

Лабораторная работа

Дисциплина: Операционные системы

Егорова Александра

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Выводы	13
4	Контрольные вопросы	14

List of Figures

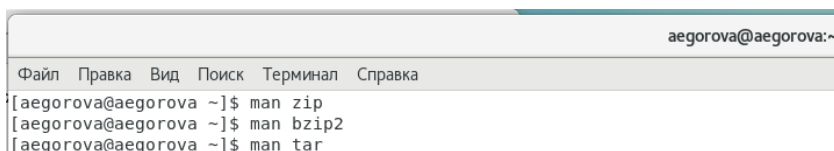
2.1	Выполнение команд	5
2.2	Команда архивации zip	5
2.3	Команда архивации bzip2	6
2.4	Команда архивации tar	6
2.5	Создаем 1 файл	7
2.6	Первый скрипт	7
2.7	Проверка первого скрипта	8
2.8	Создаем 2 файл	8
2.9	Второй скрипт	9
2.10	Проверка второго скрипта	9
2.11	Создаем 3 файл	9
2.12	Третий скрипт	10
2.13	Проверка третьего скрипта	10
2.14	Создаем 4 файл	11
2.15	Четвертый скрипт	11
2.16	Проверка четвертого скрипта	11
2.17	Проверка четвертого скрипта	12

1 Цель работы

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

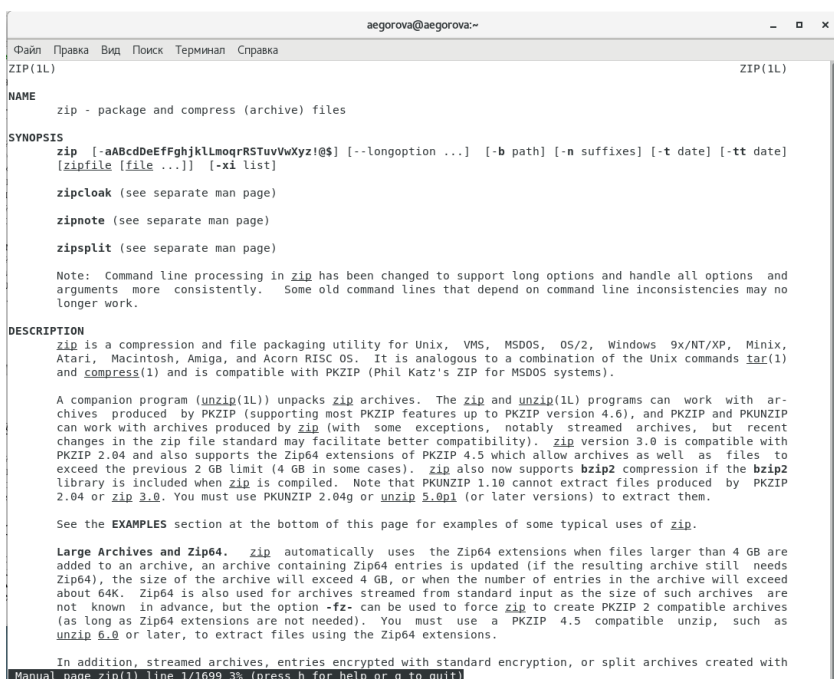
2 Выполнение лабораторной работы

- 1) Для начала я изучила команды архивации, используя команды «man zip», «man bzip2», «man tar» (рис. -fig. 2.1) (рис. -fig. 2.2) (рис. -fig. 2.3) (рис. -fig. 2.4)



```
aegorova@aegorova:~  
Файл Правка Вид Поиск Терминал Справка  
[aegorova@aegorova ~]$ man zip  
[aegorova@aegorova ~]$ man bzip2  
[aegorova@aegorova ~]$ man tar
```

Figure 2.1: Выполнение команд



```
aegorova@aegorova:~  
Файл Правка Вид Поиск Терминал Справка  
ZIP(1L) ZIP(1L)  
NAME  
zip - package and compress (archive) files  
SYNOPSIS  
zip [-aABcdDeFFghjklLmoqrRStuvVwXyz!@] [--longoption ...] [-b path] [-n suffixes] [-t date] [-tt date]  
[zipfile [file ...]] [-xi 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 ar-  
chives 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 -fz- 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 6.0 or later, to extract files using the Zip64 extensions.  
In addition, streamed archives, entries encrypted with standard encryption, or split archives created with  
Manual page zip(1) line 1/1699 3% (press h for help or q to quit)
```

