

Отчёт по лабораторной работе №10

дисциплина: Операционные системы

Максим Александрович Мишонков

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Выводы	22

Список иллюстраций

4.1	Создание файла	7
4.2	Скрипт №1	8
4.3	Проверка работы скрипта	8
4.4	Проверка работы скрипта	9
4.5	Создание файла	9
4.6	Скрипт №2	9
4.7	Проверка работы скрипта	10
4.8	Проверка работы скрипта	10
4.9	Создание файла	11
4.10	Скрипт №3	11
4.11	Проверка работы скрипта	12
4.12	Проверка работы скрипта	12
4.13	Проверка работы скрипта	13
4.14	Проверка работы скрипта	14
4.15	Проверка работы скрипта	14
4.16	Создание файла	14
4.17	Скрипт №3	15
4.18	Проверка работы скрипта	15

1 Цель работы

Целью данной лабораторной работы является изучение основ программирования в оболочке ОС UNIX/Linux, приобретение навыков написания небольших командных файлов.

2 Задание

Изучить основы программирования в оболочке ОС UNIX/Linux, приобрести навыки написания небольших командных файлов.

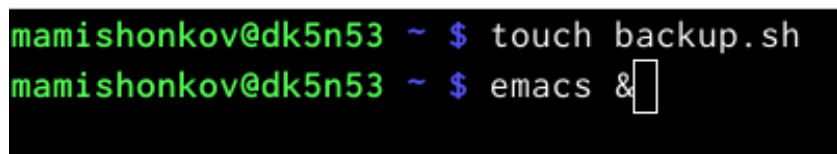
3 Теоретическое введение

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 - стандартная командная оболочка, содержащая базовый, но при этом полный набор функций.
- С-оболочка - надстройка над оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд.
- оболочка Корна напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна.
- BASH в основе своей совмещает свойства оболочек С и Корна.
- POSIX - набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

4 Выполнение лабораторной работы

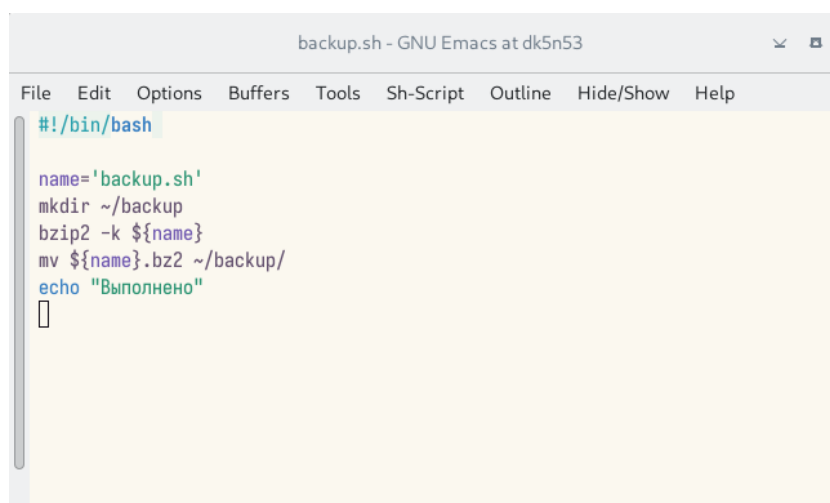
1. Создал файл, в котором буду писать первый скрипт, и открыл его в редакторе emacs, используя клавиши “Ctrl-x” и “Ctrl-f”. (рис. [4.2])



```
mamishonkov@dk5n53 ~ $ touch backup.sh  
mamishonkov@dk5n53 ~ $ emacs &
```

Рис. 4.1: Создание файла

2. Написал скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию back up в домашнем каталоге. При написании скрипта использовал архиватор bzip2. (рис. [??])

