

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

  GNU-style usage
  tar [--catenate|--concatenate] [OPTIONS] ARCHIVE ARCHIVE
  tar --create [--file ARCHIVE] [OPTIONS] [FILE...]
  tar [--diff|--compare] [--file ARCHIVE] [OPTIONS] [FILE...]
  tar --delete [--file ARCHIVE] [OPTIONS] [MEMBER...]
  tar --append [-f ARCHIVE] [OPTIONS] [FILE...]

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

Рис. 2.4: Архив tar

Создаем файл backup.sh, в котором напишем первый скрипт, и откроем в редакторе emacs (рис.5)


```
rigazizova@dk8n74 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

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

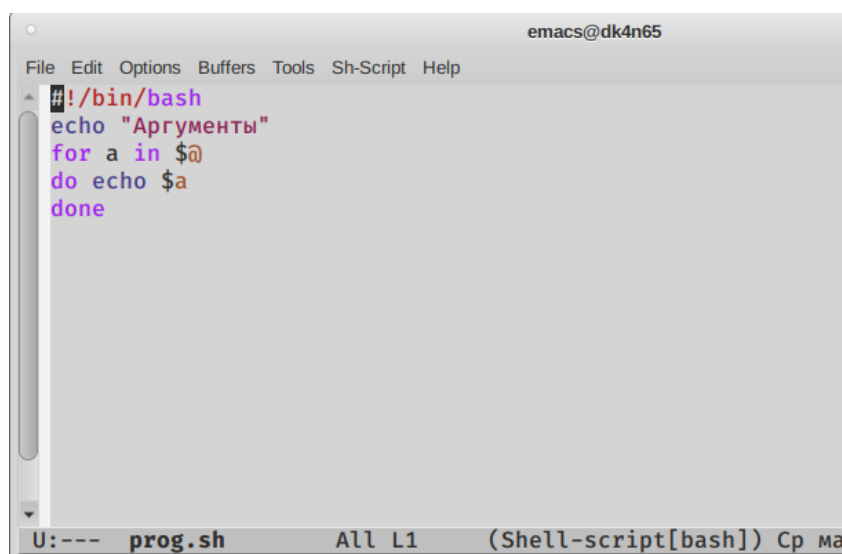
Рис. 2.9: Просмотр содержимого

2. Создадим файл prog.sh, в котором напомним второй скрипт, и откроем его в emacs (рис.10).

```
rigazizova@dk8n74 ~/backup $ touch prog.sh
rigazizova@dk8n74 ~/backup $ emacs $
```

Рис. 2.10: Создание файла

Напишем 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.11).



```
#!/bin/bash
echo "Аргументы"
for a in $@
do echo $a
done
```

Рис. 2.11: Программа

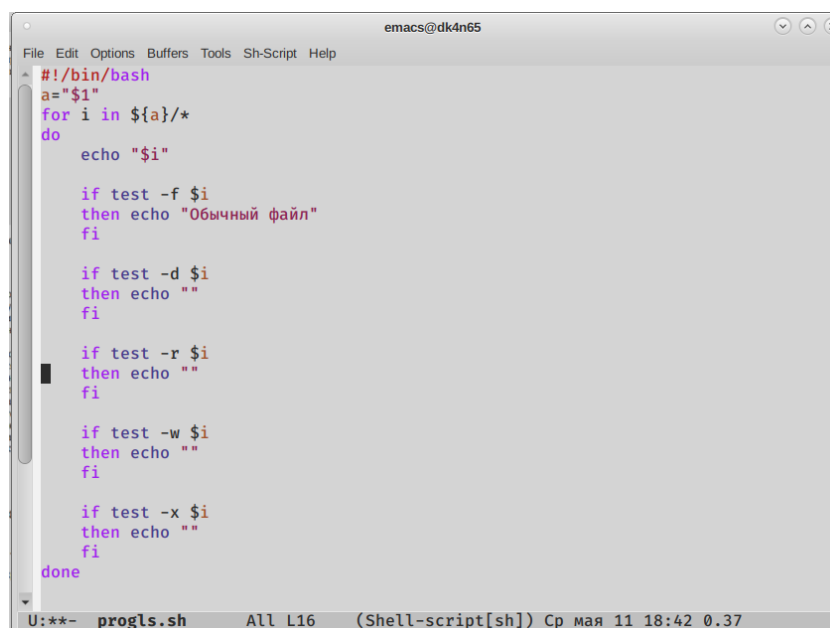
Проверим его работу, предварительно передав права доступа на выполнение, вводим разное количество аргументов меньше и больше 10 (рис.12).

