

Отчет по лабораторной работе №10

дисциплина: Операционные системы

Старков Никита Алексеевич

Содержание

| | | |
|---|--------------------------------|----|
| 1 | Цель работы | 4 |
| 2 | Выполнение лабораторной работы | 5 |
| 3 | Контрольные вопросы | 12 |
| 4 | Выводы | 18 |

Список иллюстраций

| | | |
|------|--|----|
| 2.1 | Архиватор zip | 5 |
| 2.2 | Архиватор bzip2 | 6 |
| 2.3 | Архиватор tar | 6 |
| 2.4 | Создание файла backup.sh | 6 |
| 2.5 | Скрипт №1 | 7 |
| 2.6 | Проверка работы скрипта | 7 |
| 2.7 | Проверка работы скрипта | 7 |
| 2.8 | Проверка работы скрипта | 8 |
| 2.9 | Создание файла prog2.sh | 8 |
| 2.10 | Скрипт №2 | 8 |
| 2.11 | Проверка работы скрипта | 9 |
| 2.12 | Создание файла progl.sh | 9 |
| 2.13 | Скрипт №3 | 10 |
| 2.14 | Перемещение курсора в конец буфера обмена. | 10 |
| 2.15 | Создание файла format.sh | 11 |
| 2.16 | Скрипт №4 | 11 |
| 2.17 | Проверка работы скрипта | 11 |

1 Цель работы

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

2 Выполнение лабораторной работы

1) Изучаем архиваторы zip, bzip 2 и tar с помощью команды man.

```
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 un-
    zip(1L) programs can work with archives produced by PKZIP (supporting
    Manual page zip(1) line 1 (press h for help or q to quit)
```

Рис. 2.1: Архиватор zip

| | | |
|--|-------------------------|----------|
| bzip2(1) | General Commands Manual | bzip2(1) |
| NAME | | |
| bzip2, bunzip2 - a block-sorting file compressor, v1.0.8 bzcat - decompresses files to stdout bzip2recover - recovers data from damaged bzip2 files | | |
| SYNOPSIS | | |
| bzip2 [-cdfkqstvwVL123456789] [filenames ...] bunzip2 [-fkvsVL] [filenames ...] bzcat [-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- | | |

Рис. 2.2: Архиватор bzip2

DESCRIPTION

GNU **tar** is an archiving program designed to store multiple files in a single file (an **archive**), and to manipulate such archives. The archive can be either a regular file or a device (e.g. a tape drive, hence the name of the program, which stands for **tape archiver**), which can be located either on the local or on a remote machine.

Option styles

Options to GNU **tar** can be given in three different styles. In **traditional style**, the first argument is a cluster of option letters and all subsequent arguments supply arguments to those options that require them. The arguments are read in the same order as the option letters. Any command line words that remain after all options has been processed are treated as non-optional arguments: file or archive member names.

For example, the **c** option requires creating the archive, the **v** option requests the verbose operation, and the **f** option takes an argument that sets the name of the archive to operate upon. The following command, written in the traditional style, instructs tar to store all files from the directory **/etc** into the archive file **etc.tar** verbosely listing the files being archived:

```
tar cfv etc.tar /etc
```

Рис. 2.3: Архиватор tar

Создаем файл, в котором будем писать скрипт и открываем emacs

```
nastarkov@dk3n59 ~ $ touch backup.sh
nastarkov@dk3n59 ~ $ emacs &
```

Рис. 2.4: Создание файла backup.sh

Пишем 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.

```
#!/bin/bash

name='backup.sh'      #В переменную name сохраняем файл со скриптом
mkdir ~/backup        #Создаем каталог ~/backup
bzip2 -k $(name)      #Архивируем скрипт
mv $(name).bz2 ~/backup #Перемещаем архивированный скрипт в каталог ~/backup
echo 'Выполнено'
```

```
--- backup.sh All L7 (Shell-script[sh]) Чт мая 19 16:20 0.96
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.
█
```

```
:%*- *Warnings* All L8 (Special) Чт мая 19 16:20 0.96
Wrote /afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/backup.sh
```

Рис. 2.5: Скрипт №1

Проверяем работу скрипта, предварительно добавив для него право на выполнение

```
nastarkov@dk3n59 ~ $ chmod +x *.sh
```

Рис. 2.6: Проверка работы скрипта

```
nastarkov@dk3n59 ~ $ ./backup.sh
mkdir: невозможно создать каталог «/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/backup»: Файл существует
Выполнено
```

