Лабораторная работа

Отчет по лабораторной работе

Хусяинова Адиля Фаритовна

Цель работы

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

Выполнение лабораторной работы

1. Изучим команды архивации с помощью команд man zip, man bzip2, man tar (рис.1)

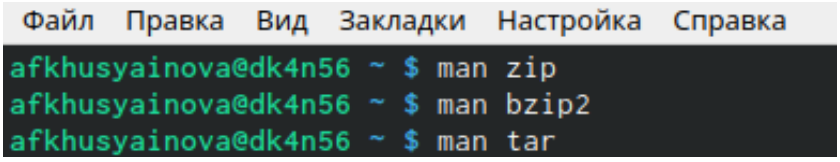

Рис. 0.1.: Команда man

- Произведем синтаксис команды zip для архивации файлов: zip [опции] [имя_файла.zip] [файлы или папки для архивации]. Синтаксис для разархивации файла: unzip [опции] [файл_архива.zip] [файлы] -x [исключить] -d [папки] (рис.2)

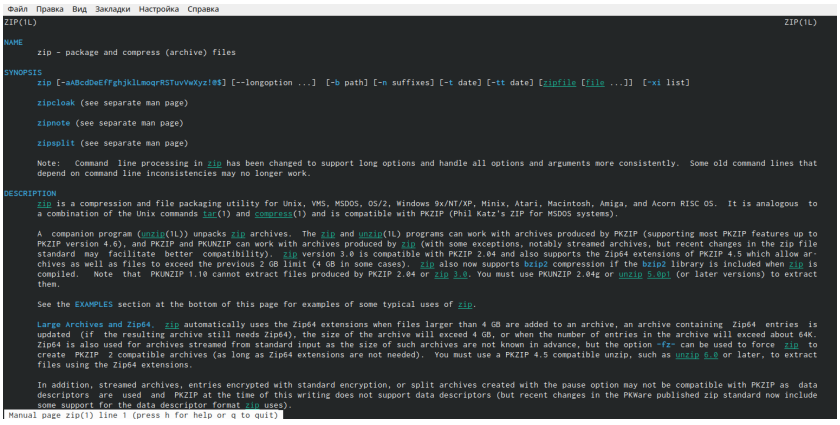

Рис. 0.2.: Архив zip

- Синтаксис команды bzip2 для архивации файлов: bzip2 [опции] [имя_файла]. Синтаксис для разархивации файла: bunzip2 [опции] [файл_архива.bz2] (рис.3)

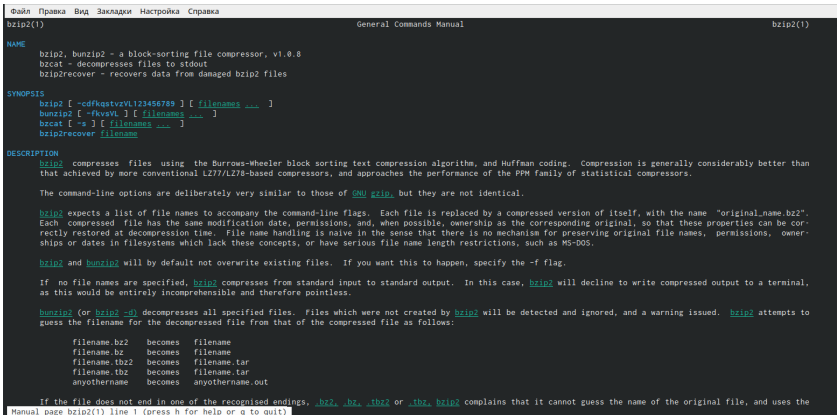

Рис. 0.3.: Архив bzip2

- Синтаксис команды tar для архивации файлов: tar [опции] [архив.tar]. Синтаксис для разархивации файла: tar [опции] [архив.tar] (рис.4)