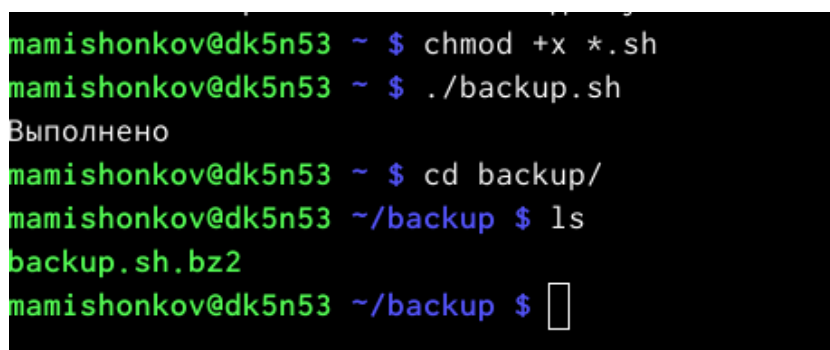


```
backup.sh - GNU Emacs at dk5n53
File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help
#!/bin/bash

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

Рис. 4.2: Скрипт №1

3. Проверил работу скрипта, предварительно добавив для него право на выполнение. Проверил, появился ли каталог backup/, перейдя в него, посмотрел его содержимое и содержимое архива. Скрипт работает корректно. (рис. [4.3],[4.4])



```
mamishonkov@dk5n53 ~ $ chmod +x *.sh
mamishonkov@dk5n53 ~ $ ./backup.sh
Выполнено
mamishonkov@dk5n53 ~ $ cd backup/
mamishonkov@dk5n53 ~/backup $ ls
backup.sh.bz2
mamishonkov@dk5n53 ~/backup $
```

Рис. 4.3: Проверка работы скрипта


```

mamishonkov@dk5n53 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
mamishonkov@dk5n53 ~/backup $ 

```

Рис. 4.4: Проверка работы скрипта

4. Создал файл, в котором буду писать второй скрипт, и открыл его в редакторе emacs. (рис. [4.5])

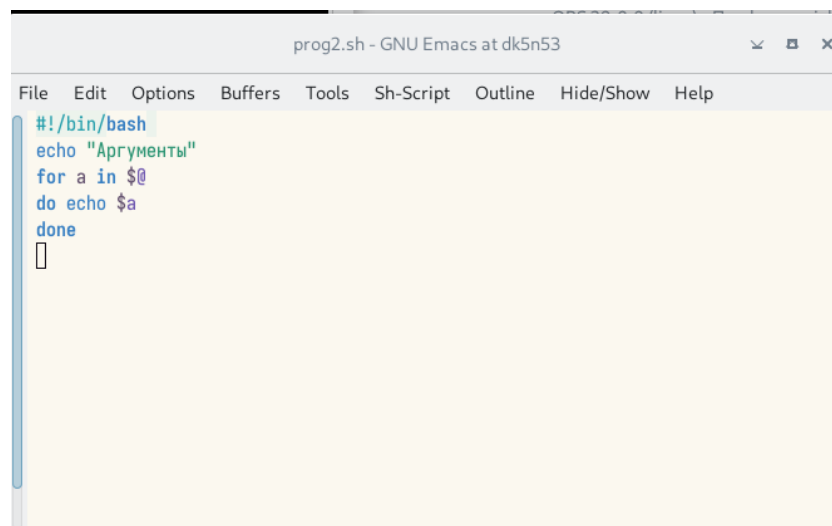
```

mamishonkov@dk5n53 ~ $ touch prog2.sh
mamishonkov@dk5n53 ~ $ emacs &

```

Рис. 4.5: Создание файла

5. Написал скрипт, обрабатывающий любое произвольное число аргументов командной строки, в том числе превышающее десять. (рис. [4.6])



```

prog2.sh - GNU Emacs at dk5n53
File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help
#!/bin/bash
echo "Аргументы"
for a in $@
do echo $a
done

```

Рис. 4.6: Скрипт №2

6. Проверил работу скрипта, предварительно добавив для него право на выполнение. Вывел аргументы, количество которых меньше десяти и больше десяти. Скрипт работает корректно. (рис. [4.7],[4.8])

```
namishonkov@dk5n53 ~ $ chmod +x *.sh
namishonkov@dk5n53 ~ $ ls
backup      conf.txt    prog2.sh    tmp         Изображения
backup.sh   file.txt    prog2.sh~   work        Общедоступные
backup.sh~  GNUstep    public      Документы   'Рабочий стол'
bin         lab07.sh~  public_html Загрузки    Шаблоны
namishonkov@dk5n53 ~ $
```

Рис. 4.7: Проверка работы скрипта

