Лабораторная работа №10

Ханина Людмила Константиновна

Table of Contents

Цель работы

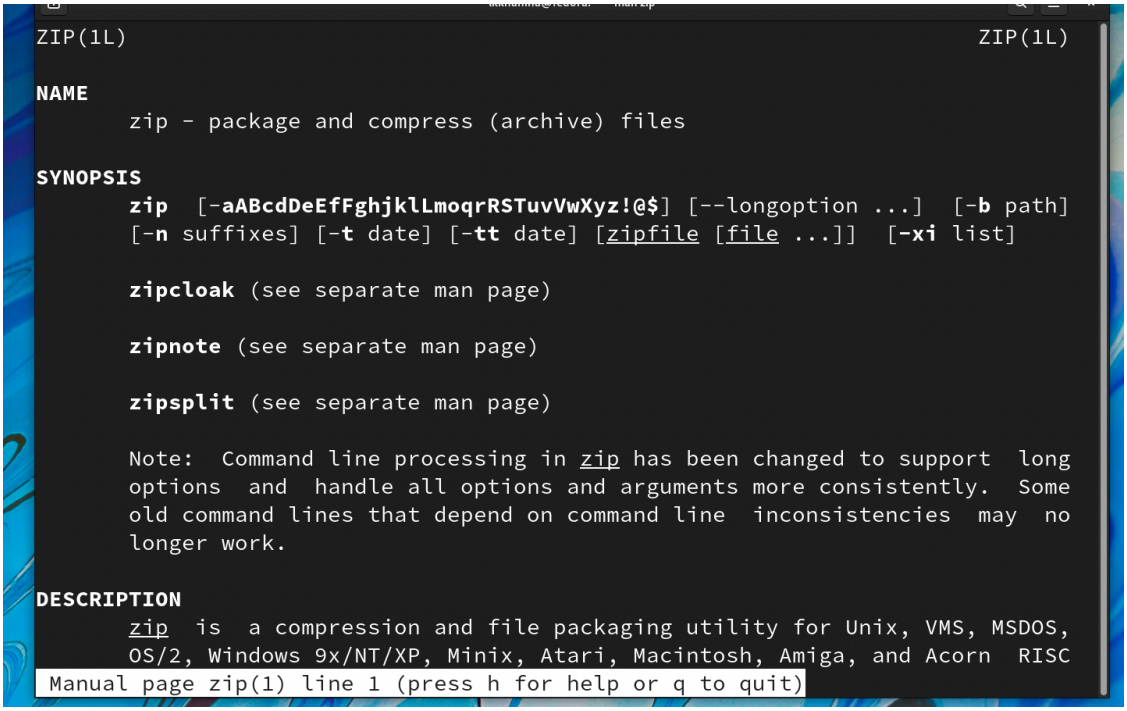
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

Задание

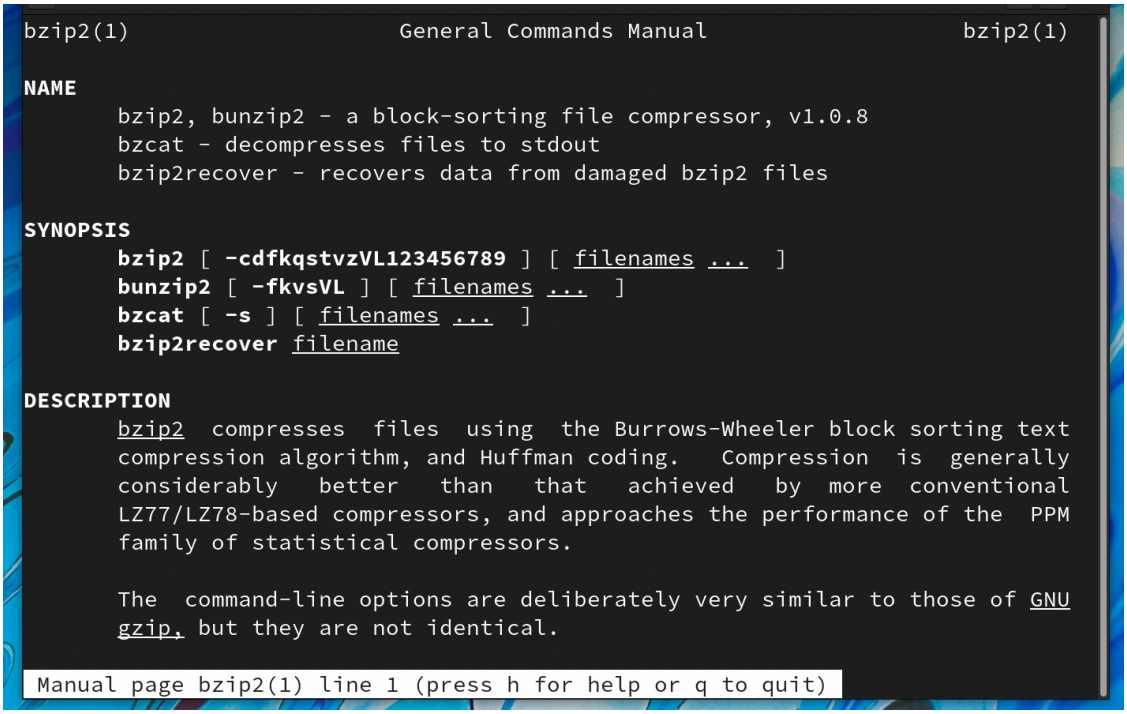
- Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
- 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.
- 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.
- Написать командный файл, который получает в качестве аргумента командно йстроки 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

