

Цель работы

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

Задание

Написать несколько скриптов.

Выполнение лабораторной работы

Изучим команды архивации, используя команды «man zip», «man bzip2», «man tar». (рис. -@fig:001)

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

{ #fig:001 width=70% }

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] (рис. -@fig:002).

```
ZIP(1) General Commands Manual ZIP(1)
NAME
  zip - package and compress (archive) files
SYNOPSIS
  zip [-aABcdDeFFghjklLmoqrSTuvVwXyzI@S] [--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, Mlnix, Atari, Macintosh, Amiga, and Acorn RISC OS. It is analogous to a combination of the Unix commands tar(1) and compress(1) and is compatible with PKZIP (Phil Katz's ZIP for MSDOS systems).
  A companion program (unzip(1)) unpacks zip archives. The zip and unzip(1) programs can work with archives produced by PKZIP (supporting most PKZIP features up to PKZIP version 4.6), and PKZIP and PKUNZIP can work with archives produced by zip (with some exceptions, notably streamed archives, but recent changes in the zip file standard may facilitate better compatibility). zip version 3.0 is compatible with PKZIP 2.04 and also supports the Zip64 extensions of PKZIP 4.5 which allow archives as well as files to exceed the previous 2 GB limit (4 GB in some cases). zip also now supports bzip2 compression if the bzip2 library is included when zip is compiled. Note that PKUNZIP 1.10 cannot extract files produced by PKZIP 2.04 or zip 3.0. You must use PKUNZIP 2.04g or unzip 5.0p1 (or later versions) to extract them.
  See the EXAMPLES section at the bottom of this page for examples of some typical uses of zip.
```

{ #fig:002 width=70% }

Синтаксис команды bzip2 для архивации файла: bzip2 [опции] [имена файлов] Синтаксис команды bzip2 для разархивации/распаковки файла: bunzip2 [опции] [архивы.bz2] (рис. -@fig:003).

```
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 [ -cdfkqstvzVL123456789 ] [ filenames ... ]
  bzip2 [ -h|--help ]
  bunzip2 [ -fkvVL ] [ filenames ... ]
  bunzip2 [ -h|--help ]
  bzip2 [ -s ] [ filenames ... ]
  bzip2 [ -h|--help ]
  bzip2recover filename
DESCRIPTION
  bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and approaches the performance of the PPM family of statistical compressors.
  The command-line options are deliberately very similar to those of GNU gzip, but they are not identical.
  bzip2 expects a list of file names to accompany the command-line flags. Each file is replaced by a compressed version of itself, with the name "original_name.bz2". Each compressed file has the same modification date, permissions, and, when possible, ownership as the corresponding original, so that these properties can be correctly restored at decompression time. File name handling is native 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.
```

{ #fig:003 width=70% }

Синтаксис команды tar для архивации файла: tar [опции] [архив.tar] [файлы_для_архивации] Синтаксис команды tar для разархивации/распаковки файла: tar [опции] [архив.tar] (рис. -@fig:004).