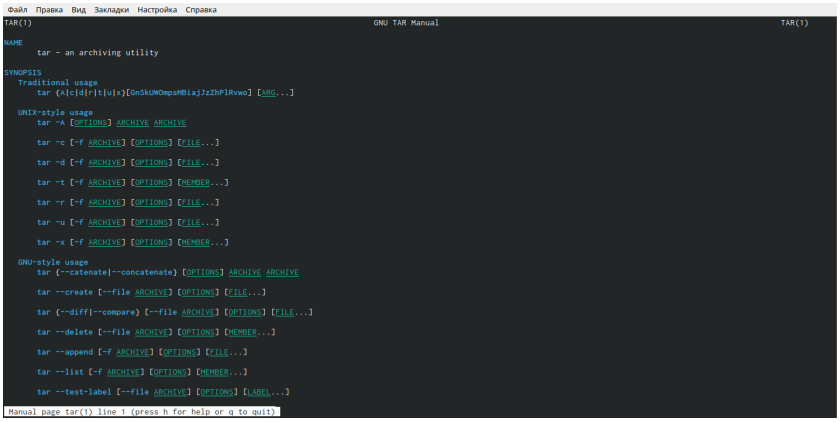

Рис. 0.4.: Архив tar

- Создаем файл backup.sh, в котором напишем первый скрипт, и откроем в редакторе emacs (рис.5)

```
afkhusyainova@dk4n56 ~ $ touch backup.sh
afkhusyainova@dk4n56 ~ $ emacs &
[1] 13656
```

Рис. 0.5.: Создание файла

- Напишем программу, которая при запуске будет делать резервную копию самого себя в другую директорию backup в нашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar (выбираем bzip2) (рис.6)

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

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

Рис. 0.6.: Программа

- Проверим работу скрпта (команда ./backup.sh), предварительно добавив для него права на выполнение (chmod +x *.sh). Перейдем в каталог backup и удостоверимся, что файл появился в этом каталоге, (рис.8) и просмотрим содержимое архива (команда bunzip2 - backup.sh.bz2) (рис.7-9)

```
afkhusyainova@dk4n56 ~ $ ls
 1.sh                       backup.sh    friend.cpp       games             lab07.sh     monthly.00   reports      tmp
'2022-04-28 10-49-58.mkv'   backup.sh~   friend.cpp.save  key               lab07.sh~    my_os        sci_plases   work
 abc1                       conf.txt     friend.sh        key.pub           lab4-1.asm   newfriend    ski_pases    Видео
 adilya                     feathers     fri.sh           khusyainova.cpp   may          public       ski_plases   Документы
 Adilya                     file.txt     fr.sh            lab01             monthly      public_html  text.txt     Загрузки
afkhusyainova@dk4n56 ~ $ chmod +x *.sh
afkhusyainova@dk4n56 ~ $ ./backup.sh
Выполнено
```

Рис. 0.7.: Права доступа

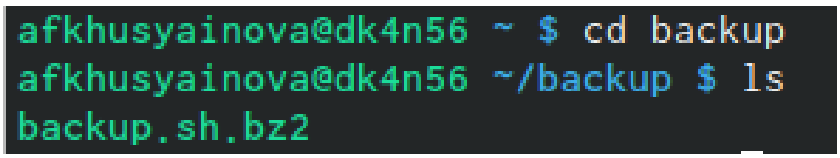

Рис. 0.8.: Просмотр каталога

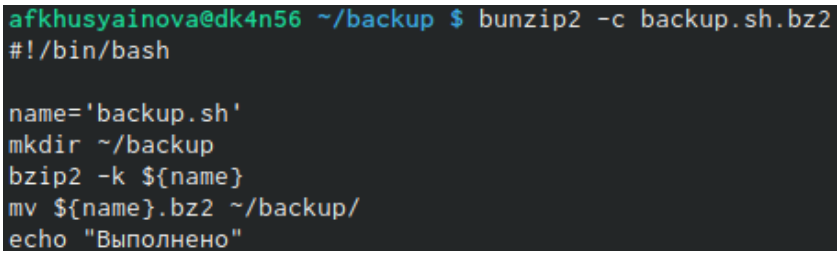

Рис. 0.9.: Просмотр содержимого

2. Создадим файл prog.sh, в котором напишем второй скрипт, и откроем его в emacs (рис.10)

