

Операционные системы

Лабораторная работа №11

Крупенникова Виктория Александровна

Содержание

1	Цель работы	3
2	Задание	4
3	Выполнение лабораторной работы	5
4	Контрольные вопросы:	13
5	Выводы	20

1 Цель работы

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

2 Задание

1. Ознакомиться с теоретическим материалом.
2. Изучить основы программирования в оболочке ОС UNIX/Linux.
3. Выполнить упражнения.
4. Ответить на контрольные вопросы.

3 Выполнение лабораторной работы

- 1) Для начала я изучила команды архивации, используя команды «man zip», «man bzip2», «man tar»

```
vakrupennikova@dk4n60 ~ $ man zip
vakrupennikova@dk4n60 ~ $ man bzip2
vakrupennikova@dk4n60 ~ $ man tar
```

Figure 3.1: Команды архивации

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]

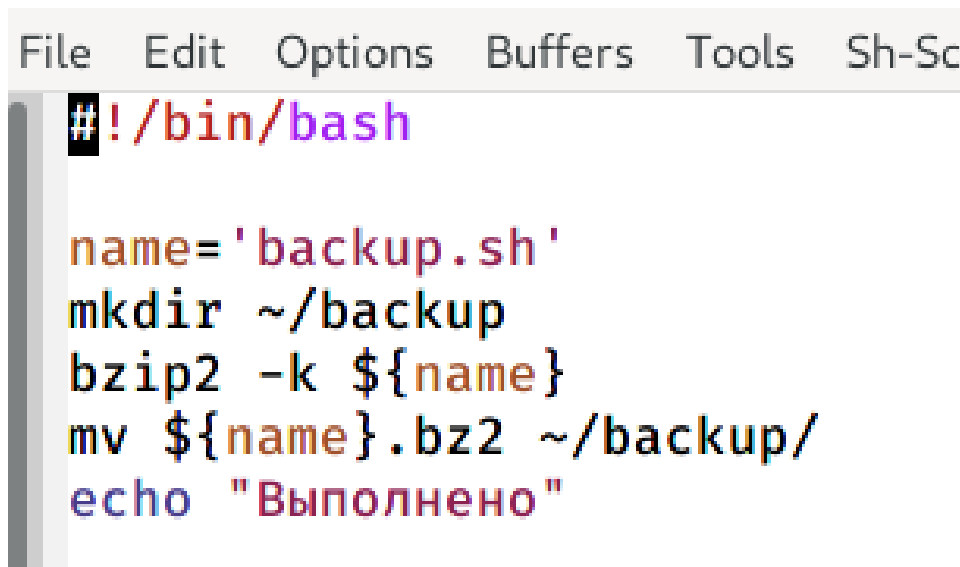
```
ZIP(1L)
NAME
    zip - package and compress (archive) files
SYNOPSIS
    zip [-sabcdefghjlmnpqrsStuvwxXyYz1234] [-i longname ...] [-b path] [-n suffixes] [-t date] [-vt date] [zipfile file ...] [-vi list]
DESCRIPTION
    zip is a compression and file packaging utility for Unix, VMS, MSDOS, OS/2, Windows NT/9x/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 archives produced by PKZIP (supporting most
    PKZIP features up to PKZIP version 4.0), and PKZIP and PKUNZIP can work with archives produced by zip (with some exceptions, notably streamed ar-
    chives, 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 3.0 cannot extract files produced by
    PKZIP 2.04 or zip 3.0. You must use PKUNZIP 2.04g or unzip 3.0a1 (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 created (if the resulting archive will exceed Zip64), the size of the archive will exceed 4 GB, or when the number of entries in
    the archive will exceed about 4294967295. Files in this sort of archive stream from standard input or the zip or bzip2 archives are not known to of-
```

Figure 3.2: Синтаксис команды zip

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

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

должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. При написании скрипта использовала архиватор bzip2.