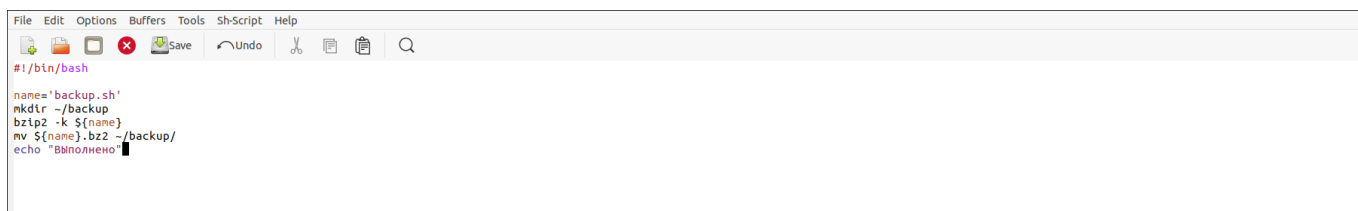
{ #fig:004 width=70% }

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



{ #fig:005 width=70% }

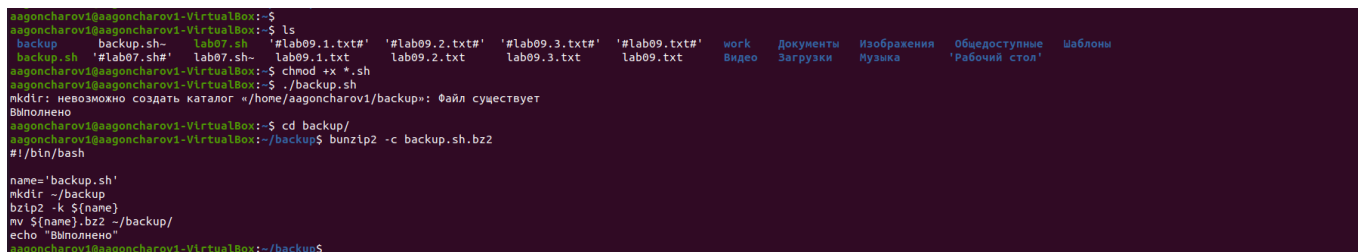
Далее напишем скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в нашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. При написании скрипта используем архиватор bzip2 (рис. -@fig:006).



{ #fig:006 width=70% }

Проверим работу скрипта командой «./backup.sh», предварительно добавив для него право на выполнение командой «chmod +x *.sh».

Проверили, появился ли каталог backup/, перейдя в него командой «cd backup/», просмотрели его содержимое командой «ls» и просмотрели содержимое архива командой «bunzip2 -c backup.sh.bz2». Скрипт работает корректно (рис. -@fig:007).



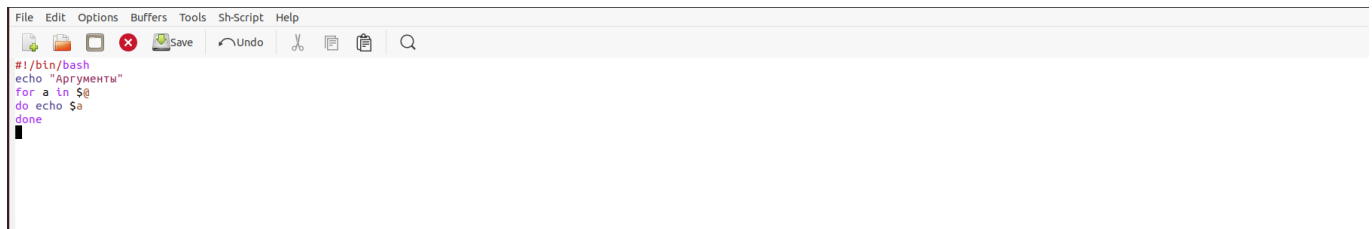
{ #fig:007 width=70% }

Создадим файл, в котором будем писать второй скрипт, и откроем его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f», командой «touch prog2.sh» и «emacs &» (рис. -@fig:008).

```
aagoncharov@aagoncharov1-VirtualBox:~/backup$ touch prog2.sh
aagoncharov@aagoncharov1-VirtualBox:~/backup$ emacs &
[1] 2084
```

{ #fig:008 width=70% }

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. -@fig:009).



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

{ #fig:009 width=70% }

Проверим работу написанного скрипта командой «./prog2.sh 0 1 2 3 4» и «./prog2.sh 1 2 3 4 5 6 7 8 9 10 11», предварительно добавив для него право на выполнение командой «chmod +x *.sh». Вводим аргументы, количество которых меньше 10 и больше 10. Скрипт работает корректно (рис. -@fig:010).

```
aagoncharov@aagoncharov1-VirtualBox:~/backup$ chmod +x *.sh
aagoncharov@aagoncharov1-VirtualBox:~/backup$ ls
backup.sh.bz2  prog2.sh  prog2.sh-
aagoncharov@aagoncharov1-VirtualBox:~/backup$ cd ..
aagoncharov@aagoncharov1-VirtualBox:~$ chmod +x *.sh
aagoncharov@aagoncharov1-VirtualBox:~$ ls
backup  backup.sh  lab07.sh  'lab09.1.txt#'  'lab09.2.txt#'  'lab09.3.txt#'  'lab09.txt#'  work  Документы  Изображения  Общедоступные  Шаблоны
aagoncharov@aagoncharov1-VirtualBox:~$ ./prog2.sh 1 2 3 4
bash: ./prog2.sh: Нет такого файла или каталога
aagoncharov@aagoncharov1-VirtualBox:~$ cd backup/
aagoncharov@aagoncharov1-VirtualBox:~/backup$ cp prog2.sh prog2.sh~
aagoncharov@aagoncharov1-VirtualBox:~/backup$ cd
aagoncharov@aagoncharov1-VirtualBox:~$ ls
backup  backup.sh  lab07.sh  'lab09.1.txt#'  'lab09.2.txt#'  'lab09.3.txt#'  'lab09.txt#'  prog2.sh  work  Документы  Изображения  Общедоступные  Шаблоны
aagoncharov@aagoncharov1-VirtualBox:~$ ./prog2.sh 1 2 3 4
Аргументы
1
2
3
4
aagoncharov@aagoncharov1-VirtualBox:~$ ./prog2.sh 1 2 3 4 5 6 7 8 9 10 11
Аргументы
1
2
3
4
5
6
7
8
9
10
11
aagoncharov@aagoncharov1-VirtualBox:~$
```

