

Лабораторная работа №10

Отчёт к лабораторной работе

Зайцева Анна Дмитриевна

Table of Contents

Цель работы

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

Задание

1. Написать 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. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать 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.
4. Написать 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.

Выполнение лабораторной работы

1. Я изучила справку о командах архивации (команды: *man zip*, *man bzip2*, *man tar*) (Рис. [-@fig:001])(Рис. [-@fig:002])(Рис. [-@fig:003])(Рис. [-@fig:004]) и открыла emacs (команда: *emacs*) (Рис. [-@fig:005]):

```
ZIP(1L)                                                                                                     ZIP(1L)

NAME
    zip - package and compress (archive) files

SYNOPSIS
    zip [-sABCdceffghjllmqrSTuvWxyzIO@] [--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, MS-DOS, 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 MS-DOS 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 3.0. You must use PKUNZIP 2.04g or unzip 5.0p1 (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 5.0 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).

    Mac OS X. Though previous Mac versions had their own zip port, zip supports Mac OS X as part of the Unix port and most Unix features apply. References to "MacOS" below generally refer to MacOS versions older than OS X. Support for some Mac OS features in the Unix Mac OS X port, such as resource forks, is expected in the next zip release.

    For a brief help on zip and unzip, run each without specifying any parameters on the command line.

USE
    The program is useful for packaging a set of files for distribution; for archiving files; and for saving disk space by temporarily compressing unused files or directories.

    The zip program puts one or more compressed files into a single zip archive, along with information about the files (name, path, date, time of last modification, protection, and check information to verify file integrity).

Manual page zip(1) line 1 (press h for help or q to quit)
```

Puc. 1

```
bzip2(1)                                                                                                     General Commands Manual                                                                                                     bzip2(1)

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

SYNOPSIS
    bzip2 [-cdfhqtuvVL123456789] [filenames ...]
    bunzip2 [-fuvvL] [filenames ...]
    bzcvt [-s] [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 LZ77/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.

    bunzip2 (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 original name with .out appended.

    As with compression, supplying no filenames causes decompression from standard input to standard output.

    bunzip2 will correctly decompress a file which is the concatenation of two or more compressed files. The result is the concatenation of the corresponding uncompressed files. Integrity testing (-t) of concatenated compressed files is also supported.

Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Puc. 2

```

TAR(1)

NAME
    tar - an archiving utility

SYNOPSIS
    Traditional usage
        tar {A|c|d|r|t|u|x}[GnSkUW0mpsMBiajJzZhPlRvwo] [ARG...]

    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...]

        tar --update [--file ARCHIVE] [OPTIONS] [FILE...]

        tar --update [-f ARCHIVE] [OPTIONS] [FILE...]

        tar {--extract|--get} [-f ARCHIVE] [OPTIONS] [MEMBER...]
Manual page tar(1) line 1 (press h for help or q to quit)

```