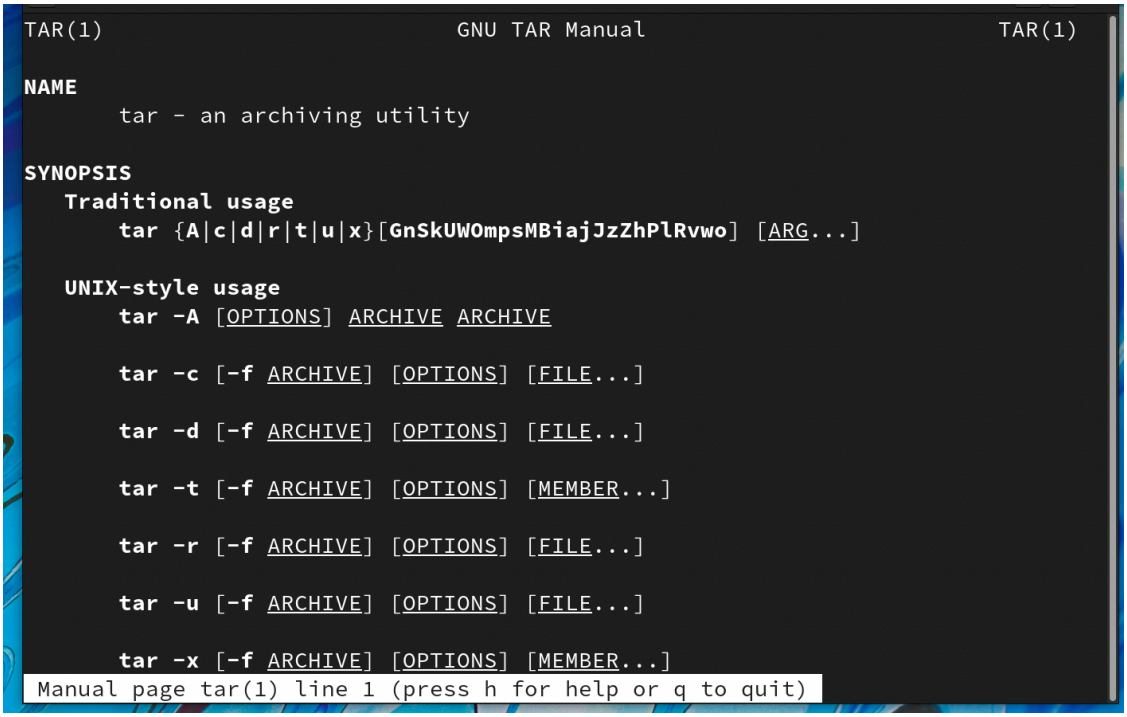
1. С помощью команды man узнаем информацию про zip, bzip2, tar.


ZIP(1L) ZIP(1L)
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)

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 [-cdfkqstvzVL123456789] [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.
Manual page bzip2(1) line 1 (press h for help or q to quit)

bzip2



tar

2. Создаем файл lab01first.sh, в котором будем писать скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в нашем домашнем каталоге. Файл будет архивироваться с помощью архиватора bzip2. Перед запуском изменим права доступа, чтобы иметь возможность запускать скрипт.

```
[lkkhanina@fedora ~]$ vi lab10first.sh
[lkkhanina@fedora ~]$ cat lab10first.sh
#!/bin/bash
name='lab10first.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Успех"
[lkkhanina@fedora ~]$ chmod +x lab10first.sh
```

Первый скрипт

3. Посмотрим корректность исполнения скрипта. Для этого зайдем в появившийся каталог backup и посмотрим текст архивированного файла. Все хорошо.

```
[lkkhanina@fedora ~]$ ./lab10first.sh
Успех
[lkkhanina@fedora ~]$ ls ~/backup
lab10first.sh.bz2
[lkkhanina@fedora ~]$
```

Запускаем первый скрипт

```
[lkkhanina@fedora ~]$ cd ~/backup
[lkkhanina@fedora backup]$ bunzip2 -c lab10first.sh.bz2
#!/bin/bash
name='lab10first.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Успех"
[lkkhanina@fedora backup]$
```