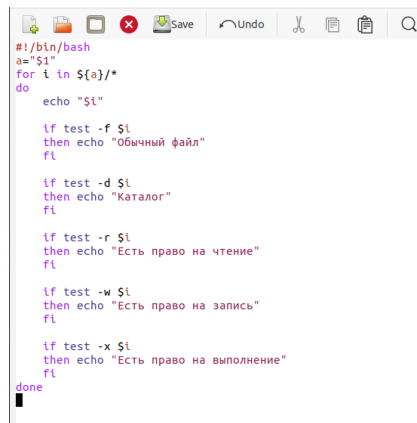
{ #fig:010 width=70% }

Создадим файл, в котором будем писать третий скрипт, и откроем его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f», командами «touch proglis.sh» и «emacs &». (рис. -@fig:011).

```
aagoncharov@aagoncharov1-VirtualBox:~$ touch proglis.sh
aagoncharov@aagoncharov1-VirtualBox:~$ emacs &
[1] 2184
aagoncharov@aagoncharov1-VirtualBox:~$
```

{ #fig:011 width=70% }

Напишем 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:012).



```
#!/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 "Каталог"
    fi

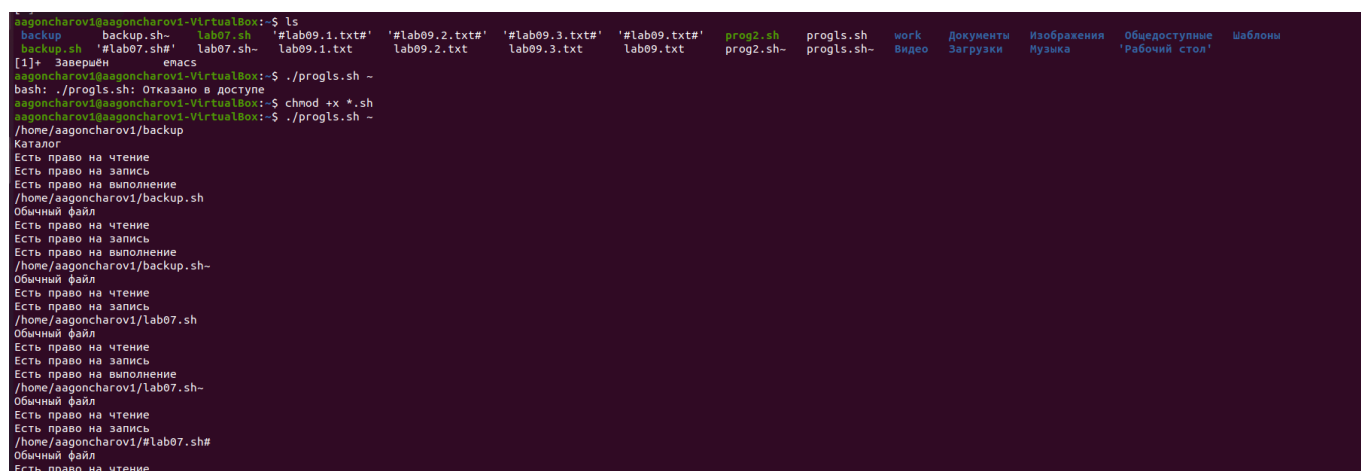
    if test -r $i
    then echo "Есть право на чтение"
    fi

    if test -w $i
    then echo "Есть право на запись"
    fi

    if test -x $i
    then echo "Есть право на выполнение"
    fi
done
```

