

Отчёт по лабораторной работе №10

Дисциплина: Операционные системы

Горпинич Елена Михайловна

Содержание

Цель работы.....	1
Задание.....	1
Выполнение лабораторной работы.....	1
Вывод	8
Контрольные вопросы:	8

Цель работы

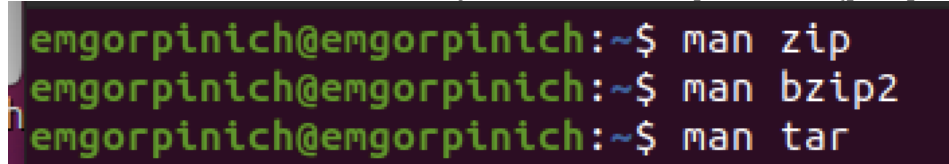
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

Задание

1. Ознакомиться с теоретическим материалом.
2. Сделать отчёт по лабораторной работе №10 в формате Markdown.
3. Изучить основы программирования в оболочке ОС UNIX/Linux.

Выполнение лабораторной работы

- 1) С помощью команды «man», изучим команды архивации(рис.[1-4]).



```
emgorpinich@emgorpinich:~$ man zip
emgorpinich@emgorpinich:~$ man bzip2
emgorpinich@emgorpinich:~$ man tar
```

Синтаксис команды zip для архивации файла: zip [опции] [имя файла.zip] [файлы или папки, которые будем архивировать]

Синтаксис команды zip для разархивации/распаковки файла: unzip [опции] [файл_архива.zip][файлы]-x[исключить]-d[папка]

```

ZIP(1)                                General Commands Manual                                ZIP(1)

NAME
    zip - package and compress (archive) files

SYNOPSIS
    zip [-aABcdDeEfFghjKlLmoqrRSTuvVwXyz!@$] [--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
    Manual page zip(1) line 1 (press h for help or q to quit)

```

2

Синтаксис команды bzip2 для архивации файла: bzip2 [опции] [имена файлов]

Синтаксис команды bzip2 для разархивации/распаковки файла: bunzip2[опции]
[архивы.bz2]

```

bzip2(1)                                General Commands Manual                                bzip2(1)

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

SYNOPSIS
    bzip2 [ -cdfkqstvzVL123456789 ] [ filenames ... ]
    bzip2 [ -h|--help ]
    bunzip2 [ -fkvsVL ] [ filenames ... ]
    bunzip2 [ -h|--help ]
    bzip2recover [ -s ] [ filenames ... ]
    bzip2recover [ -h|--help ]
    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.

```

Синтаксис команды tar для архивации файла:
tar[опции][архив.tar][файлы_для_архивации]

Синтаксис команды tar для разархивации/распаковки файла: tar[опции][архив.tar]

```
TAR(1) GNU TAR Manual

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...]

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

- 2) Создала файл, в котором буду писать первый скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &»).(рис.[5])

```
emgorpinich@emgorpinich:~$ touch backup.sh
emgorpinich@emgorpinich:~$ emacs &
[1] 7835
```

- 3) Написала скрипт, который при запуске будет делать резервную копию самого себя в другую директорию back up 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.[6])

```
#!/bin/bash

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

- 4) Проверила работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Проверила, появился ли каталог backup/, перейдя в него, посмотрела его содержимое и просмотрела содержимое архива (команда «bunzip2 -cbackup.sh.bz2»). Скрипт работает

корректно. (рис.[7-8])

```
emgorpinich@emgorpinich:~$ chmod +x *.sh
emgorpinich@emgorpinich:~$ ./backup.sh
Выполнено
emgorpinich@emgorpinich:~$ cd backup
emgorpinich@emgorpinich:~/backup$ ls
backup.sh.bz2
emgorpinich@emgorpinich:~/backup$ bunzip2 -c backup.sh.bz2
#!/bin/bash

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

- 5) Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs, используя те же команды что и на 2 шаге(рис.[9])

```
emgorpinich@emgorpinich:~$ touch prog2.sh
emgorpinich@emgorpinich:~$ emacs &
[1] 8150
```

- 6) Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. (рис.[10])

```
#!/bin/bash
echo "Аргументы"
for a in $@ #Цикл для прохода по аргументам
do echo $a #Вывод аргумента
done
```

- 7) Проверила работу написанного скрипта (команды «./prog2.sh0 1 2 3 4» и «./prog2.sh0 1 2 3 45 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.[11-12])

```

emgorpinich@emgorpinich:~$ chmod +x *.sh
emgorpinich@emgorpinich:~$ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
emgorpinich@emgorpinich:~$ ./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11
Аргументы
0
1
2
3
4
5
6
7
8
9
10
11

```

- 8) Создала файл, в котором буду писать третий скрипт(шаг 2))(рис.[13])

```

emgorpinich@emgorpinich:~$ touch proglis.sh
emgorpinich@emgorpinich:~$ emacs &
[2] 8256

```

- 9) Написала командный файл – аналог команды ls. Он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях

доступа к файлам этого каталога (рис.[14])

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

    if test -f $i
    then echo "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
done
```

- 10) Далее проверила работу скрипта (команда «./progl.sh~»), предварительно добавив для него право на выполнение. Скрипт работает корректно. (рис.[15])

```
emgorpinich@emgorpinich:~$ chmod +x *.sh
emgorpinich@emgorpinich:~$ ./progl.sh ~
/home/emgorpinich/backup
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/emgorpinich/backup.sh
Файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/emgorpinich/backup.sh~
Файл
Чтение разрешено
Запись разрешена
/home/emgorpinich/Desktop
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/emgorpinich/Documents
```

- 11) Создадим файл для четвёртого скрипта(рис.[16])

```
emgorpinich@emgorpinich:~$ touch format.sh
emgorpinich@emgorpinich:~$ emacs &
[3] 8357
```

- 12) Написала командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной

строки (рис.[17])

```
#!/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
```

- 13) Проверила работу написанного скрипта (команда «./format.sh ~ pdf sh txt doc»), предварительно добавив для него право на выполнение. Скрипт работает корректно (рис.[18])

```
emgorpinich@emgorpinich:~$ chmod +x *.sh
emgorpinich@emgorpinich:~$ ./format.sh
emgorpinich@emgorpinich:~$ ./format.sh ~ pdf sh txt doc
0 файлов содержится в каталоге /home/emgorpinich с разрешением pdf
5 файлов содержится в каталоге /home/emgorpinich с разрешением sh
4 файлов содержится в каталоге /home/emgorpinich с разрешением txt
0 файлов содержится в каталоге /home/emgorpinich с разрешением doc
```

Вывод

В ходе выполнения данной лабораторной работы я познакомилась с операционной системой Linux и получила практические навыки работы с редактором Emacs.

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. C-оболочка (или csh) – надстрой-ка на оболочке Борна, использующая Сподобный синтаксис команд с возмож- ностью 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 (разработана компанией 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 мета-символ `.`, «mvaf ile{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, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать 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 (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 вво- да 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 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: 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.