

Отчёт по лабораторной работе номер 11

Операционные системы

Нитусова Диана Денисовна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Контрольные вопросы	15
5	Выводы	21

Список таблиц

Список иллюстраций

3.1	Рисунок 1	7
3.2	Рисунок 2	7
3.3	Рисунок 3	8
3.4	Рисунок 4	8
3.5	Рисунок 5	9
3.6	Рисунок 6	9
3.7	Рисунок 7	10
3.8	Рисунок 8	10
3.9	Рисунок 9	10
3.10	Рисунок 10	11
3.11	Рисунок 11	11
3.12	Рисунок 12	12
3.13	Рисунок 13	12
3.14	Рисунок 14	13
3.15	Рисунок 15	13
3.16	Рисунок 16	14

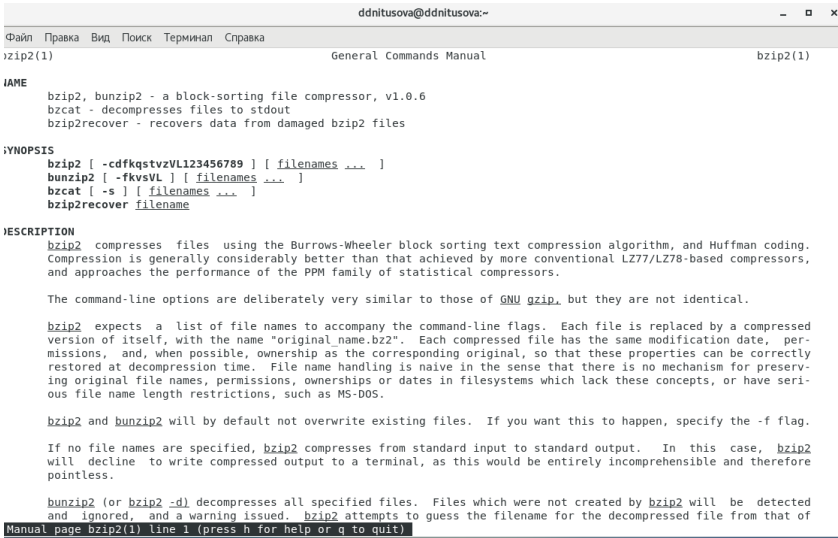
1 Цель работы

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

2 Задание

Написать несколько скриптов.

Синтаксис команды `bzip2` для разархивации/распаковки файла: `bunzip2` [опции] [архивы.bz2] (рис. -fig. 3.3).



```
ddnitusova@ddnitusova:~
Файл Правка Вид Поиск Терминал Справка
bzip2(1) General Commands Manual bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.6
  bzip2 - decompresses files to stdout
  bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
  bzip2 [ -cdfkqstvzVL123456789 ] [ 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, per-
  missions, 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 preserv-
  ing original file names, permissions, ownerships or dates in filesystems which lack these concepts, or have seri-
  ous file name length restrictions, such as MS-DOS.

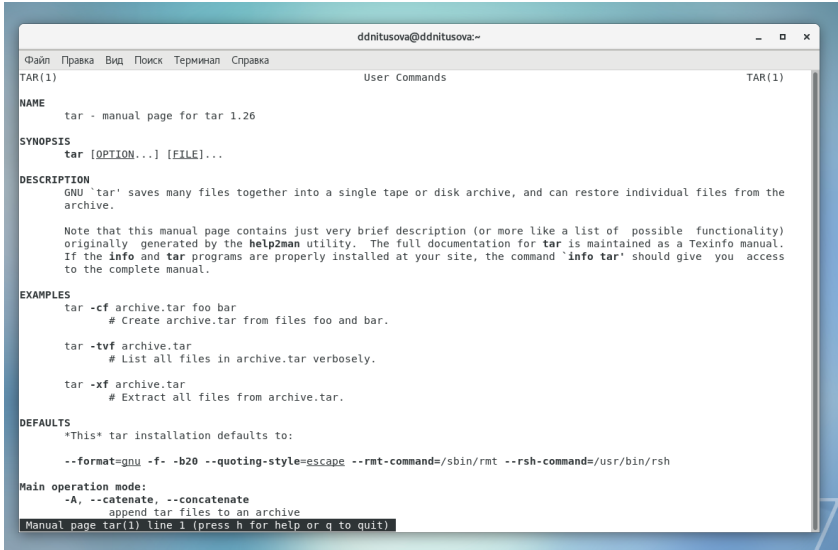
  bzip2 and bunzip2 will by default not overwrite existing files. If you want this to happen, specify the -f flag.

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

Рис. 3.3: Рисунок 3

Синтаксис команды `tar` для архивации файла: `tar` [опции] [архив.tar] [фай-
лы_для_архивации] Синтаксис команды `tar` для разархивации/распаковки файла:
`tar` [опции] [архив.tar] (рис. -fig. 3.4).