```
[adzayjceva@adzayjceva lab10]$ man zip
[adzayjceva@adzayjceva lab10]$ man bzip2
[adzayjceva@adzayjceva lab10]$ man tar
[adzayjceva@adzayjceva lab10]$
```

Рис. 4

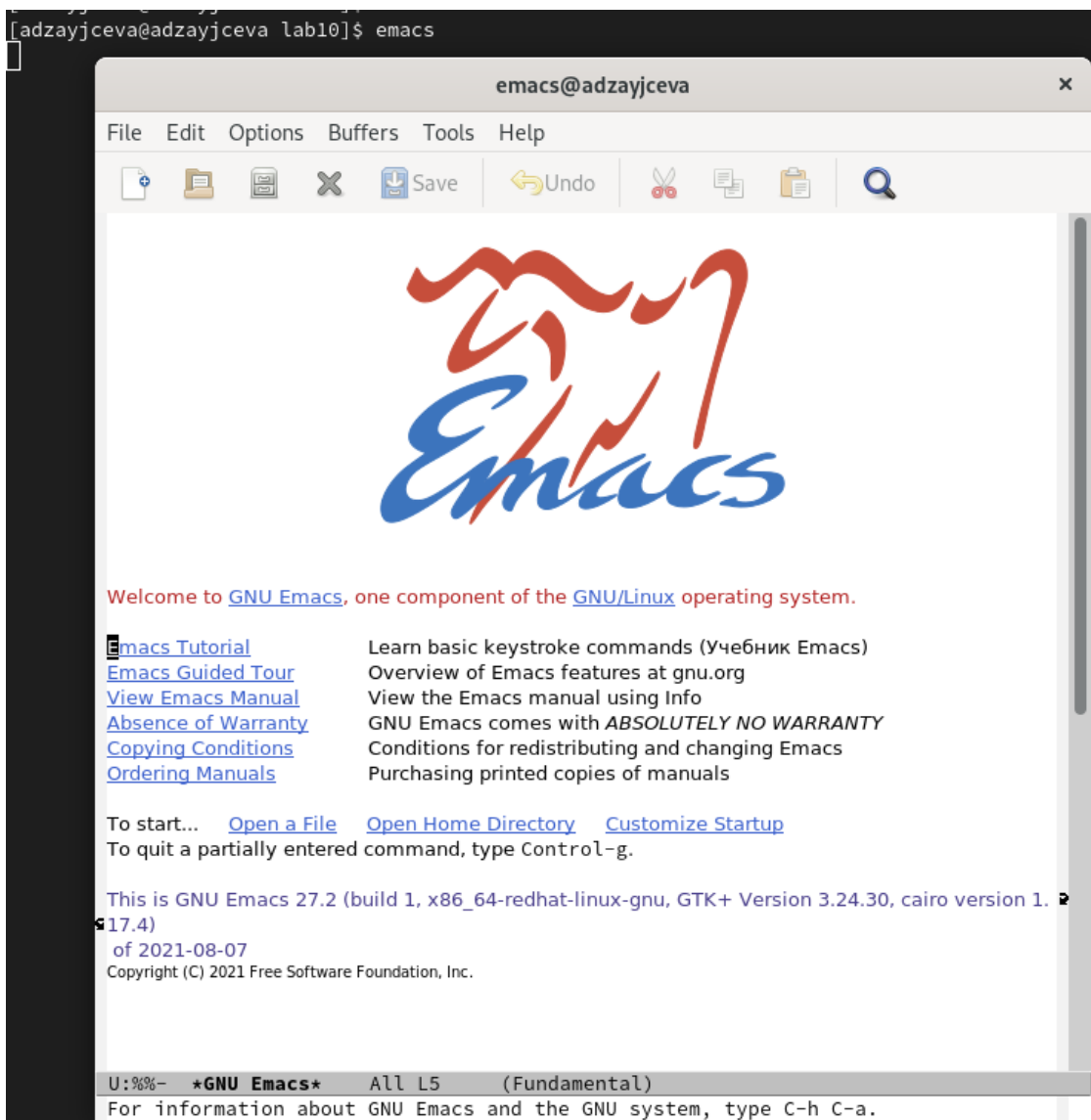


Рис. 5

2. Создала файл backup.sh с помощью комбинации Ctrl-x Ctrl-f (C-x C-f) (Рис. [-@fig:006]):

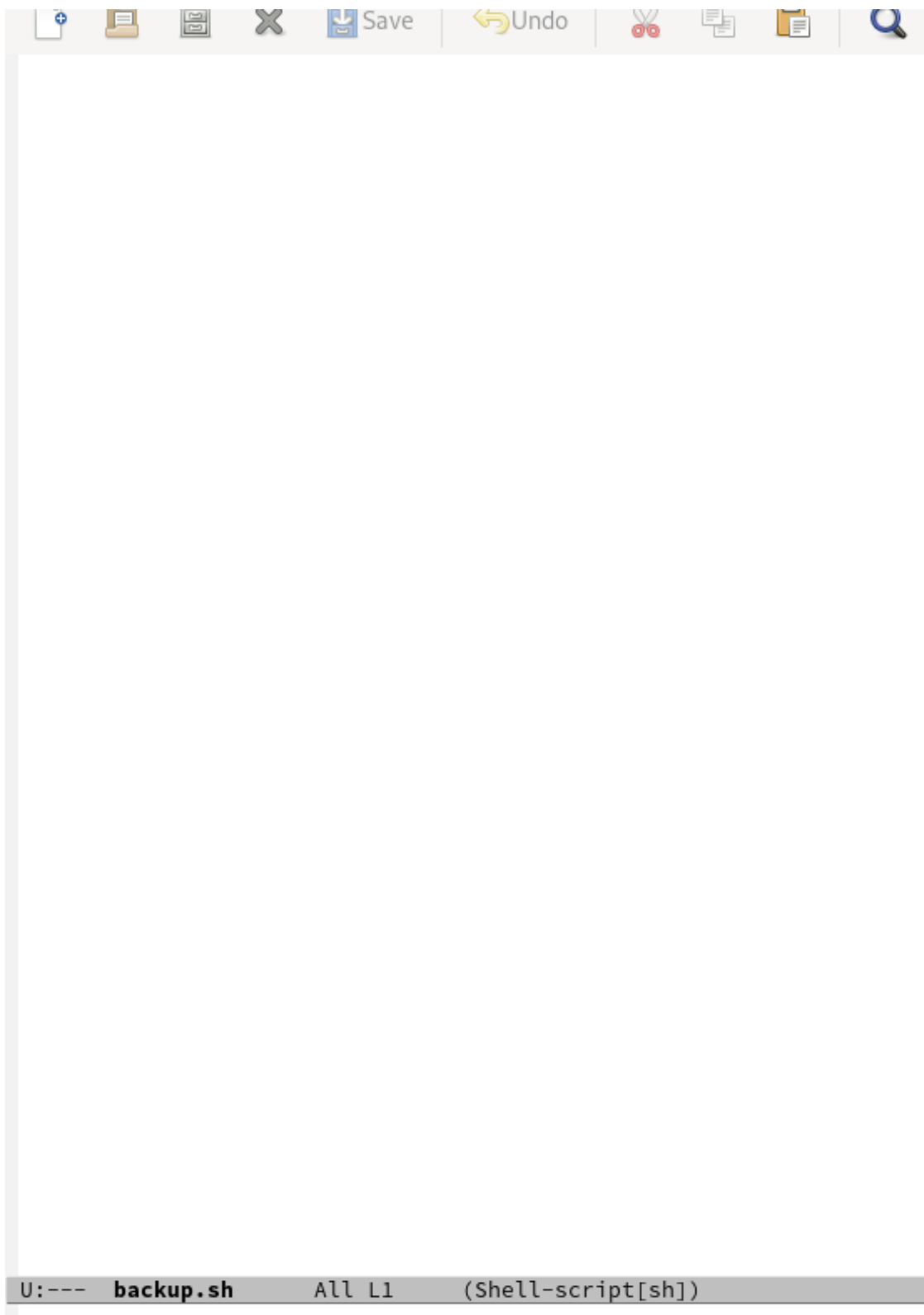


Рис. 6

3. 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 моём домашнем каталоге. При этом файл архивируется

одним из архиваторов на выбор zip, bzip2 или tar. Я выбрала bzip2. (Рис. [-@fig:007]):

