

Отчёт по лабораторной работе №11

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Матвеева Анастасия Сергеевна

Содержание

1	Цель работы	4
2	Задачи лабораторной работы	5
3	Выполнение лабораторной работы	6
4	Контрольные вопросы	15
5	Выводы	21
6	Библиография	22

List of Figures

3.1	Команда <code>man</code>	6
3.2	Синтаксис команды <code>zip</code>	6
3.3	Синтаксис команды <code>bzip2</code>	7
3.4	Синтаксис команды <code>tar</code>	7
3.5	Создание файла и открытие <code>emacs</code>	8
3.6	Первый скрипт	8
3.7	Проверка работы скрипта	9
3.8	Проверка работы скрипта	9
3.9	Создание файла и открытие <code>emacs</code>	9
3.10	Второй скрипт	10
3.11	Проверка работы скрипта	10
3.12	Создание файла и открытие <code>emacs</code>	11
3.13	Третий скрипт	12
3.14	Проверка работы скрипта	13
3.15	Создание файла и открытие <code>emacs</code>	13
3.16	Четвертый скрипт	14
3.17	Проверка работы скрипта	14

1 Цель работы

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

2 Задачи лабораторной работы

Задачи:

1. Познакомиться с командными процессорами.
2. Изучить переменные, арифметические операторы в языке программирования `bash`.
3. Изучить операторы цикла `for`, `while` и `until`, оператор выбора `case`, условный оператор `if`.
4. В ходе работы написать 4 скрипта.

3 Выполнение лабораторной работы

1. Для начала я изучила команды архивации, используя команды «man zip», «man bzip2», «man tar» (рис. 3.1)

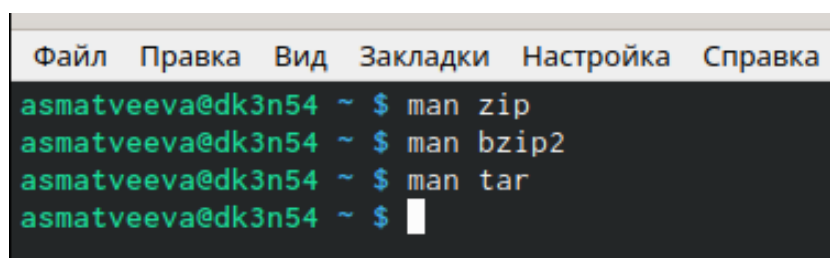


Figure 3.1: Команда man

Синтаксис команды zip для архивации файла (рис. 3.2):

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]

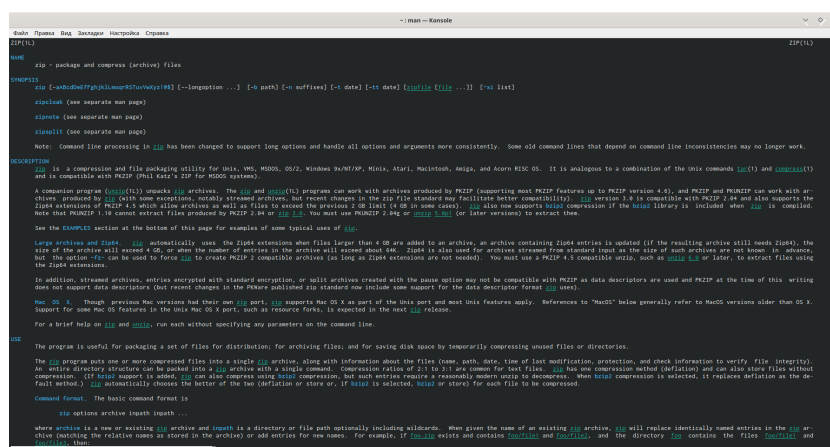


Figure 3.2: Синтаксис команды zip

bunzip2[опции] [архивы.bz2]

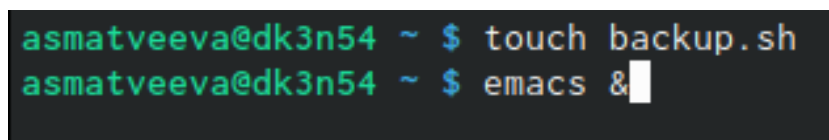


tar[опции][архив.tar]