Figure 2.2: Команда архивации zip

```
aegorova@aegorova:~$ man bzip2
bzip2(1)                                General Commands Manual                                bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.6
  bzcatt - decompresses files to stdout
  bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
  bzip2 [ -cdfkqstzVL123456789 ] [ filenames ... ]
  bunzip2 [ -fkvsVL ] [ filenames ... ]
  bzcatt [ -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 L77/L78-based compressors, and approaches the perfor-
  mance 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 possi-
  ble, 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 fol-
  lows:

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

Figure 2.3: Команда архивации bzip2

```
aegorova@aegorova:~$ man tar
tar(1)                                User Commands                                tar(1)

NAME
  tar - manual page for tar 1.26

SYNOPSIS
  tar [OPTION...] [FILE]...

DESCRIPTION
  GNU 'tar' saves many files together into a single tape or disk archive, and can restore individual files from the archive.

  Note that this manual page contains just very brief description (or more like a list of possible functionality) originally
  generated by the help2man utility. The full documentation for tar is maintained as a Texinfo manual. If the info and tar
  programs are properly installed at your site, the command 'info tar' should give you access to the complete manual.

EXAMPLES
  tar -cf archive.tar foo bar
  # Create archive.tar from files foo and bar.

  tar -tvf archive.tar
  # List all files in archive.tar verbosely.

  tar -xf archive.tar
  # Extract all files from archive.tar.

DEFAULTS
  *This* tar installation defaults to:

  --format=gnu -f- -b20 --quoting-style=escape --rmt-command=/sbin/rmt --rsh-command=/usr/bin/rsh

Main operation mode:
  -A, --catenate, --concatenate
  append tar files to an archive

  -C, --create
  create a new archive

  -d, --diff, --compare
  find differences between archive and file system

  --delete
  delete from the archive (not on mag tapes!)
```

Figure 2.4: Команда архивации tar

Далее я создала файл, в котором буду писать первый скрипт, и открыла его в редакторе emacs (touch backup.sh и 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. -fig. 2.5) (рис. -fig. 2.6)

```
[aegorova@aegorova ~]$ touch backup.sh  
[aegorova@aegorova ~]$ emacs &
```

Figure 2.5: Создаем 1 файл

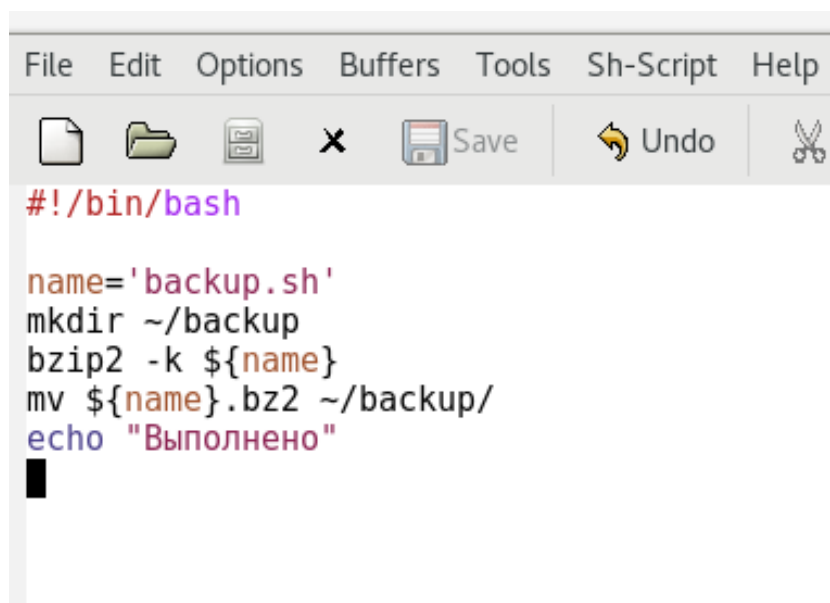


Figure 2.6: Первый скрипт

Для начала добавила для скрипта право на выполнение (команда «chmod +x *.sh»). Проверила работу скрипта (команда «./backup.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. -fig. 2.7)