{ #fig:012 width=70% }

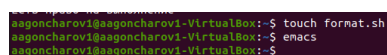
Далее проверим работу скрипта командой «./progl.sh ~», предварительно добавив для него право на выполнение командой «chmod +x *.sh». Скрипт работает корректно (рис. -@fig:013).



```
aagongcharov1@aagongcharov1-VirtualBox:~$ ls
backup  backup.sh  lab07.sh  #lab09.1.txt#  #lab09.2.txt#  #lab09.3.txt#  #lab09.txt#  prog2.sh  progl.sh  work
backup.sh  #lab07.sh#  lab07.sh-  lab09.1.txt  lab09.2.txt  lab09.3.txt  lab09.txt  prog2.sh-  progl.sh-  Видео
[1]+  Завершен  emacs
aagongcharov1@aagongcharov1-VirtualBox:~$ ./progl.sh ~
bash: ./progl.sh: Отказано в доступе
aagongcharov1@aagongcharov1-VirtualBox:~$ chmod +x *.sh
aagongcharov1@aagongcharov1-VirtualBox:~$ ./progl.sh ~
Каталог
Есть право на чтение
Есть право на запись
Есть право на выполнение
Есть право на выполнение
/home/aagongcharov1/backup.sh
Обычный файл
Есть право на чтение
Есть право на запись
Есть право на выполнение
/home/aagongcharov1/backup.sh-
Обычный файл
Есть право на чтение
Есть право на запись
Есть право на выполнение
/home/aagongcharov1/lab07.sh
Обычный файл
Есть право на чтение
Есть право на запись
Есть право на выполнение
/home/aagongcharov1/lab07.sh-
Обычный файл
Есть право на чтение
Есть право на запись
Есть право на выполнение
/home/aagongcharov1/lab07.sh#
Обычный файл
Есть право на чтение
```

{ #fig:013 width=70% }

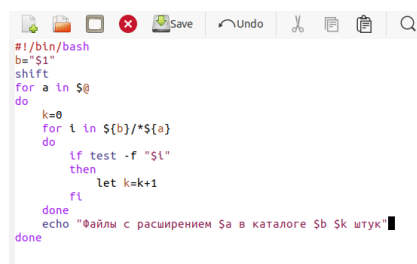
Для четвертого скрипта также создадим файл командой «touch format.sh» и откроем его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (рис. -@fig:014).



```
aagongcharov1@aagongcharov1-VirtualBox:~$ touch format.sh
aagongcharov1@aagongcharov1-VirtualBox:~$ emacs
aagongcharov1@aagongcharov1-VirtualBox:~$
```

{ #fig:014 width=70% }

Напишем командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и считает количество таких файлов в указанной директории. Путь к директории также передается в виде аргумента командной строки (рис. -@fig:015).



```
#!/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 "Файлы с расширением $a в каталоге $b $k штук"
done
```

{ #fig:015 width=70% }

Проверили работу написанного скрипта командой «./format.sh ~ pdf sh txt doc», предварительно добавив для него право на выполнение командой «chmod +x *.sh», а также создав дополнительные

```

aagongcharov@aagongcharov-VirtualBox:~$ touch format.sh
aagongcharov@aagongcharov-VirtualBox:~$ emacs
aagongcharov@aagongcharov-VirtualBox:~$ chmod +x *.sh
aagongcharov@aagongcharov-VirtualBox:~$ cd /home/aagongcharov
aagongcharov@aagongcharov-VirtualBox:~/aagongcharov$ ls
backup  backup.sh  format.sh  lab09_1.txt#  #lab09_2.txt#  #lab09_3.txt#  #lab09.txt#  prog2.sh  proglis.sh  work  документы  Изображения  Общедоступные  Шаблоны
backup.sh  format.sh  #lab07.sh#  lab07.sh-  lab09_1.txt  lab09_2.txt  lab09_3.txt  lab09.txt  prog2.sh-  proglis.sh-  Видео  Загрузки  Музыка  "Рабочий стол"
aagongcharov@aagongcharov-VirtualBox:~/aagongcharov$ ./format.sh -txt sh
Файлы с расширением txt в каталоге /home/aagongcharov: 4 штук
Файлы с расширением sh в каталоге /home/aagongcharov: 5 штук
aagongcharov@aagongcharov-VirtualBox:~$

```

Контрольные вопросы

- 5 / 8

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 (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.

Выводы

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