2. Далее я создала файл, в котором буду писать первый скрипт, и открыла его

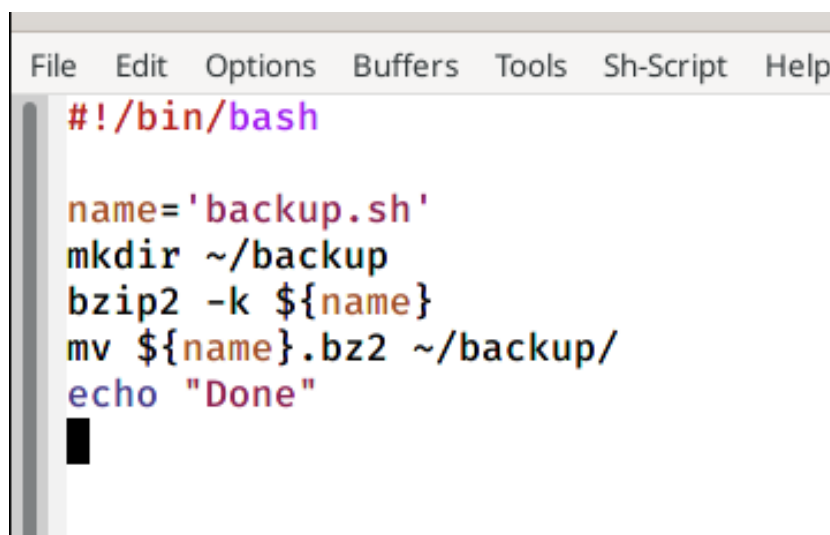
в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &»). (рис. 3.5)



```
asmatveeva@dk3n54 ~ $ touch backup.sh
asmatveeva@dk3n54 ~ $ emacs &
```

Figure 3.5: Создание файла и открытие emacs

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. 3.6)



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

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

Figure 3.6: Первый скрипт

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. 3.7) (рис. 3.8)


```

asmatveeva@dk3n54 ~ $ ls
abc1      backup.sh~  example2.txt  file.txt     lab03b      lab07.sh~   lab7.asm
asdfg.asm conf.txt    example3.txt  GNUstep     lab05.asm   lab2.asm    lp.asm
australia equiment    example4.txt  lab02       lab07.asm   lab5.asm    Makefile-1
backup.sh  example1.txt feathers      lab03a      lab07.sh    lab6.asm    may
asmatveeva@dk3n54 ~ $ chmod +x *.sh
asmatveeva@dk3n54 ~ $ ./backup.sh
Done
asmatveeva@dk3n54 ~ $ cd backup/
asmatveeva@dk3n54 ~/backup $ ls
backup.sh.bz2

```

Figure 3.7: Проверка работы скрипта

```

asmatveeva@dk3n54 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

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

```

Figure 3.8: Проверка работы скрипта

3. 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 &»). (рис. 3.9)

```

asmatveeva@dk3n54 ~ $ touch prog1.sh
asmatveeva@dk3n54 ~ $ emacs &
[1] 6835

```

Figure 3.9: Создание файла и открытие emacs

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. 3.10)

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
echo "Arguments"
for a in $@
do echo $a
done
```

Figure 3.10: Второй скрипт

Проверила работу написанного скрипта (команды «./prog2.sh 0 1 2 3 4 5» и «./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Вводила аргументы, количество которых меньше 10 и больше 10. Скрипт работает корректно. (рис. 3.11)