```
ddnitusova@ddnitusova:~
Файл Правка Вид Поиск Терминал Справка
TAR(1) User Commands TAR(1)

NAME
  tar - manual page for tar 1.26

SYNOPSIS
  tar [OPTION...] [FILE]...

DESCRIPTION
  GNU 'tar' saves many files together into a single tape or disk archive, and can restore individual files from the
  archive.

  Note that this manual page contains just very brief description (or more like a list of possible functionality)
  originally generated by the help2man utility. The full documentation for tar is maintained as a Texinfo manual.
  If the info and tar programs are properly installed at your site, the command 'info tar' should give you access
  to the complete manual.

EXAMPLES
  tar -cf archive.tar foo bar
  # Create archive.tar from files foo and bar.

  tar -tvf archive.tar
  # List all files in archive.tar verbosely.

  tar -xf archive.tar
  # Extract all files from archive.tar.

DEFAULTS
  *This* tar installation defaults to:

  --format=gnu --f -b20 --quoting-style=escape --rmt-command=/sbin/rmt --rsh-command=/usr/bin/rsh

Main operation mode:
  -A, --concatenate
  append tar files to an archive
  Manual page tar(1) line 1 (press h for help or q to quit)
```

Рис. 3.4: Рисунок 4

Далее я создала файл, в котором буду писать первый скрипт, и открыла его в ре-

дакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (рис. -fig. 3.5).

```
[ddnitusova@ddnitusova ~]$ touch backup.sh
[ddnitusova@ddnitusova ~]$ emacs &
[1] 3146
```

Рис. 3.5: Рисунок 5

После 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 вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. При написании скрипта использовала архиватор bzip2 (рис. -fig. 3.6).

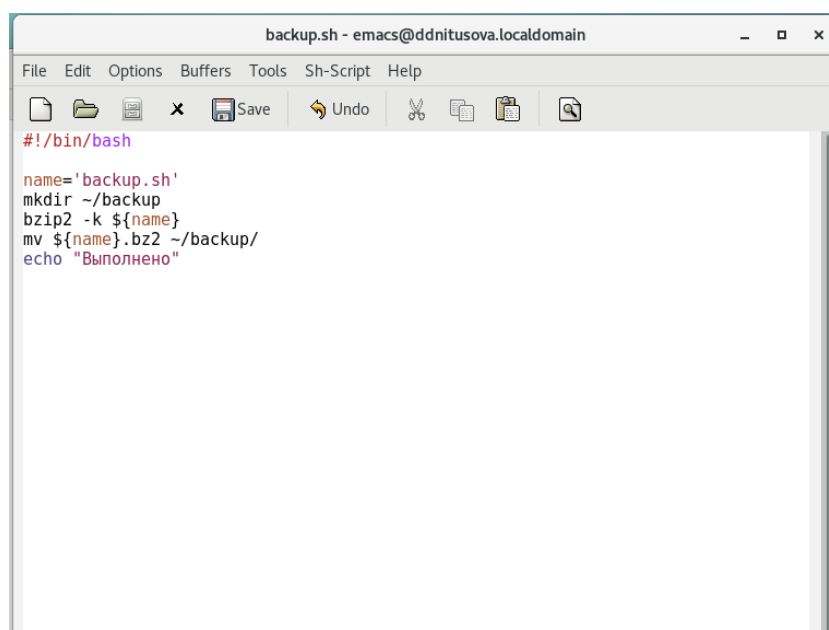


Рис. 3.6: Рисунок 6

Проверила работу скрипта (команда «./backup.sh»), предварительно 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»). Скрипт работает корректно (рис. -fig. 3.7).