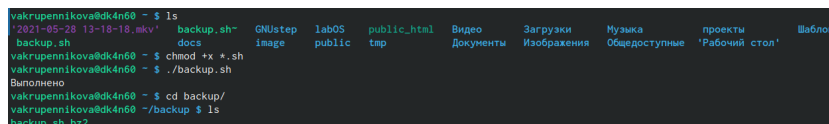


```
File Edit Options Buffers Tools Sh-Sc
#!/bin/bash

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

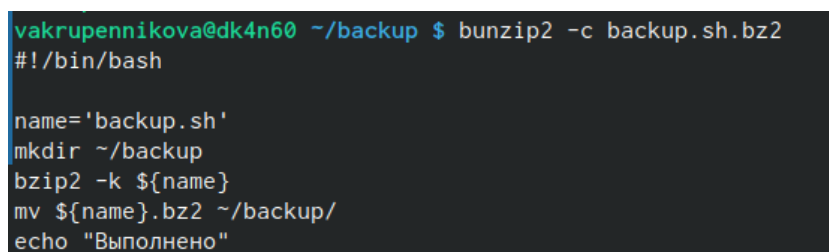
Figure 3.6: Первый скрипт

- 4) 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.



```
vakrupennikova@dk4n60 ~ $ ls
'2021-05-28 13-18-18.mkv'  backup.sh~  GNUstep  labOS  public.html  Видео  Загрузки  Музыка  проекты  Шаблоны
backup.sh                  docs        image     public  tmp          Документы  Изображения  Общедоступные  'Рабочий стол'
vakrupennikova@dk4n60 ~ $ chmod +x *.sh
vakrupennikova@dk4n60 ~ $ ./backup.sh
Выполнено
vakrupennikova@dk4n60 ~ $ cd backup/
vakrupennikova@dk4n60 ~/backup $ ls
backup.sh.bz2
```

Figure 3.7: Проверка работы скрипта



```
vakrupennikova@dk4n60 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

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

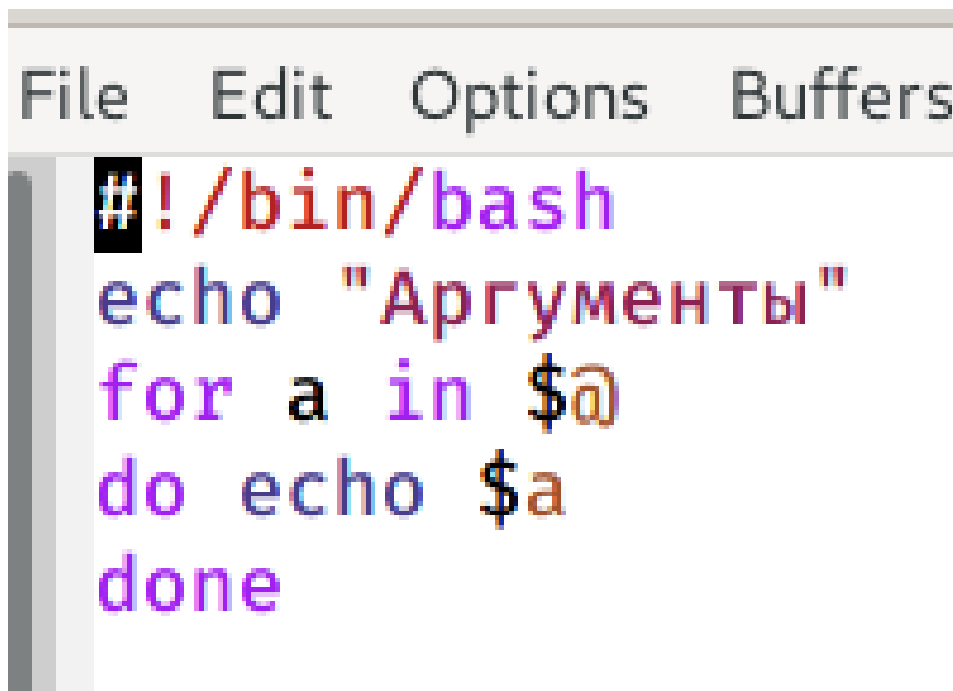
Figure 3.8: Вывод скрипта

- 5) Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touchprog2.sh» и «emacs&»)

```
vakrupennikova@dk4n60 ~ $ touch prog2.sh
vakrupennikova@dk4n60 ~ $ emacs &
```

Figure 3.9: Создание файла

- 6) 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.