```
[1] 6835
asmatveeva@dk3n54 ~ $ chmod +x *.sh
asmatveeva@dk3n54 ~ $ ls
abcl backup.sh example1.txt feathers lab03a lab07.sh lab6.asm may Nastya prog1.sh
asdfg.asm backup.sh~ example2.txt file.txt lab03b lab07.sh~ lab7.asm monthly pfog.asm prog1.sh~
australia conf.txt example3.txt GNUstep lab05.asm lab2.asm lp.asm morefun01 pfog.lst prog.asm
backup equiment example4.txt lab02 lab07.asm lab5.asm Makefile-1 my_os play public
asmatveeva@dk3n54 ~ $ ./prog1.sh 0 1 2 3 4
Arguments
0
1
2
3
4
asmatveeva@dk3n54 ~ $ ./prog1.sh 0 1 2 3 4 5 6 7 8 9 10 11
Arguments
0
1
2
3
4
5
6
7
8
9
10
11
```

Figure 3.11: Проверка работы скрипта

4. Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prls.sh» и «emacs &») (рис. 3.12)

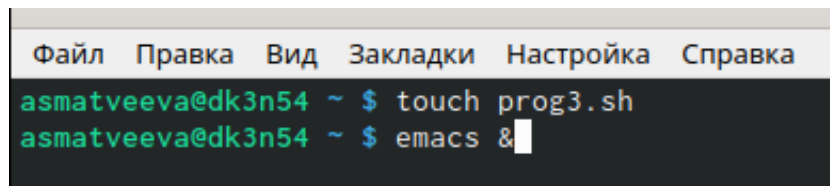
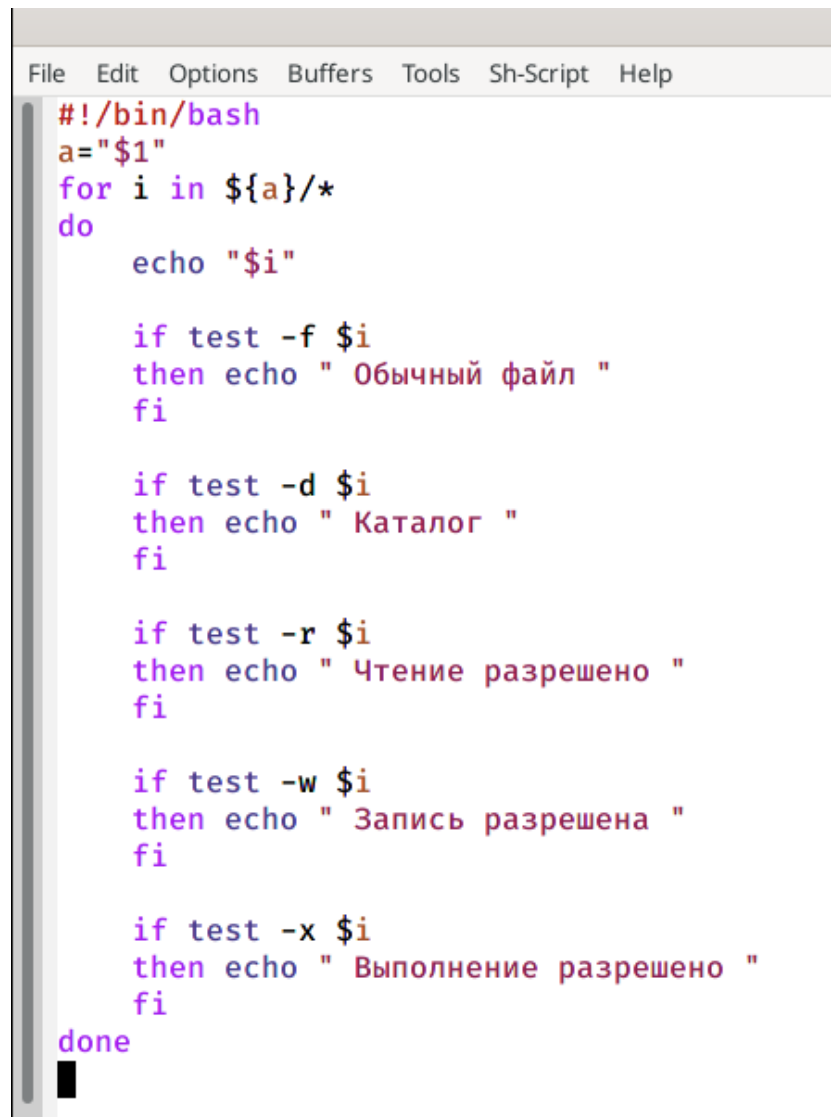
A terminal window with a menu bar containing 'Файл', 'Правка', 'Вид', 'Закладки', 'Настройка', and 'Справка'. The terminal shows two commands being executed: 'touch prog3.sh' and 'emacs &'. The prompt is 'asmatveeva@dk3n54 ~ \$'.

Figure 3.12: Создание файла и открытие emacs

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. 3.13)