```
aegorova@aegorova:~  
Файл Правка Вид Поиск Терминал Справка  
[aegorova@aegorova ~]$ emacs &  
[1] 5053  
[aegorova@aegorova ~]$ chmod +x *.sh  
[1]+  Done emacs  
[aegorova@aegorova ~]$ ./backup.sh  
Выполнено  
[aegorova@aegorova ~]$ cd backup/  
[aegorova@aegorova backup]$ ls  
backup.sh.bz2  
[aegorova@aegorova backup]$ bunzip2 -c backup.sh.bz2  
#!/bin/bash  
name='backup.sh'  
mkdir ~/.backup  
bzip2 -k ${name}  
mv ${name}.bz2 ~/.backup/  
echo "Выполнено"
```

Figure 2.7: Проверка первого скрипта

- 2) Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs («touch prog2.sh» и «emacs &»). (рис. -fig. 2.8)

```
[aegorova@aegorova ~]$ touch prog2.sh  
[aegorova@aegorova ~]$ emacs &
```

Figure 2.8: Создаем 2 файл

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. Проверила работу написанного скрипта («./prog2.sh 0 1 2 3 4 5» и «./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Вводила аргументы, количество которых меньше 10 и больше 10. Скрипт работает корректно. (рис. -fig. 2.9) (рис. -fig. 2.10)

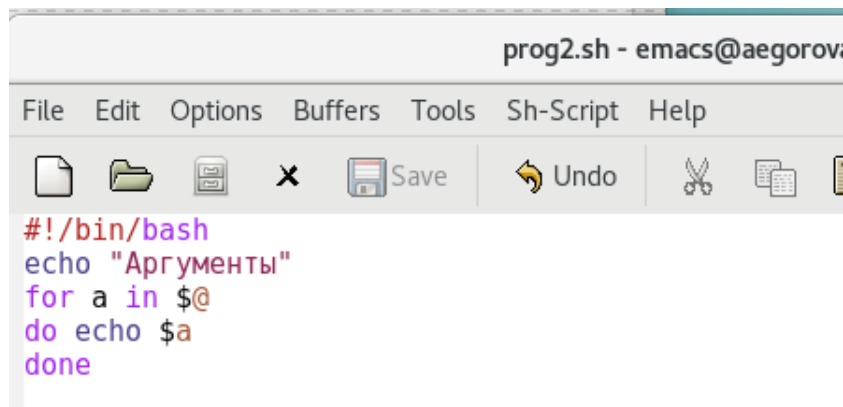


Figure 2.9: Второй скрипт

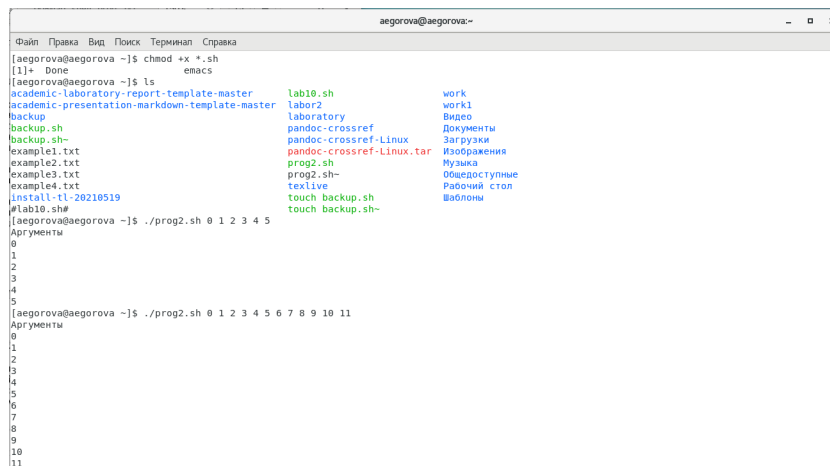


Figure 2.10: Проверка второго скрипта

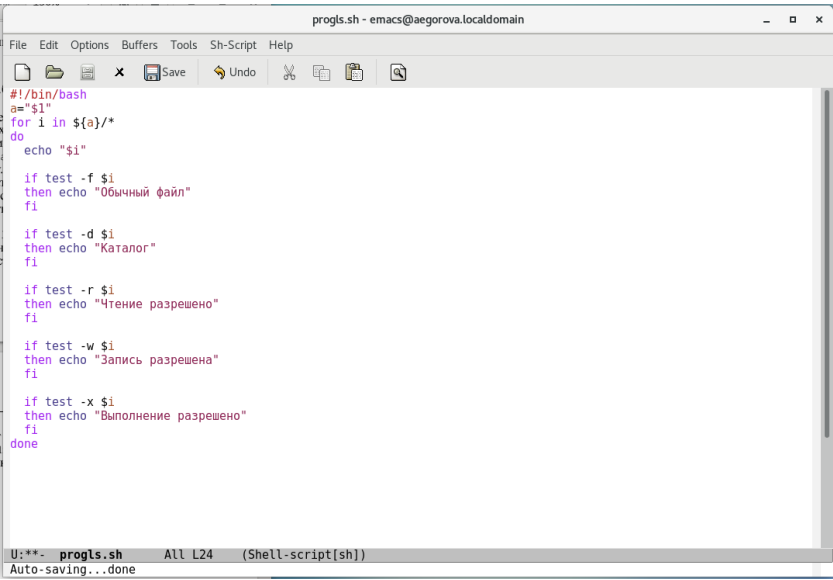
- 3) Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs («touch progl.sh» и «emacs &»). (рис. -fig. 2.11)

