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

1. для начала я изучил команды архивации,используя команды “man zip , man bzip2,maz tar” (рис1) (рис2) (рис3) (рис4)

```
[rrManatov@fedora ~]$ man zip
[rrManatov@fedora ~]$ man bzip
Нет справочной страницы для bzip
[rrManatov@fedora ~]$ man bzip2
[rrManatov@fedora ~]$ man tar
```

Рисунок 1 получаем информацию

```
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 unzip(1L) programs can  work  with  ar-
       chives  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.

       Large Archives and Zip64.   zip  automatically  uses  the Zip64 extensions when files larger than 4 GB are
       added to an archive, an archive containing Zip64 entries is updated (if the resulting archive still  needs
       Zip64), the size of the archive will exceed 4 GB, or when the number of entries in the archive will exceed
       about 64K.  Zip64 is also used for archives streamed from standard input as the size of such archives  are
       not  known  in advance, but the option -fz- can be used to force zip to create PKZIP 2 compatible archives
       (as long as Zip64 extensions are not needed).  You must use a PKZIP 4.5  compatible  unzip,  such  as  un-
       zip 6.0 or later, to extract files using the Zip64 extensions.

       In addition, streamed archives, entries encrypted with standard encryption, or split archives created with
       the pause option may not be compatible with PKZIP as data descriptors are used and PKZIP at  the  time  of
       this  writing  does  not support data descriptors (but recent changes in the PKWare published zip standard
 Manual page zip(1) line 1 (press h for help or q to quit)
```

Рисунок 2 информация об man zip

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 [**-cdfkqstvzVL123456789**] [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:

filename.bz2	becomes	filename
filename.bz	becomes	filename
filename.tbz2	becomes	filename.tar
filename.tbz	becomes	filename.tar
anyothername	becomes	anyothername.out

If the file does not end in one of the recognised endings, .bz2, .bz, .tbz2 or .tbz, bzip2 complains that it cannot guess the name of the original file, and uses the original name with .out appended.