```
File Edit Options Buffers Tools Sh-Script 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
```

Figure 3.13: Третий скрипт

Далее проверила работу скрипта (команда «./prls.sh~»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Скрипт работает корректно. (рис. 3.14)


```
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

Проверила работу написанного скрипта (команда «./format.sh ~ pdf jpg doc txt png»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Скрипт работает корректно. (рис. 3.17)

```
asmatveeva@dk3n54 ~ $ chmod +x *.sh
asmatveeva@dk3n54 ~ $ touch file01.pdf file02.doc file03.doc
asmatveeva@dk3n54 ~ $ ls
abcl      backup.sh~  example3.txt  file03.doc  lab02      lab07.sh     lab7.asm     morefun01  play
asdfg.asm  conf.txt    example4.txt  file.txt    lab03a     lab07.sh~    lp.asm       my_os      prog1.sh
australia  equipment   feathers      format.sh   lab03b     lab2.asm     Makefile-1   Nastya     prog1.sh~
backup     example1.txt file01.pdf    format.sh~  lab05.asm  lab5.asm     may          pfog.asm   prog3.sh
backup.sh  example2.txt file02.doc    GNUstep     lab07.asm  lab6.asm     monthly      pfog.lst   prog3.sh~
asmatveeva@dk3n54 ~ $ ./format.sh ~ pdf sh txt doc
1 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/s/asmatveeva с расширением pdf
5 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/s/asmatveeva с расширением sh
7 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/s/asmatveeva с расширением txt
2 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/a/s/asmatveeva с расширением doc
asmatveeva@dk3n54 ~ $
```

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:

1. оболочка Борна (Bourne shell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
2. C-оболочка (или csh) – надстройка на оболочке Борна, использующая подобный синтаксис команд с возможностью сохранения истории выполнения команд;
3. Оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
4. 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 `.`, «`mva file{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 `setc` 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, 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:

1. 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.
2. 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 >.
3. HOME: имя домашнего каталога пользователя. Если команда вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
4. IFS: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (newline).
5. 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 Youhavemail(у Вас есть почта).
6. TERM: тип используемого терминала.
7. 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 `unset` 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 окру-

жения. 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:

1. \$* –отображается вся командная строка или параметры оболочки;
2. \$? –код завершения последней выполненной команды;
3. \$\$ –уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
4. \$! –номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
5. \$--значение флагов командного процессора;

6. `${#}` –возвращает целое число –количество слов, которые были результатом `$`;
7. `${#name}` –возвращает целое значение длины строки в переменной `name`;
8. `${name[n]}` –обращение к `n`-му элементу массива;
9. `${name[*]}` –перечисляет все элементы массива, разделённые пробелом;
10. `${name[@]}` –то же самое, но позволяет учитывать символы пробелы в самих переменных;
11. `${name:-value}` –если значение переменной `name` не определено, то оно будет заменено на указанное `value`;
12. `${name:value}` –проверяется факт существования переменной;
13. `${name=value}` –если `name` не определено, то ему присваивается значение `value`;
14. `${name?value}` –останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке;
15. `${name+value}` –это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `value`;
16. `${name#pattern}` –представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`);
17. `${#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.

6 Библиография

1. https://esystem.rudn.ru/pluginfile.php/1142090/mod_resource/content/2/008-lab_shell_prog_1.pdf
2. Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: <http://lib.rudn.ru/MegaPro2/Download/MObject/6118>.
3. Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немнюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ЕТ 60)
4. Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ЕТ 50)