

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)

```
Файл  Правка  Вид  Закладки  Настройка  Справка
afkhusyainova@dk4n56 ~ $ man zip
afkhusyainova@dk4n56 ~ $ man bzip2
afkhusyainova@dk4n56 ~ $ man tar
```

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)

```
Файл  Правка  Вид  Закладки  Настройка  Справка
ZIP(1L)
NAME
zip - package and compress (archive) files
SYNOPSIS
zip [-aABcdDeFFghjklLmooqRSTuvVwXyz{##}] [--longoption ...] [-b path] [-n suffixes] [-t date] [-tt date] [zipfile [file ...]] [-x list]
zipcloak (see separate man page)
zipnote (see separate man page)
zipsplit (see separate man page)
Note: Command line processing in zip has been changed to support long options and handle all options and arguments more consistently. Some old command lines that depend on command line inconsistencies may no longer work.
DESCRIPTION
zip is a compression and file packaging utility for Unix, VMS, MSDOS, OS/2, Windows 9x/NT/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 4.6), and PKZIP and PKUNZIP can work with archives produced by zip (with some exceptions, notably streamed archives, 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 1.10 cannot extract files produced by PKZIP 2.04 or zip 2.0. You must use PKUNZIP 2.042 or unzip 5.81 (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 updated (if the resulting archive still needs Zip64), the size of the archive will exceed 4 GB, or when the number of entries in the archive will exceed about 64K. Zip64 is also used for archives streamed from standard input as the size of such archives are not known in advance, but the option -F can be used to force zip to create PKZIP 2 compatible archives (as long as Zip64 extensions are not needed). You must use a PKZIP 4.5 compatible unzip, such as unzip 5.8 or later, to extract files using the Zip64 extensions.
In addition, streamed archives, entries encrypted with standard encryption, or split archives created with the pause option may not be compatible with PKZIP as data descriptors are used and PKZIP at the time of this writing does not support data descriptors (but recent changes in the PKware published zip standard now include some support for the data descriptor format zip uses).
Manual page zip(1) line 1 (press h for help or q to quit)
```

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)

```

bzip2(1)                                     General Commands Manual                 bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
  bcat - decompresses files to stdout
  bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
  bzip2 [-cdffkqvz] [filenames ...]
  bunzip2 [-fkvz] [filenames ...]
  bcat [-rs] [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 LZW/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, permissions, 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 preserving original file names, permissions, ownerships or dates in filesystems which lack these concepts, or have serious 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.

  bzip2 (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 the compressed file as follows:

      filename.bz2 becomes filename
      filename.bz becomes filename
      filename.tbz2 becomes filename.tar
      filename.tbz becomes filename.tar
      anyothername becomes anyothername.out

  If the file does not end in one of the recognised endings, .bz2, .bz, .tbz2 or .tbz, bzip2 complains that it cannot guess the name of the original file, and uses the
Manual page bzip2(1) line 1 (press h for help or q to quit)

```

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)

```

TAR(1)                                     GNU TAR Manual                         TAR(1)

NAME
  tar - an archiving utility

SYNOPSIS
  Traditional usage
  tar (a|c|d|f|t|u|x)[C|S|K|W|Q|P|W|B|I|J|z|Z|P|l|h|v|w] [ABG...]

  UNIX-style usage
  tar -A [OPTIONS] ARCHIVE ARCHIVE
  tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
  tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
  tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
  tar -r [-f ARCHIVE] [OPTIONS] [FILE...]
  tar -u [-f ARCHIVE] [OPTIONS] [FILE...]
  tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

  GNU-style usage
  tar [--catenate|--concatenate] [OPTIONS] ARCHIVE ARCHIVE
  tar --create [--file ARCHIVE] [OPTIONS] [FILE...]
  tar [--diff|--compare] [--file ARCHIVE] [OPTIONS] [FILE...]
  tar --delete [--file ARCHIVE] [OPTIONS] [MEMBER...]
  tar --append [-f ARCHIVE] [OPTIONS] [FILE...]
  tar --list [-f ARCHIVE] [OPTIONS] [MEMBER...]
  tar --test-label [--file ARCHIVE] [OPTIONS] [LABEL...]
Manual page tar(1) line 1 (press h for help or q to quit)

```

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 скрипта (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_places    work
abc1          cat.txt       friend.sh      key.pub        lab4-i.asm   newfriend     ski_places    Видео
adilya        feathers      fr1.sh        khusyainova.cpp may            public        ski_places    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

```
afkhusyainova@dk4n56 ~ $ cd backup
afkhusyainova@dk4n56 ~/backup $ ls
backup.sh.bz2
```

Рис. 0.8.: Просмотр каталога

```
afkhusyainova@dk4n56 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

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

Рис. 0.9.: Просмотр содержимого

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

```
~ : bash — Konsole
Файл Правка Вид Закладки Настройка Справка
afkhusyainova@dk4n56 ~ $ touch prog.sh
afkhusyainova@dk4n56 ~ $ emacs &
[1] 15322

emacs@dk4n56
File Edit Options Buffers Tools Sh-Script Help

U:--- prog.sh All L1 (Shell-script[sh]) Чт мая 19 11:50 1.01
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
█