Рисунок 3 информация об man bzip2

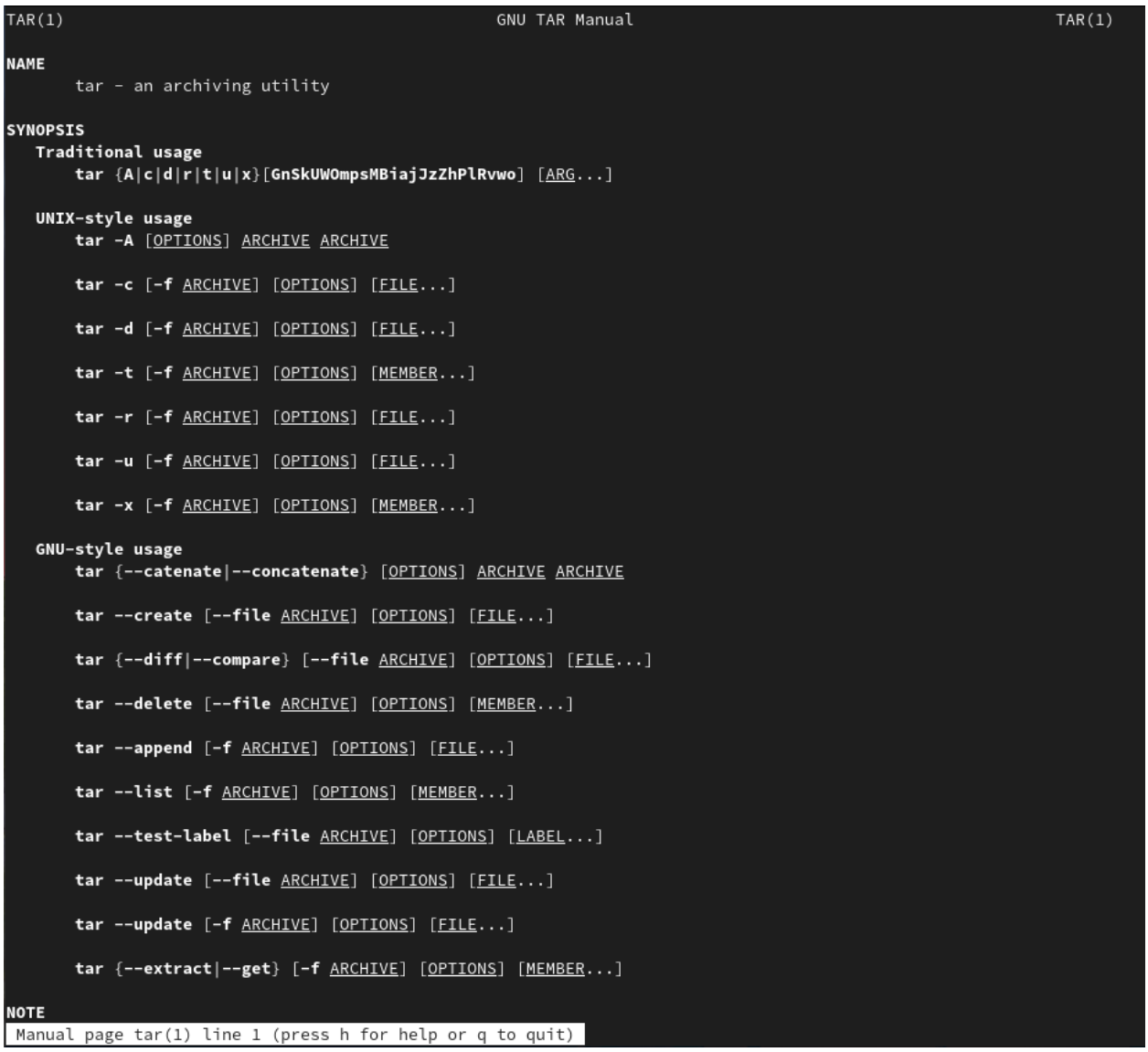

Рисунок 4 information about man tar

2) Для начала я создал каталог где я буду работать и файл где буду работать (рис5)

```
[rrManatov@fedora ~]$ cd lab10
[rrManatov@fedora lab10]$ touch backup.sh
[rrManatov@fedora lab10]$ emacs &
```

Рисунок 5 Рисунок 5 начинаем редактировать файл

После напишу 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

```
#!/bin/bash

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

Рисунок 6 код для программы

3)Теперь проверим работу скрипта предварительно добавив для него право на выполнение .Проверим также работу нашей программы

```
[rrManatov@fedora ~]$ ls
 abc1        '~exit'     '#lAb.04PRO#'   lab07       '#text.txt#'   work.       Загрузки       'Рабочий стол'
 backup.sh    file.cpp    lAb.04PRO      lab09        text.txt      work_       Изображения     Шаблоны
 backup.sh~   file.old.   lAB.04.PRO    os-intro     tmp           Видео       Музыка
 conf.txt     file.txt    lab05          ski.plases   work          Документы   Общедоступные
[1]+  Завершён        emacs
[rrManatov@fedora ~]$ cdmod +x *.sh
bash: cdmod: command not found...
[rrManatov@fedora ~]$ chmod +x *.sh
[rrManatov@fedora ~]$ ./backup.sh
Выполнено
[rrManatov@fedora ~]$ cd backup/
[rrManatov@fedora backup]$ ls
backup.sh.bz2
[rrManatov@fedora backup]$
```

Рисунок 7 Проверили работу нашей программы

```
[rrManatov@fedora backup]$ bunzip2 -c backup.sh.bz2
#!/bin/bash

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

Рисунок 8

2. Создадим файл где я буду писать второй код и откроем его в редакторе emacs

```
[rrManatov@fedora ~]$ touch code2.sh
[rrManatov@fedora ~]$ emacs &
[2] 3032
```

Рисунок 9

Написал пример командного файла ,обрабатывающего любое произвольное число аргументов командной строки,в том числе превышающее десять.Например код распечатывающий значения всех переданных аргументов

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

Рисунок 10 код программы

1)Проверим работу написанного кода предварительно добавив право на выполнение команда chmod

```
[rrManatov@fedora ~]$ chmod +x *.sh
[rrManatov@fedora ~]$ ls
 abc1         code2.sh    file.cpp       lAb.04PRO    lab09         text.txt   work_       Изображения    Шаблоны
 backup       code2.sh~   file.old.      lAB.04.PRO   os-intro      tmp        Видео       Музыка
 backup.sh    conf.txt    file.txt       lab05        ski.plases    work       Документы   Общедоступные
 backup.sh~  '~exit'     '#lAb.04PRO#'   lab07       '#text.txt#'   work.      Загрузки   'Рабочий стол'
[rrManatov@fedora ~]$ ./code2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
[rrManatov@fedora ~]$ ./code2.sh 0 1 2 3 4 5 6 7 8 9 10 11
Аргументы
0
1
2
3
4
5
6
7
8
9
10
11
[rrManatov@fedora ~]$
```

Рисунок 11 программа работает корректно

3. Создам файл я,в котором буду писать третий код и откроем его в редакторе emacs

```
[rrManatov@fedora ~]$ touch progls.sh
[2]+  Завершён        emacs
[rrManatov@fedora ~]$ emacs &
[2] 3348
```

Рисунок 12

2)Написал командный файл аналог команды ls .Он должен выдавать информацию о нужно каталоге и возможностях доступа к файлам этого каталога

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

    echo "$i"

    if test -f $1
    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 $1
    then echo "Запись разрешена"
    fi

    if test -x $1
    then echo "Выполнение разрешено"
    fi
done
```

Рисунок 13 файл с кодом

3)Далее проверим работу файла предварительно добавив для него право на выполнение