```
namishonkov@dk5n53 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
namishonkov@dk5n53 ~ $ ./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
```

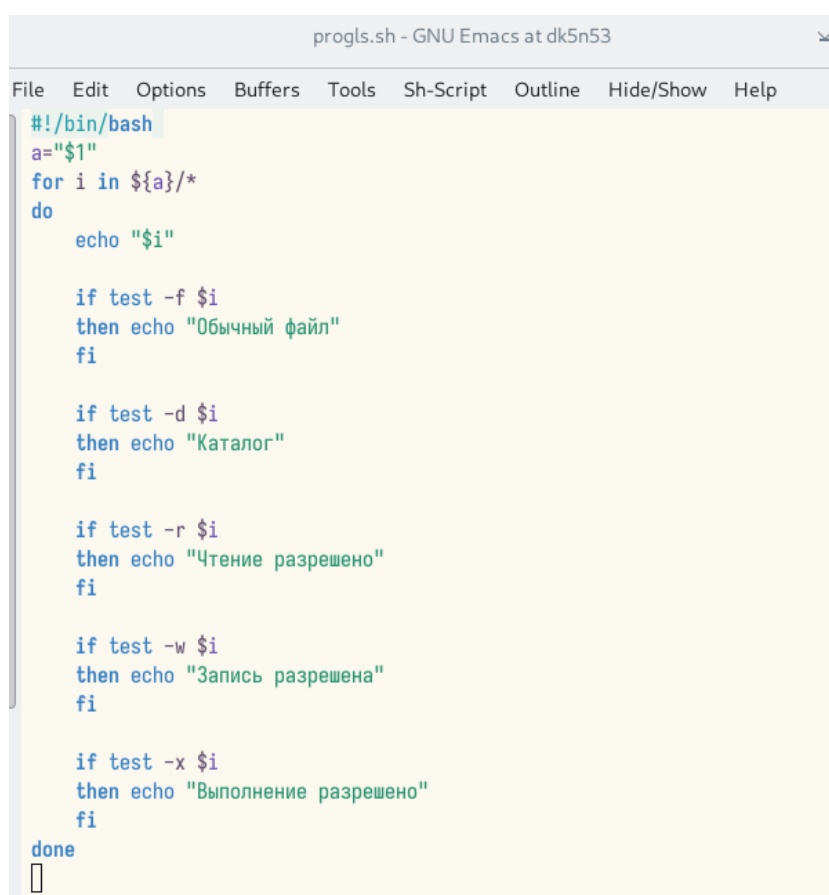
Рис. 4.8: Проверка работы скрипта

7. Создал файл, в котором буду писать третий скрипт, и открыл его в редакторе emacs. (рис. [4.9])

```
mamishonkov@dk5n53 ~ $ touch progl5.sh
mamishonkov@dk5n53 ~ $ emacs &
```

Рис. 4.9: Создание файла

5. Написал командный файл - аналог команды ls (без использования самой команды и команды dir). он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога. (рис. [4.10])



```
progl5.sh - GNU Emacs at dk5n53
File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help
#!/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 "Чтение разрешено"
    fi

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

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

Рис. 4.10: Скрипт №3

6. Проверил работу скрипта, предварительно добавив для него право на выполнение. Скрипт работает корректно. (рис. [4.11],[4.12],[4.13],[4.14],[4.15])

```

mamishonkov@dk5n53 ~ $ chmod +x *.sh
chmod: неверный режим: «+x.»
По команде «chmod --help» можно получить дополнительную информацию.
[1]+  Завершён      emacs
mamishonkov@dk5n53 ~ $ chmod +x *.sh
mamishonkov@dk5n53 ~ $ ls
backup      file.txt    proglis.sh  work        'Рабочий стол'
backup.sh   GNUstep    proglis.sh~  Документы   Шаблоны
backup.sh~  lab07.sh~  public      Загрузки
bin         prog2.sh   public_html  Изображения
conf.txt    prog2.sh~  tmp         Общедоступные
mamishonkov@dk5n53 ~ $

```

Рис. 4.11: Проверка работы скрипта

```

mamishonkov@dk5n53 ~ $ ./proglis.sh ~
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/backup.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/backup.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/bin
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/conf.txt
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/file.txt
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/GNUstep
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/lab07.sh~