```
ukmorozova@dk4n65 ~ $ ./prog.sh 1 2 3 1 4 10 11 23
Аргументы
1
2
3
1
4
10
11
23
```

Рис. 2.12: Выполнение

3. Создадим файл progls.sh для третьего скрипта (команда touch progls.sh) и откроем его в emacs.

Напишем командный файл — аналог команды ls (без использования самой этой команды и команды dir) (рис.13). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.



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

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

    if test -d $i
    then echo ""
    fi

    if test -r $i
    then echo ""
    fi

    if test -w $i
    then echo ""
    fi

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

Рис. 2.13: Код программы

Проверим его работу, предварительно передав права доступа на выполнение (chmod +x *.sh) и командой ./progl.sh запустим его (рис.14).

```

rigazizova@dk8n74 ~ $ emacs $
rigazizova@dk8n74 ~ $ chmod +x *.sh
rigazizova@dk8n74 ~ $ ./proglis.sh ~
/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova
Обычный файл

```

Рис. 2.14: Запуск файла

4. Создадим файл `format.sh` для четвертого скрипта (команда `touch format.sh`) и откроем его в `emacs` (рис.15).

```

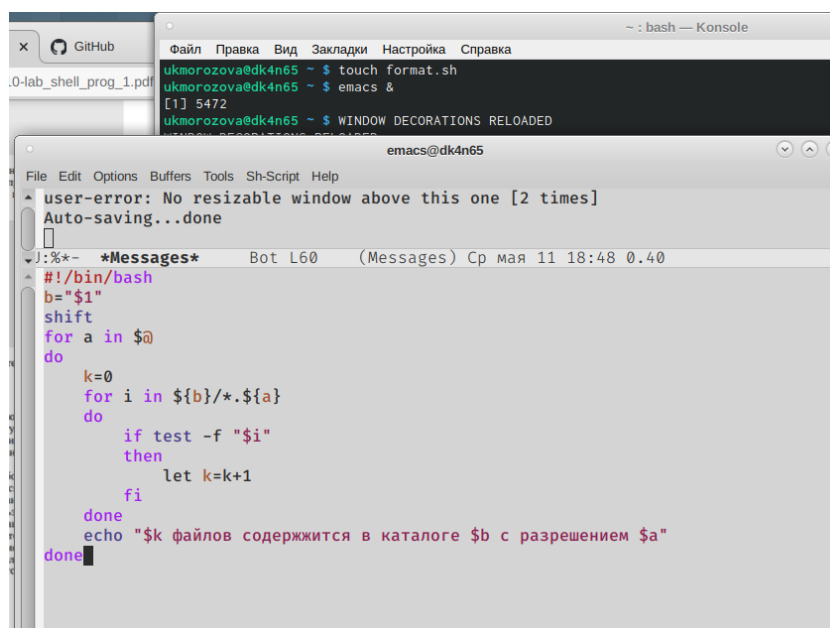
rigazizova@dk8n74 ~ $ touch format.sh
rigazizova@dk8n74 ~ $ emacs $

```

Рис. 2.15: Создание файла

Напишем 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.16).



```

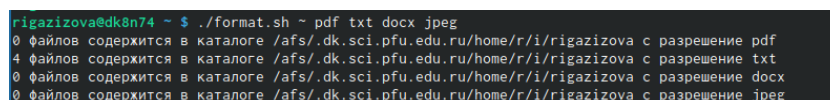
ukmorofova@dk4n65 ~ $ touch format.sh
ukmorofova@dk4n65 ~ $ emacs &
[1] 5472
ukmorofova@dk4n65 ~ $ WINDOW DECORATIONS RELOADED

emacs@dk4n65
File Edit Options Buffers Tools Sh-Script Help
user-error: No resizable window above this one [2 times]
Auto-saving...done
j:~*~ *Messages* Bot L60 (Messages) Cp мая 11 18:48 0.40
#!/bin/bash
b="$1"
shift
for a in $@
do
    k=0
    for i in ${b}/*.$a
    do
        if test -f "$i"
        then
            let k=k+1
        fi
    done
    echo "$k файлов содержится в каталоге $b с разрешением $a"
done

```

Рис. 2.16: Код программы

Проверим его работу, предварительно передав права доступа на выполнение (chmod +x *.sh) и командой ./format.sh ~ pdf txt docx запустим его (рис.17).



```

rigazizova@dk8n74 ~ $ ./format.sh ~ pdf txt docx jpeg
0 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova с разрешение pdf
4 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova с разрешение txt
0 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova с разрешение docx
0 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/r/i/rigazizova с разрешение jpeg

```

Рис. 2.17: Запуск файла

3 Выводы

Я научилась писать небольшие команды в оболочке Linux.

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; C-оболочка (или csh) – надстройка на оболочке 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 `.`, «*`mva file{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 `setc` 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 единственный 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 отобрать 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 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 ка-

вычки. 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 `unsetc`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.