Рис. 2.7: Проверка работы скрипта

```
nastarkov@dk3n59 ~ $ ./backup.sh
mkdir: невозможно создать каталог «/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/backup»: Файл существует
Выполнено
[4] Завершил emacs
nastarkov@dk3n59 ~ $ cd backup/
nastarkov@dk3n59 ~/backup $ ls
backup.sh.bz2
nastarkov@dk3n59 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'      #В переменную name сохраняем файл со скриптом
mkdir ~/backup        #Создаем каталог ~/backup
bzip2 -k ${name}       #Архивируем скрипт
mv ${name}.bz2 ~/backup #Перемещаем архивированный скрипт в каталог ~/backup
echo 'Выполнено'
```

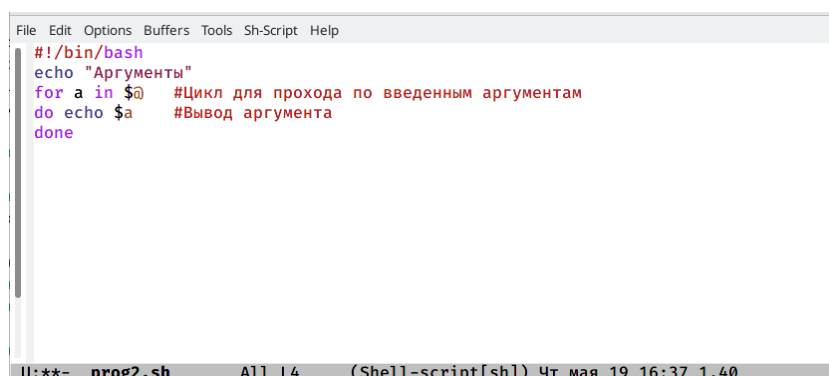
Рис. 2.8: Проверка работы скрипта

2)Создаем файл, в котором будем писать второй скрипт и так же открываем его в emacs.

```
nastarkov@dk3n59 ~/backup $ cd ~
nastarkov@dk3n59 ~ $ touch prog2.sh
nastarkov@dk3n59 ~ $ emacs &
```

Рис. 2.9: Создание файла prog2.sh

Написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов



```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
echo "Аргументы"
for a in $@ #Цикл для прохода по введенным аргументам
do echo $a  #Вывод аргумента
done
U:***- prog2.sh All L4 (Shell-script[sh]) Чт мая 19 16:37 1.40
```

Рис. 2.10: Скрипт №2

Проверил работу данного скрипта, предварительно добавив для него право на выполнение


```

nastarkov@dk3n59 ~ $ chmod +x *.sh
nastarkov@dk3n59 ~ $ ls
abc1      conf.txt      f3.txt      lab07.sh      play      text.txt      Музыка
australia course-directory-student-template f3.txt~    lab07.sh~    prog2.sh   tmp           Общедоступные
backup    equipment     f4.txt~    las.cpp       prog2.sh~  work          'Рабочий стол'
backup.sh f1.txt~      f4.txt~    may           public      Видео         Шаблоны
backup.sh~ f1.txt~     feathers  monthly       public_html Документы
bin       f2.txt~     file.txt   my_os         reports     Загрузки
blog      f2.txt~     GNUstep   'NOVIY KATALOG' ski.plases  Изображения

nastarkov@dk3n59 ~ $ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
nastarkov@dk3n59 ~ $ ./prog2.sh 0 1 2 3 4 5 6 7 8 9 10
Аргументы
0
1
2
3
4
5
6
7
8
9
10

```

Рис. 2.11: Проверка работы скрипта

3) Создаем файл для третьего скрипта и открываем в редакторе emacs

```

nastarkov@dk3n59 ~ $ touch progl.sh
nastarkov@dk3n59 ~ $ emacs &

```

Рис. 2.12: Создание файла progl.sh

Пишем 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.