U:%*- *Warnings* All L8 (Special) Чт мая 19 11:50 1.01
Indentation setup for shell type sh
```

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)

```

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

U:***- prog.sh All L5 (Shell-script[sh]) Чт мая 19 11:52 1.85
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
[]

U:***- *Warnings* All L8 (Special) Чт мая 19 11:52 1.85

```

Рис. 0.11.: Программа

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

```

afkhusyainova@den56 ~ % chmod +x *.sh
afkhusyainova@den56 ~ % ls
./sh backup file.txt fr.sh lab01 monthly prog.sh ski.pases Видео Общедоступные
abct backup.sh friend.cpp key lab07.sh my.os public_html text.txt Загрузки "Рабочий стол"
adilya cmt.txt friend.sh key.pub lab4-1.asm newfriend reports tap Изображения Шаблоны
adilya feathers fri.sh khusyainova.cpp may prog.sh sci.pases work Музыка
afkhusyainova@den56 ~ % ./prog.sh 1 2 3 4 5 6 7
Аргументы
1
2
3
4
5
6
7

```

Рис. 0.12.: Выполнение


```
afkhusyainova@edk4n56 ~ $ ./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 ./progl.sh запустим его (рис.15)

```
afkhusyainova@dk4n56 ~ $ chmod +x *.sh
afkhusyainova@dk4n56 ~ $ ls
1.sh          backup      file.txt    fr.sh       lab01       monthly    proglis.sh~  reports     tmp         Изображения
2022-04-28 10-49-58.mkv backup.sh    friend.cpp  games       lab07.sh   monthly_00 prog.sh      sci.places  work        Музыка
abc1          backup.sh~  friend.cpp.save key         lab07.sh~  mv.sh       prog.sh~     ski.places  video       Ожидается
adilya       conf.txt    friend.sh   key.pub     lab4-1.asm newfriend   public      ski.places  документы  'Рабочий стол'
adilya       feathers    fri.sh     khusyainova.cpp may         proglis.sh  public_html text.txt    Загрузки   Шаблоны
afkhusyainova@dk4n56 ~ $ ./proglis.sh ~
/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/1.sh
Облачный файл

/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/2022-04-28 10-49-58.mkv
./proglis.sh: строка 7: test: /afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/2022-04-28: ожидается бинарный оператор
./proglis.sh: строка 11: test: /afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/2022-04-28: ожидается бинарный оператор
./proglis.sh: строка 15: test: /afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/2022-04-28: ожидается бинарный оператор
./proglis.sh: строка 19: test: /afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/2022-04-28: ожидается бинарный оператор
./proglis.sh: строка 23: test: /afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/2022-04-28: ожидается бинарный оператор
/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/abc1
Облачный файл

/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/adilya
Облачный файл

/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/adilya
Облачный файл

/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/backup
./afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/backup.sh
Облачный файл

/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/backup.sh~
Облачный файл

/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/conf.txt
Облачный файл

/afs/dk.sci.pfu.edu.ru/home/a/f/afkhusyainova/feathers
Облачный файл
```

Рис. 0.15.: Запуск файла

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

```
afkhusyainova@dk4n56 ~ $ touch format.sh
afkhusyainova@dk4n56 ~ $ emacs &
[1] 17619
```

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)

```
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 разрешением $a"
done
```

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)

```
afkhusyainova@dk4n56 ~$ chmod +x *.sh
[13]~ Завершен emacs
afkhusyainova@dk4n56 ~$ chmod +x *.sh
afkhusyainova@dk4n56 ~$ touch fail.pdf file.docx file2.docx
afkhusyainova@dk4n56 ~$ ls
. .sh 2022-04-28 18:49:58 mv~ backup feathrs format.sh~ fr.sh lab01 monthly proglis.sh~ reports tmp Изображения
abcl backup.sh file2.docx friend.cpp save key lab07.sh~ monthly.sh prog.sh~ sci.plases work Имена
adilya docx.txt file.txt friend.sh key.pub lab07.sh~ my_os prog.sh~ ski.plases Видео
adilya fail.pdf format.sh fri.sh khusyainova.cpp may newfriend public_html test.txt Загрузки "Рабочий стол"
afkhusyainova@dk4n56 ~$ ./format.sh ~ pdf txt docx
0 файлов содержится в каталоге /afs/ dk.sci.pfu.edu.ru/home/afkhusyainova с разрешением pdf
0 файлов содержится в каталоге /afs/ dk.sci.pfu.edu.ru/home/afkhusyainova с разрешением txt
0 файлов содержится в каталоге /afs/ dk.sci.pfu.edu.ru/home/afkhusyainova с разрешением docx
```

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 (Bourne shell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; C-оболочка (или csh) – надстройка на оболочке 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.
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 `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.

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 `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.
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 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 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 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 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.

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 `unset` с флагом `-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 `Debian` команда `«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 –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 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`.