```
[adzayjceva@adzayjceva lab10]$ ./backup.sh
All done
[adzayjceva@adzayjceva lab10]$ cd ~/backup
[adzayjceva@adzayjceva backup]$ ls
backup.sh.bz2
[adzayjceva@adzayjceva backup]$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'          # копируем в переменную файл со скриптом
mkdir ~/backup             # создаём каталог в домашнем каталоге
bzip2 -k ${name}           # архивируем
mv ${name}.bz2 ~/backup/   # перемещаем архив в созданный каталог
echo "All done"           # выводим сообщение о том, что программа выполнена успешно
[adzayjceva@adzayjceva backup]$
```

Рис. 7

4. 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. Создала файл script2.sh с помощью комбинации Ctrl-x Ctrl-f (C-x C-f), добавила ему права на исполнение (команда: *chmod +x .sh*). Запустила файл с 2 разными наборами аргументов, в которых первый набор включал в себя число аргументов меньше 10, а второй - больше 10. Далее я вывела в терминал скрипт программы (команда: *cat script2.sh**). (Рис. [-@fig:008]):

```

[adzayjceva@adzayjceva lab10]$ chmod +x *.sh
[adzayjceva@adzayjceva lab10]$ ls -l
итого 28
-rwxrwxr-x. 1 adzayjceva adzayjceva 522 мая 21 19:15 backup.sh
-rwxrwxr-x. 1 adzayjceva adzayjceva 522 мая 21 19:10 backup.sh~
drwxr-xr-x. 1 adzayjceva adzayjceva 70 мая 21 19:25 lab10_images
-rw-rw-r--. 1 adzayjceva adzayjceva 5295 мая 21 16:51 lab10_presentation.md
-rw-r--r--. 1 adzayjceva adzayjceva 4548 мая 21 18:19 lab10_report.md
-rwxrwxr-x. 1 adzayjceva adzayjceva 164 мая 21 19:27 script2.sh
[adzayjceva@adzayjceva lab10]$ ./script2.sh 0 1 2 3 4 5
Args
0
1
2
3
4
5
[adzayjceva@adzayjceva lab10]$ ./script2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Args
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
[adzayjceva@adzayjceva lab10]$ cat script2.sh
#!/bin/bash