```

[ddnitusova@ddnitusova ~]$ ls
abc      backup.sh~  lab03      lab07.sh~  lab09.4.txt  my_os      reports    work      Загрузки      Рабочий стол
abcl     conf.txt   lab05      lab09.1.txt  may          nmdir1     ski.places  work.     Изображения  Шаблоны
australia feathers  #lab07.sh# lab09.2.txt  mowdir       OS          text.txt   Видео      Музыка        Документы
backup.sh file.txt  lab07.sh   lab09.3.txt  monthly      play       usr        Документы  Общедоступные
[1]+  Done                  emacs
[ddnitusova@ddnitusova ~]$ chmod +x *.sh
[ddnitusova@ddnitusova ~]$ ./backup.sh
Выполнено
[ddnitusova@ddnitusova ~]$ cd backup/
[ddnitusova@ddnitusova backup]$ ls
backup.sh.bz2
[ddnitusova@ddnitusova backup]$ bunzip2 -c back.sh.bz2
bunzip2: Can't open input file back.sh.bz2: No such file or directory.
[ddnitusova@ddnitusova backup]$ bunzip2 -c backup.sh.bz2
#!/bin/bash

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

```

Рис. 3.7: Рисунок 7

Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и «emacs &») (рис. -fig. 3.8).

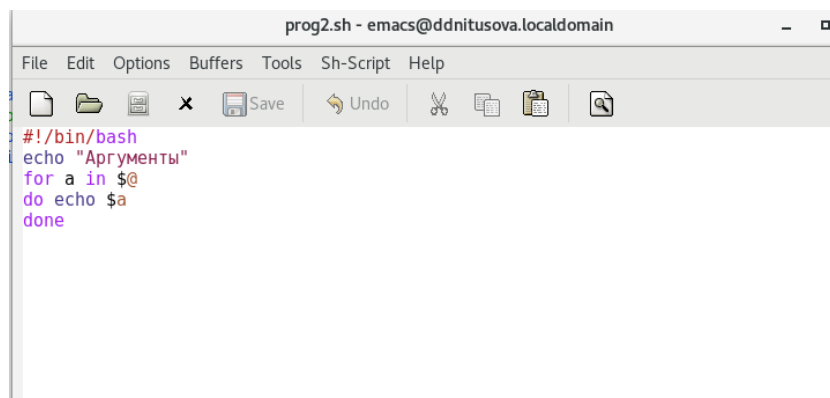
```

[ddnitusova@ddnitusova backup]$ touch prog2.sh
[ddnitusova@ddnitusova backup]$ emacs &
[1] 3354

```

Рис. 3.8: Рисунок 8

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. -fig. 3.9).



```

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

```

Рис. 3.9: Рисунок 9

Проверила работу написанного скрипта (команды «./prog2.sh 0 1 2 3 4» и

«./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Вводила аргументы, количество которых меньше 10 и больше 10. Скрипт работает корректно (рис. -fig. 3.10).

```
[ddnitusova@ddnitusova backup]$ chmod +x *.sh
[1]+  Done                  emacs
[ddnitusova@ddnitusova backup]$ ls
backup.sh.bz2 prog2.sh
[ddnitusova@ddnitusova backup]$ cd ..
[ddnitusova@ddnitusova ~]$ chmod +x *.sh
[ddnitusova@ddnitusova ~]$ ls
abc      backup.sh  file.txt  lab07.sh  lab09.3.txt  monthly  play      text.txt  Видео  Музыка
abc1     backup.sh- lab03     lab07.sh- lab09.4.txt  my_os    prog2.sh  usr      Документы  Общедоступные
australia conf.txt  lab05     lab09.1.txt  may          nemdir1  reports   work      Загрузки  Рабочий стол
backup   feathers  #lab07.sh# lab09.2.txt  newdir       05      ski.places work.     Изображения  Шаблоны
[ddnitusova@ddnitusova ~]$ ./prog2.sh 1 2 3 4
Аргументы
1
2
3
4
[ddnitusova@ddnitusova ~]$ ./prog2.sh 1 2 3 4 5 6 7 8 9 10
Аргументы
1
2
3
4
5
6
7
8
9
10
[ddnitusova@ddnitusova ~]$
```

Рис. 3.10: Рисунок 10

Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch progl.sh» и «emacs &») (рис. -fig. 3.11).