Содержание архивированного файла

4. Далее создаем файл lab10second, в котором будет второй скрипт, обрабатывающий любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов. Сразу изменим права доступа.

```
[lkkhanina@fedora backup]$ touch lab10second.sh
[lkkhanina@fedora backup]$ vi lab10second.sh
[lkkhanina@fedora backup]$ cat lab10second.sh
#!/bin/bash
echo "Печатаем аргументы:"
for arg in $@
do echo $arg
done
[lkkhanina@fedora backup]$ chmod +x lab10second.sh
```

Второй скрипт

5. Запустим скрипт и убедимся, что он исправно работает.

```
[lkkhanina@fedora backup]$ ./lab10second.sh 1 2 3 4 5
Печатаем аргументы:
1
2
3
4
5
```

Запускаем второй скрипт

6. Далее создаем файл lab10third, в котором будет третий скрипт, 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. Также изменим права доступа, чтобы суметь запустить скрипт.

```
[lkkhanina@fedora ~]$ vi lab10third.sh
[lkkhanina@fedora ~]$ chmod +x lab10third.sh
[lkkhanina@fedora ~]$ cat lab10third.sh
#!/bin/bash
a="$1"
for arg in ${a}
do
    echo "$arg"
    if test -f $arg
    then echo "File"
    fi

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

    if test -r $arg
    then echo "Can read"
    fi

    if test -w $arg
    then echo "Can write"
    fi
done
[lkkhanina@fedora ~]$
```

Третий скрипт

7. Запустим скрипт и убедимся, что он исправно работает.

```
[lkkhanina@fedora ~]$ ./lab10third.sh ~
/home/lkkhanina
Directory
Can read
Can write
[lkkhanina@fedora ~]$
```

Запускаем третий скрипт

8. Далее создаем файл lab10fourth, в котором будет четвертый скрипт, 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. Также изменим права доступа, чтобы суметь запустить скрипт.

```
[lkkhanina@fedora ~]$ vi lab10fourth.sh
[lkkhanina@fedora ~]$ chmod +x lab10fourth.sh
[lkkhanina@fedora ~]$ cat lab10fourth.sh
#!/bin/bash
a="$1"
shift
for arg in $@
do
    k=0
    for i in ${a}/*.${arg}
    do
        if test -f "$i"
        then let k=k+1
        fi
    done
    echo "total $k files that contain in directory $a with format $arg"
done
[lkkhanina@fedora ~]$
```

Четвертый скрипт

9. Запустим скрипт и убедимся, что он исправно работает.

```
done
[lkkhanina@fedora ~]$ ./lab10fourth.sh ~ txt
total 4 files that contain in directory /home/lkkhanina with format txt
[lkkhanina@fedora ~]$
```

Запускаем четвертый скрипт

Контрольные вопросы

1. Командный процессор (командная оболочка, интерпретатор команд shell) – это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В Linux доступны следующие командные оболочки:

- Bash — самая распространённая оболочка под Linux. Она ведёт историю команд и предоставляет возможность их редактирования;
- pdksh — клон korn shell, хорошо известной оболочки в системах UNIX;

- tcsh — улучшенная версия >C shell;
- zsh — новейшая из перечисленных здесь оболочек; реализует улучшенное дополнение и другие удобные функции.

2. POSIX (англ. Portable Operating System Interface — переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов. Стандарт создан для обеспечения совместимости различных UNIX-подобных операционных систем и переносимости прикладных программ на уровне исходного кода, но может быть использован и для не-Unix систем.

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. Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами.

4. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Команда read позволяет читать значения переменных со стандартного ввода.

5. В bash можно применять арифметические операции: сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).

6. В (( )) можно записывать условия оболочки bash.

7. Стандартные имена переменных: PATH, HOME, IFS, MAIL, TERM, LOGNAME.

8. Метасимволы – это специальные символы, имеющие особое значение, такие как подстановочный символ, символ повторения, символ несовпадения или диапазон символов.

9. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ’ , , ”.

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
```

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.

12. Чтобы выяснить, является ли файл каталогом или обычным файлом, можно воспользоваться командой

```
test -f [путь до файла]
```

для проверки, является ли обычным файлом и

```
test -d [путь до файла]
```

для проверки, является ли он каталогом.

13.
    - Команду set используют для вывода списка переменных окружения.
- Команду unset используют для удаления переменной из вашего окружения командной оболочки.
- Команда typeset является встроенной и предназначена для наложения ограничений на переменные.

14. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. 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 языка bash и их назначение:

- $0 — в этом переменной лежит путь и имя скрипта, который запустил пользователь;
- $# — количество параметров, переданных скрипту из командной строки;
- $? — код возврата (exit code, result code), с которым завершилась предыдущая команда;
- $! — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
- $$ - номер процесса, в котором выполняется данный скрипт.

Выводы

Я научилась писать небольшие командный файлы для решения различных вопрсов.