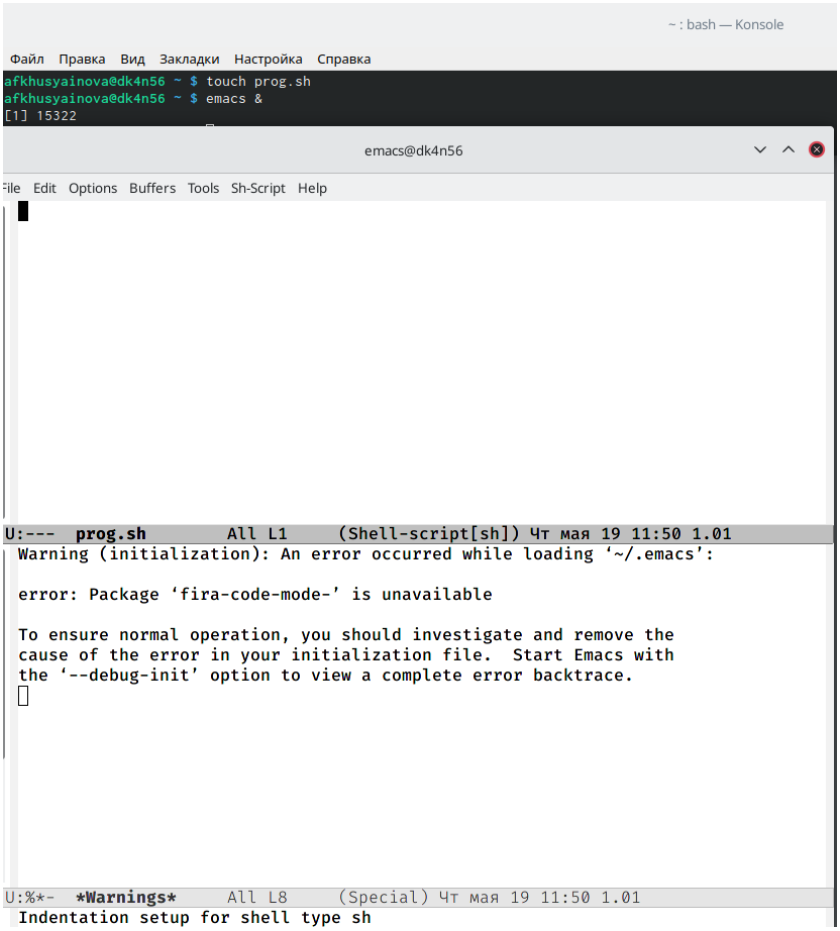

Рис. 0.10.: Создание файла

- 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.11)

Рис. 0.11.: Программа

- Проверим его работу, предварительно передав права доступа на выполнение, вводим разное количество аргументов меньше и больше 10 (рис.12-13)

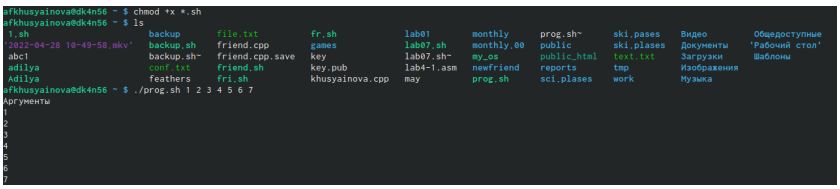

Рис. 0.12.: Выполнение

```
afkhusyainova@dk4n56 ~ $ ./prog.sh 1 2 3 2 6 15 17 8
Аргументы
1
2
3
2
6
15
17
8
```