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

Figure 3.10: Второй скрипт

- 7) Проверила работу написанного скрипта (команды «./prog2.sh0 1 2 3 4» и «./prog2.sh0 1 2 3 45 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.

```
vakrupennikova@dk4n60 ~ $ chmod +x *.sh
vakrupennikova@dk4n60 ~ $ ls
2021-05-28 13-18-18.mkv  backup.sh  docs  image  prog2.sh  public  tmp  Документы  Изображения  Общедоступные  Рабочий стол
backup  backup.sh  GMLstep  lab05  prog2.sh  public_html  Видео  Загрузки  Музыка  проекта  Шаблоны
vakrupennikova@dk4n60 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
vakrupennikova@dk4n60 ~ $ ./prog2.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
```

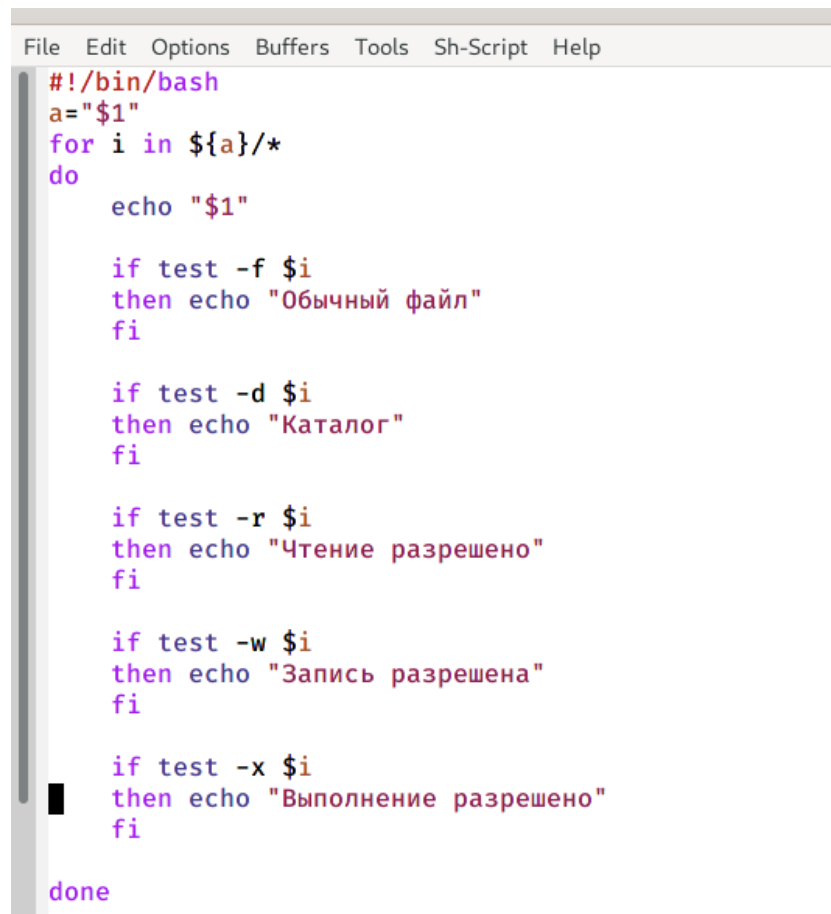
Figure 3.11: Проверка работы скрипта

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

```
vakrupennikova@dk4n60 ~ $ touch progl.sh
vakrupennikova@dk4n60 ~ $ emacs &
```