```
[rrManatov@fedora ~]$ chmod +x *.sh
[rrManatov@fedora ~]$ ls
 abc1         code2.sh~   file.txt       lab07        ski.plases    work.        Изображения
 backup       conf.txt   '#lAb.04PRO#'   lab09       '#text.txt#'   work_        Музыка
 backup.sh   '~exit'      lAb.04PRO      os-intro     text.txt      Видео        Общедоступные
 backup.sh~   file.cpp    lAB.04.PRO     progls.sh    tmp           Документы   'Рабочий стол'
 code2.sh     file.old.   lab05          progls.sh~   work          Загрузки    Шаблоны
[rrManatov@fedora ~]$ ./progls.sh ~
/home/rrManatov/abc1
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/rrManatov/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/rrManatov/backup.sh
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/rrManatov/backup.sh~
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/rrManatov/code2.sh
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/rrManatov/code2.sh~
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/rrManatov/conf.txt
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/rrManatov/~exit
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/rrManatov/file.cpp
Чтение разрешено
```

Рисунок 14 Код работает корректно

4.Для четвертого скрипта также создадим файл и откроем его в редакторе emacs (рис15)

```
[rrManatov@fedora ~]$ touch format.sh
[rrManatov@fedora ~]$ emacs &
[1] 3563
```

Рисунок 15

Напишем командный файл,Который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории.Путь к директории также передается в виде аргумента командной строки

```
#!/bin/bash
b="$1"
shift
for a in $@
do

    k = 0
    for i in ${b}/*.${a}
    do

        if test -f "$1"
        then
            let k=k+1
        fi
    done
    echo "$k файлов содержится в каталоге $b с расширением $a"
done
```

Рисунок 16

Проверим работу написанного кода

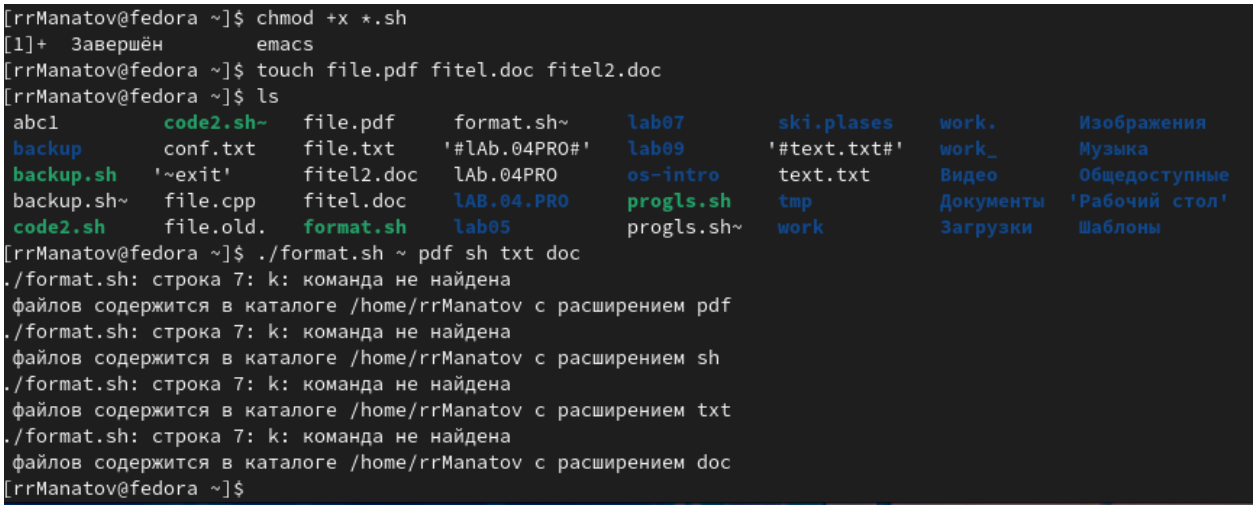

Рисунок 17 работает корректно

Вывод: В ходе выполнения данной лабораторной работы я изучил 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 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/Linuxподобных 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

4) Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение − это единичный терм (term), обычно целочисленный. Команда let берет два операнда и присваивает их переменной. Команда read позволяет читать значения переменных со стандартного ввода: «echo "Please enter Month and Day of Birth ?"» «read mon day trash» В переменные mon и day будут считаны соответствующие значения, введённые с клавиатуры, а переменная trash нужна для того, чтобы 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 (new line). • MAIL: командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение You have mail (у Вас есть почта). • TERM: тип используемого терминала. • LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

8) Такие символы, как ' < > * ? | \ " &, являются метасимволами и имеют для командного процессора специальный смысл.

9) Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа \, который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует

все метасимволы, кроме $, ' , \, ". Например, – echo * выведет на экран символ *, – echo ab'*\|*'cd выведет на экран строку ab*\|*cd.

10) Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: «bash командный_файл [аргументы]» Чтобы не вводить каждый раз последовательности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды «chmod +x имя_файла» Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит её интерпретацию.

11) Группу команд можно объединить в функцию. Для этого существует 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 гделибо 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