Рис. 0.13.: Выполнение

3. Создадим файл progls.sh для третьего скрипта (команда touch progls.sh) и откроем его в emacs. Напишем командный файл — аналог команды ls (без использования самой этой команды и команды dir) . Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога (рис.14)

```
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 "Облачный файл"
    fi

    if test -d $i
    then echo ""
    fi

    if test -r $i
    then echo ""
    fi

    if test -w $i
    then echo ""
    fi

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

Рис. 0.14.: Код программы

- Проверим его работу, предварительно передав права доступа на выполнение (chmod +x *.sh) и командой ./progls.sh запустим его (рис.15)

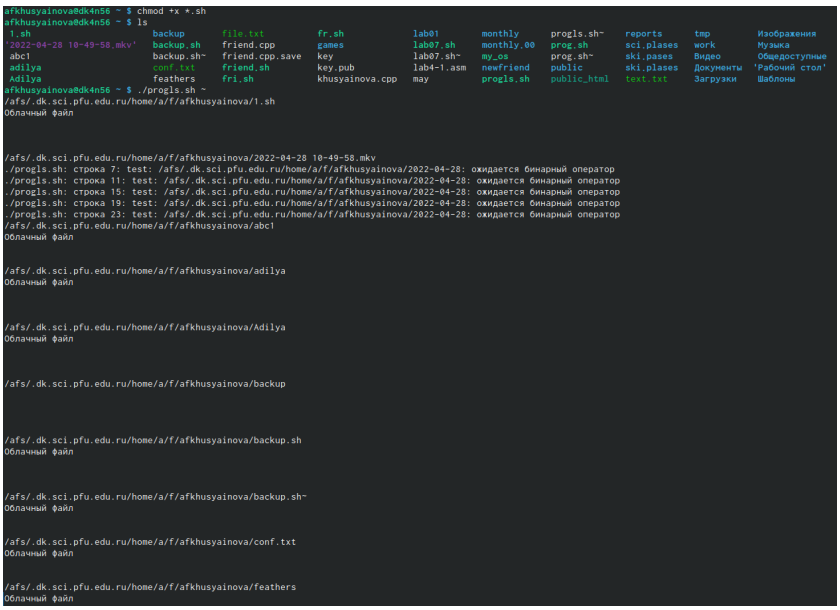

Рис. 0.15.: Запуск файла

4. Создадим файл format.sh для четвертого скрипта (команда touch format.sh) и откроем его в emacs (рис.16)

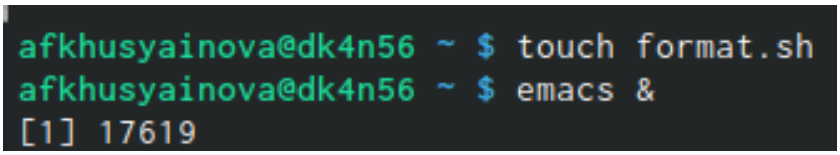

Рис. 0.16.: Создание файла

- Напишем 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.17)

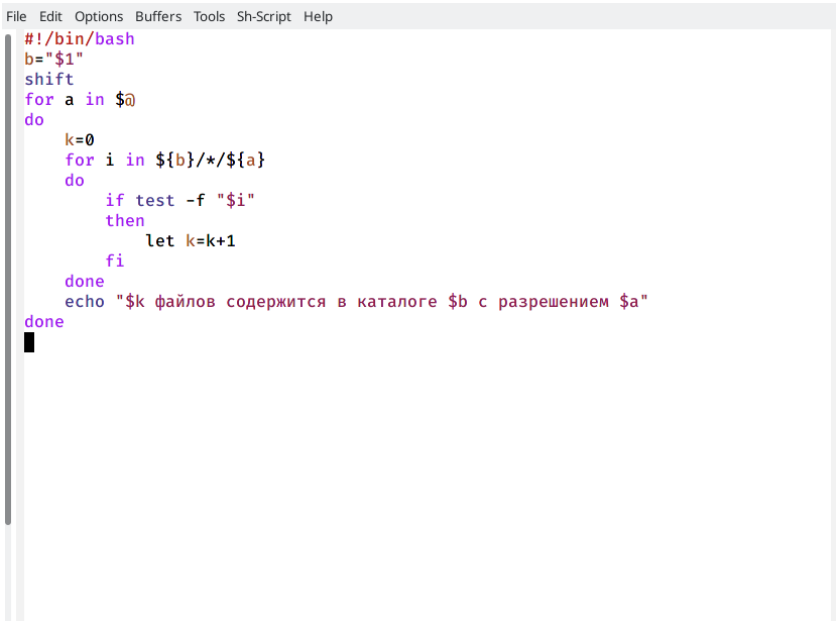

Рис. 0.17.: Код программы

- Проверим его работу, предварительно передав права доступа на выполнение (chmod +x *.sh) и командой ./format.sh ~ pdf txt docx запустим его (рис.18)

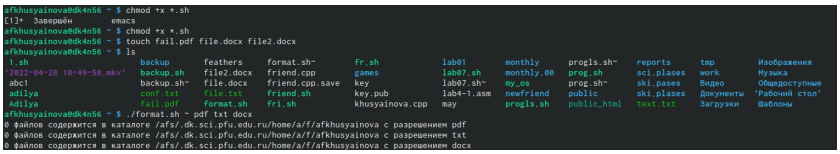

Рис. 0.18.: Запуск файла

Выводы

Я изучила основы программирования в оболочке ОС UNIX/Linux, научилась писать небольшие команды в оболочке Linux.

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 (Bourneshellили 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 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 Electricaland 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.
2. 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 {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 -Astates Delaware Michigan “New Jersey”». Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента.

3. 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 monи day будут считаны соответствующие значения, введённые с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её.
4. В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).
5. В (( ))можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать

результат.

6. 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 корневогоили 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 cdвводится 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 Youhavemail(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 осу-

ществлено 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 будтоон 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.

7. 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 unsetcфлагом -f.
8. 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).
9. Команду «set» можно использовать для вывода списка переменных окружения. В системах Ubuntu и Debia nкоманда «set» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка перемен-

ных 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.

10. 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.
11. 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 –количествослов, 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 суще-

ствования 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.