Figure 3.12: Создание файла

- 9) 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

The image shows a screenshot of a text editor window titled 'Sh-Script'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The script content is as follows:

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

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

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

Figure 3.13: Третий скрипт

- 10) Далее проверила работу скрипта (команда «./progl.sh~»), предварительно добавив для него право на выполнение (команда «chmod+x*.sh»). Скрипт работает корректно.

```
vakrupennikova@dk4n60 ~ $ chmod +x *.sh
[1] Завершён emacs
[2] Завершён emacs
vakrupennikova@dk4n60 ~ $ ls
'2021-05-28 13-18-18.mkv' backup.sh- image prog2.sh- public Видео Изображения проекты
backup.sh docs lab05 prog1s.sh public_html Документы Музыка 'Рабочий стол'
'2021-05-28 13-18-18.mkv' prog2.sh prog1s.sh tmp Загрузки Общедоступные Шаблоны
vakrupennikova@dk4n60 ~ $ ./prog1s.sh
./afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova
./prog1s.sh: строка 7: test: /afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova/2021-05-28: ожидается бинарный оператор
./prog1s.sh: строка 11: test: /afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova/2021-05-28: ожидается бинарный оператор
./prog1s.sh: строка 15: test: /afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova/2021-05-28: ожидается бинарный оператор
./prog1s.sh: строка 19: test: /afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova/2021-05-28: ожидается бинарный оператор
./prog1s.sh: строка 23: test: /afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova/2021-05-28: ожидается бинарный оператор
./afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
./afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
./afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
./afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
./afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
./afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
./afs/.dk.sci.pfu.edu.ru/home/v/a/vakrupennikova
```

Figure 3.14: Проверка работы скрипта

- 11) Для четвертого скрипта также создала файл (команда «touchformat.sh»)и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs&»)

```
vakrupennikova@dk4n60 ~ $ touch format.sh
vakrupennikova@dk4n60 ~ $ emacs &
```

Figure 3.15: Создание файла

- 12) Написала командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdfи т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки

```
File Edit Options Buffers Tools Sh-Script Help
#!/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 расширением $a"
done
```

Figure 3.16: Четвертый скрипт

- 13) Проверила работу написанного скрипта (команда «./format.sh~ pdfshtxt doc»), предварительно добавив для него право на выполнение (команда «chmod+x*.sh»), а также создав дополнительные файлы с разными расширениями (команда «touchfile.pdf file1.doc file2.doc»). Скрипт работает корректно.

```
vakrupennikova@dk4n68 ~$ chmod +x *.sh
vakrupennikova@dk4n68 ~$ touch file.pdf file1.doc file2.doc
vakrupennikova@dk4n68 ~$ ls
(2021-05-28 13:18:18,skv) backup.sh~ file2.doc~ format.sh~ labOS~ prog1s.sh~ public_html~ Документы~ Музыка~ 'Рабочий стол'
backup file1.doc~ file.pdf~ GNUstep~ prog2.sh~ prog1s.sh~ public Загрузки~ Общедоступные~ Временны
backup.sh~ file1.doc~ format.sh~ image~ prog2.sh~ public Видео~ Изображения~ проекты
vakrupennikova@dk4n68 ~$ ./format.sh ~ pdf sh txt doc
1 файлов содержится в каталоге /afs/dk.sci.pfu.edu.ru/home/v/a/vakrupennikova с расширением pdf
4 файлов содержится в каталоге /afs/dk.sci.pfu.edu.ru/home/v/a/vakrupennikova с расширением sh
0 файлов содержится в каталоге /afs/dk.sci.pfu.edu.ru/home/v/a/vakrupennikova с расширением txt
2 файлов содержится в каталоге /afs/dk.sci.pfu.edu.ru/home/v/a/vakrupennikova с расширением doc
vakrupennikova@dk4n68 ~$
```

Figure 3.17: Проверка работы скрипта

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 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 `$`. 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 отобразить 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`.

5 Выводы

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