```

Рис. 4.12: Проверка работы скрипта

```
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/prog2.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/prog2.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/progls.sh
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/progls.sh~
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/public
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/public_html
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/tmp
```

Рис. 4.13: Проверка работы скрипта

```

Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/work
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Документы
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Загрузки
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Изображения
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Общедоступные

```

Рис. 4.14: Проверка работы скрипта

```

Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Рабочий стол
./progl.s.sh: строка 7: test: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Рабочий: ожидается бинарный оператор
./progl.s.sh: строка 11: test: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Рабочий: ожидается бинарный оператор
./progl.s.sh: строка 15: test: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Рабочий: ожидается бинарный оператор
./progl.s.sh: строка 19: test: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Рабочий: ожидается бинарный оператор
./progl.s.sh: строка 23: test: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Рабочий: ожидается бинарный оператор
/afs/.dk.sci.pfu.edu.ru/home/m/a/mamishonkov/Шаблоны
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено

```

Рис. 4.15: Проверка работы скрипта

7. Создал файл, в котором буду писать четвёртый скрипт, и открыл его в редакторе emacs. (рис. [4.16])

```

mamishonkov@dk5n53 ~ $ touch format.sh
mamishonkov@dk5n53 ~ $ emacs &

```

Рис. 4.16: Создание файла

5. Написал командный файл, который получает в качестве аргумента командной строки формат файла и вычиляет количество таких файлов в указанной директории. Путь к директории так же передаётся в виде аргумента командной строки. (рис. [4.17])

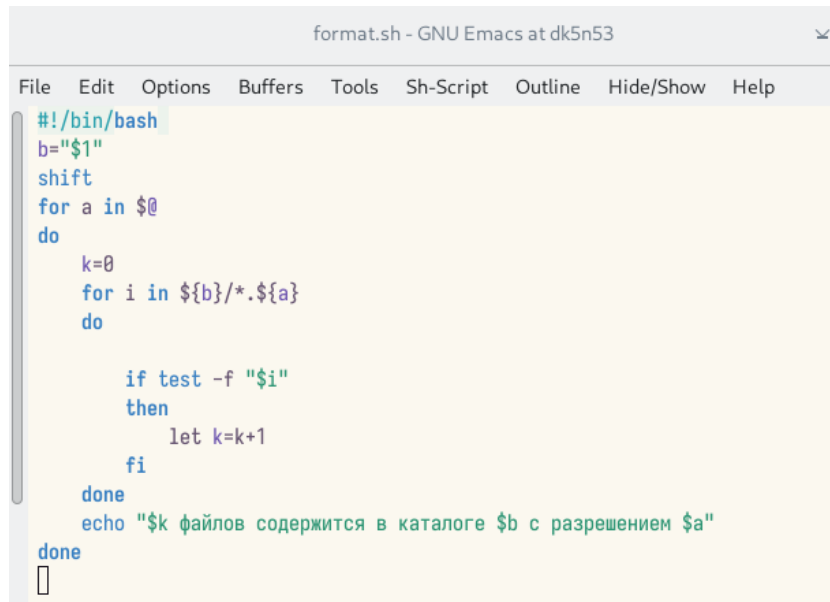


Рис. 4.17: Скрипт №3

6. Проверил работу скрипта, предварительно добавив для него право на выполнение и создав дополнительные файлы с разными расширениями. Скрипт работает корректно. (рис. [4.18])

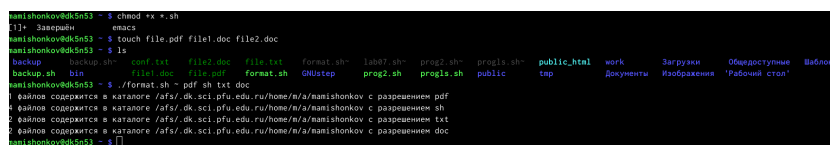


Рис. 4.18: Проверка работы скрипта

Ответы на 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 (BourneShell 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 BourneAgainShell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании FreeSoftwareFoundation).

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

`{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` 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, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить 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 зна-

чением 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 кавычки. 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`.

5 Выводы

В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и приобрёл навыки написания небольших командных файлов.