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

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

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

    if test -r $i
    then echo "Чтение разрешено"
    fi

    if test -w $i
    then echo "Запись разрешена"
    fi

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

Рис. 2.13: Скрипт №3

Далее проверяем работу скрипта, предварительно добавив для него право на выполнение

```
nastarkov@dk3n59 ~ $ chmod +x *.sh
nastarkov@dk3n59 ~ $ ls
abcl      conf.txt      f3.txt      lab07.sh      play        reports      Загрузки
australia course-directory-student-template f3.txt~     lab07.sh~    prog2.sh    ski.plases  Изображения
backup    equipment     f4.txt      las.cpp       prog2.sh~   text.txt    Музыка
backup.sh f1.txt       f4.txt~     may          progls.sh   tmp         Общедоступные
backup.sh~ f1.txt~      feathers    monthly      progls.sh~  work        'Рабочий стол'
bin       f2.txt       file.txt    my_os        public      Видео       Шаблоны
blog      f2.txt~      GNUstep    'NOVIY KATALOG' public_html  Документы

nastarkov@dk3n59 ~ $ ./proglis.sh ~
/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/abcl
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/australia
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/backup.sh
Обычный файл
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/backup.sh~
Обычный файл
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/bin
Каталог
Чтение разрешено
Запись разрешена
```

Рис. 2.14: Перемещение курсора в конец буфера обмена.

4)Создаем файл для четвертого скрипта и открываем в редакторе emacs

```
nastarkov@dk3n59 ~ $ touch format.sh
nastarkov@dk3n59 ~ $ emacs
```

Рис. 2.15: Создание файла format.sh

Пишем 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

```
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. 2.16: Скрипт №4

Далее проверяем работу скрипта, предварительно добавив для него право на выполнение

```
nastarkov@dk3n59 ~ $ touch file.pdf file.doc file2.doc
nastarkov@dk3n59 ~ $ ls
australia  course-directory~student-template  f3.txt~  file.txt  my_os  public  Изображения
backup     equipment  f4.txt~  format.sh  'NOVIY KATALOG'  public_html  Музыка
backup.sh  f1.txt~   f4.txt~  format.sh~  play  text.txt  Общедоступные
backup.sh~ f1.txt~   feathers  GNUstep  prog2.sh  work  'Рабочий стол'
bin        f2.txt~   file2.doc  lab07.sh  prog2.sh~  Видео  Шаблоны
blog       f2.txt~   file.doc  lab07.sh~  progl.s.sh  Документы
conf.txt   f3.txt~   file.pdf  may  progl.s.sh~  Загрузки
nastarkov@dk3n59 ~ $ ./format.sh ~ pdf sh txt doc
1 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov с разрешением pdf
5 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov с разрешением sh
7 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov с разрешением txt
2 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov с разрешением doc
```

Рис. 2.17: Проверка работы скрипта

3 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, с

C-оболочка (или csh) – надстройка на оболочкой Борна, использующая Сподобный синтаксис

Оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления программой

BASH – сокращение от BourneAgainShell (опять оболочка Борна), в основе своей совместимая

2). POSIX (Portable Operating System Interface for Computer Environments) – набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX – совместимые оболочки разработаны на базе оболочки Корна.

3). Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение

некоторой строки символов. Например, команда «`mark=/usr/andy/bin`» присваивает значение строки символов `/usr/andy/bin` переменной `mark` типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол `.`, «*`mva file{mark}`*» переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. оболочка `bash` позволяет работать с массивами. Для создания массива используется команда `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 единственный терм (`term`), обычно целочисленный. Команда `let` берет два операнда и присваивает их переменной. Команда `read` позволяет читать значения переменных со стандартного ввода: «`echo "Please enter Month and Day of Birth ?"`» «`read mon day trash`». В переменные `mon` и `day` будут считаны соответствующие значения, введённые с клавиатуры, а переменная `trash` нужна для того, чтобы 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:

`PATH`: значением данной переменной является список каталогов, в которых командный

PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. Если программа, запущенная командным процессором, требует ввода, то эти переменные определяют, что будет введено.

HOME: имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то программа переходит в этот каталог.

IFS: последовательность символов, являющихся разделителями в командной строке, например пробелы.

MAIL: командный процессор каждый раз перед выводом на экран промптера проверяет, есть ли почта для пользователя.

TERM: тип используемого терминала.

LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при запуске терминала.

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'|` 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 Debia нкоманда «`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 пр

`$!` –номер процесса, в рамках которого выполняется последняя вызванная на выполне

`--`–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 заменен

`${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 выв

`${name+value}` -это выражение работает противоположно `${name-value}`. Если переменная

`${name#pattern}` -представляет значение переменной `name` с удалённым самым коротким

`${#name[*]}` и `${#name[@]}`-эти выражения возвращают количество элементов в массиве

4 Выводы

Вывод: в ходе выполнения лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux, научился писать небольшие командные файлы