```
[aegorova@aegorova ~]$ touch progl.sh
[aegorova@aegorova ~]$ emacs &
```

Figure 2.11: Создаем 3 файл

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. Далее

проверила работу скрипта («./progl.sh ~»), предварительно добавив для него право на выполнение («chmod +x *.sh»). Скрипт работает корректно. (рис. -fig. 2.12) (рис. -fig. 2.13)



```
#!/bin/bash
a="$1"
for i in ${a}/*
do
  echo "$i"

  if test -f $i
  then echo "Обычный файл"
  fi

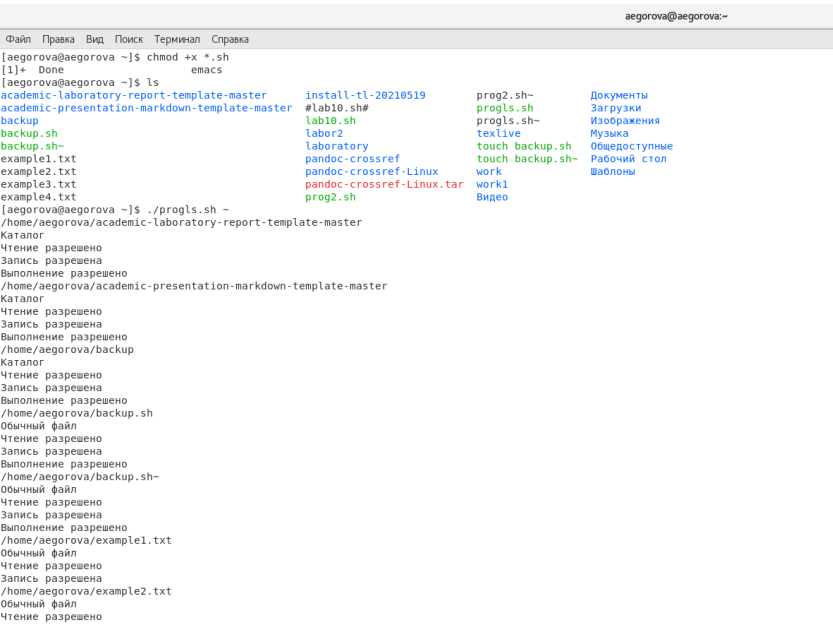
  if test -d $i
  then echo "Каталог"
  fi

  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
fi
done
```

Figure 2.12: Третий скрипт