echo "Args"
for a in $@      #цикл для прохода по введенным аргументам
do echo $a      #вывод аргумента
done
[adzayjceva@adzayjceva lab10]$

```

Рис. 8

5. Написала командный файл—аналог команды `ls` (без использования самой этой команды и команды `dir`). Он выдаёт информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога. Создала файл `script3.sh` с помощью комбинации `Ctrl-x Ctrl-f` (C-x C-f), добавила ему права на исполнение (команда: `chmod +x *.sh`). (Рис. [-@fig:009]):

```
[adzayjceva@adzayjceva lab10]$ chmod +x *.sh
[adzayjceva@adzayjceva lab10]$ ./script3.sh ~
./script3.sh: строка 3: а: команда не найдена
/bin
Directory
Read+
Write-
Execute+
/boot
Directory
Read+
Write-
Execute+
/dev
Directory
Read+
Write-
Execute+
/etc
Directory
Read+
Write-
Execute+
/home
Directory
Read+
Write-
Execute+
/lab03
File
Read+
Write+
Execute-
/lib
Directory
Read+
Write-
Execute+
/lib64
Directory
Read+
```


Рис. 9

```
#!/bin/bash
```

```
a = "$1"                                # в переменную а сохраняем путь до заданного
каталога                                # цикла, пробегающий по всем каталогам и
for i in ${a}/*
файлам
do
    echo "$i"

    if test -f $i
    then echo "File"
    fi

    if test -d $i
    then echo "Directory"
    fi

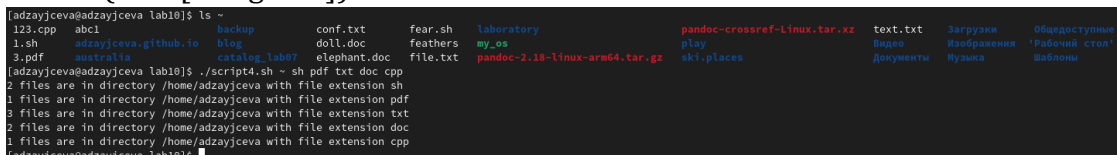
    if test -r $i
    then echo "Read+"
    else echo "Read-"
    fi

    if test -w $i
    then echo "Write+"
    else echo "Write-"
    fi

    if test -x $i
    then echo "Execute+"
    else echo "Execute-"
    fi

done
```

6. Написала 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. Создала файл script4.sh с помощью комбинации Ctrl-x Ctrl-f (C-x C-f), добавила ему права на исполнение (команда: *chmod +x .sh**). (Рис. [-@fig:010]):



```
[adzajceva@adzajceva lab10]$ ls ~
123.cpp  abc1  backup  conf.txt  fear.sh  laboratory  pandoc-crossref-Linux.tar.xz  text.txt  Загрузки  Общедоступные
1.sh     adzajceva.github.io  blog  doll.doc  feathers  my_os  play  Видео  Изображения  'Рабочий стол'
3.pdf    astralis  cat4log_lab07  elephant.doc  file.txt  pandoc-2.10-linux-arm64.tar.gz  ski.places  Документы  Музыка  Шаблоны
[adzajceva@adzajceva lab10]$ ./script4.sh ~ sh pdf txt doc cpp
2 files are in directory /home/adzajceva with file extension sh
1 files are in directory /home/adzajceva with file extension pdf
3 files are in directory /home/adzajceva with file extension txt
2 files are in directory /home/adzajceva with file extension doc
1 files are in directory /home/adzajceva with file extension cpp
[adzajceva@adzajceva lab10]$
```

Рис. 10

```
#!/bin/bash
```

```
b=$1
```

```

shift                # удаляет первый параметр и сдвигает все остальные на
места предыдущих
for a in $@
do
    c=0                # счётчик
    for i in ${b}/*.${a}
    do
        if test -f "$i"
        then
            let c=c+1
        fi
    done
    echo "$c files are in directory $b with file extension $a"
done

```

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:
 1. оболочка Борна (BourneShell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
 2. C-оболочка (или csh) – надстройка на оболочкой Борна, использующая Сподобный 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 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 «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 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 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.
- 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 cd 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 `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:
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.

Вывод

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