```
[ddnitusova@ddnitusova ~]$ touch progl.sh
[2]+  Done                  emacs
[ddnitusova@ddnitusova ~]$ emacs &
[1] 3504
```

Рис. 3.11: Рисунок 11

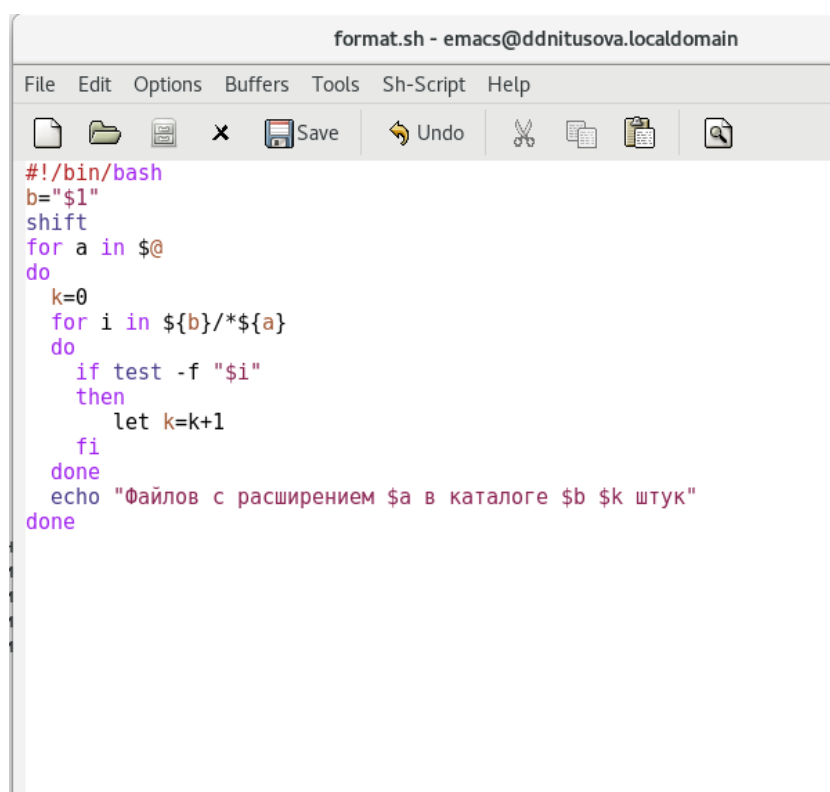
Написала командный файл – аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога (рис. -fig. 3.12).

открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (рис. -fig. 3.14).

```
ddnitusova@ddnitusova ~]$ touch format.sh
ddnitusova@ddnitusova ~]$ emacs &
[1] 3631
```

Рис. 3.14: Рисунок 14

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. -fig. 3.15).



```
format.sh - emacs@ddnitusova.localdomain
File Edit Options Buffers Tools Sh-Script Help
[Icons: File, Folder, Print, X, Save, Undo, Cut, Copy, Paste, Find]
#!/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 "Файлов с расширением $a в каталоге $b $k штук"
done
```

Рис. 3.15: Рисунок 15

Проверила 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 17). Скрипт работает корректно (рис. 3.16).

```
[ddnitusova@ddnitusova ~]$ chmod +x *.sh
[1]+  Done                  emacs
[ddnitusova@ddnitusova ~]$ ls
abc      backup.sh~  format.sh~  lab07.sh~  may      OS      reports  work.      Музыка
abc1     conf.txt    lab03       lab09.1.txt mewdir   play    ski.places Видео      Общедоступные
australia feathers  lab05       lab09.2.txt monthly  prog2.sh text.txt Документы  Рабочий стол
backup   file.txt    #lab07.sh#  lab09.3.txt my_os    progl.s.sh usr       Загрузки  Шаблоны
backup.sh format.sh  lab07.sh    lab09.4.txt nemdir1  progl.s.sh~ work      Изображения
[ddnitusova@ddnitusova ~]$ ./format.sh ~ txt sh
Файлов с расширением txt в каталоге /home/ddnitusova 7 штук
Файлов с расширением sh в каталоге /home/ddnitusova 5 штук
[ddnitusova@ddnitusova ~]$
```

Рис. 3.16: Рисунок 16

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 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/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` 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, 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,

заклѹчѣнных 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`.

5 Выводы

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