```
aegorova@aegorova:~$ chmod +x *.sh
aegorova@aegorova:~$ ls
academic-laboratory-report-template-master  install-til-20210519  prog2.sh
academic-presentation-markdown-template-master  lab10.sh              prog2.sh
backup                                         lab2                  touch backup.sh
example1.txt                                laboratory            touch backup.sh
example2.txt                                pandoc-crossref       work
example3.txt                                pandoc-crossref-Linux work1
example4.txt                                pandoc-crossref-Linux.tar  work2
```

Figure 2.13: Проверка третьего скрипта

4) Для четвертого скрипта также создала файл и открыла его в редакторе

emacs. (команда «prog4.sh» и «emacs &»). (рис. -fig. 2.14)

```
[aegorova@aegorova ~]$ touch prog4.sh
[aegorova@aegorova ~]$ emacs &
```

Figure 2.14: Создаем 4 файл

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. (рис. -fig. 2.15)

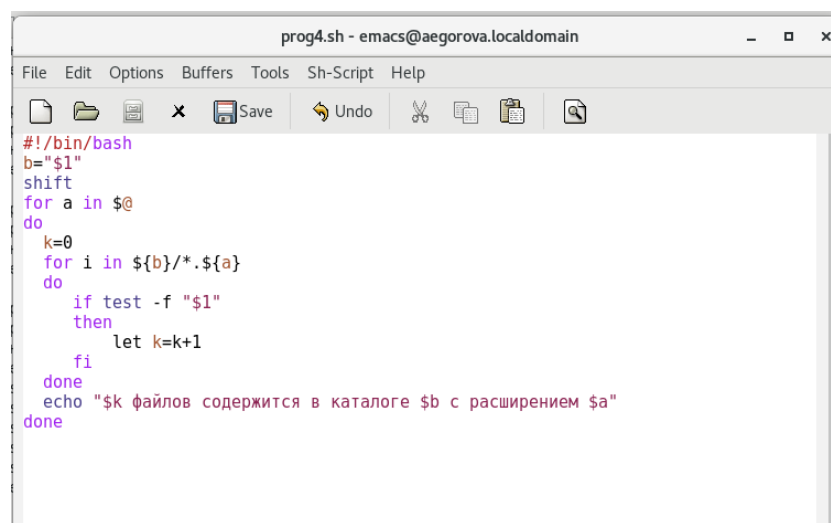


Figure 2.15: Четвертый скрипт

Проверила работу написанного скрипта («./format.sh ~ pdf sh txt doc»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»), а также создав дополнительные файлы с разными расширениями («touch file.pdf file1.doc file2.doc»). Скрипт работает корректно. (рис. -fig. 2.16)(рис. -fig. 2.17)

```
[aegorova@aegorova ~]$ chmod +x *.sh
[1]+  Done                  emacs
[aegorova@aegorova ~]$ touch file.pdf file2.doc
[aegorova@aegorova ~]$ ls
academic-laboratory-report-template-master  file.pdf
academic-presentation-markdown-template-master  install-tl-20210519
backup                                          #lab10.sh#
backup.sh                                     lab10.sh
backup.sh~                                   labor2
example1.txt                                  laboratory
example2.txt                                pandoc-crossref
example3.txt                                pandoc-crossref-Linux
example4.txt                                pandoc-crossref-Linux.tar
file2.doc                                    prog2.sh
                                              prog2.sh~
                                              prog4.sh
                                              prog4.sh~
                                              progl5.sh
                                              progl5.sh~
                                              texlive
                                              touch backup.sh
                                              touch backup.sh~
                                              work
                                              work1
```

Figure 2.16: Проверка четвертого скрипта

```
[aegorova@aegorova ~]$ ./prog4.sh ~ pdf sh txt doc
1 файлов содержится в каталоге /home/aegorova с расширением pdf
6 файлов содержится в каталоге /home/aegorova с расширением sh
5 файлов содержится в каталоге /home/aegorova с расширением txt
1 файлов содержится в каталоге /home/aegorova с расширением doc
[aegorova@aegorova ~]$
```

Figure 2.17: Проверка четвертого скрипта

3 Выводы

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

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) С-оболочка (или csh) – надстройка на оболочкой Борна, использующая подобный синтаксис команд с возможностью сохранения истории выполнения команд;
 - 3) оболочка Корна (или ksh) – напоминает оболочку С, 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 С 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/Linuxподобных 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 `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.

- 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, 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: 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 (`new line`). 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 `You have mail` (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 метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом. 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 , – `echo ab'|'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 осуществит 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` с 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`.