

Отчёт лабораторной работы №11

Дисциплина: Операционные системы

Касьянов Даниил Владимирович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	13
4	Выводы	20
5	Библиография	21

Список таблиц

Список иллюстраций

1 Цель работы

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

2 Выполнение лабораторной работы

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.

Изучаю справку по каждому из архиваторов (Рис. 1-6):

```
dvkasjyanov@dvkasjyanov:~$ man zip
```

(Рисунок 1)

```
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 ...]]
  [-xt 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(1)) unpacks zip archives. The zip and unzip(1) programs can work with archives produced by PKZIP (supporting most PKZIP features up to PKZIP version 4.0), 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 com-
Manual page zip(1) line 1 (press h for help or q to quit)
```

(Рисунок 2)

```
dvkasjyanov@dvkasjyanov:~$ man bzip2
```

(Рисунок 3)

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

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

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

    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:

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

(Рисунок 4)

```
dvkasjyanov@dvkasjyanov:~$ man tar
```

(Рисунок 5)

```
TAR(1)                                GNU TAR Manual                                TAR(1)

NAME
    tar - an archiving utility

SYNOPSIS
    Traditional usage
    tar {A|c|d|r|t|u|x}[GnSkUN0mpsMBiajJzZhPlRvw0] [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...]

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

(Рисунок 6)

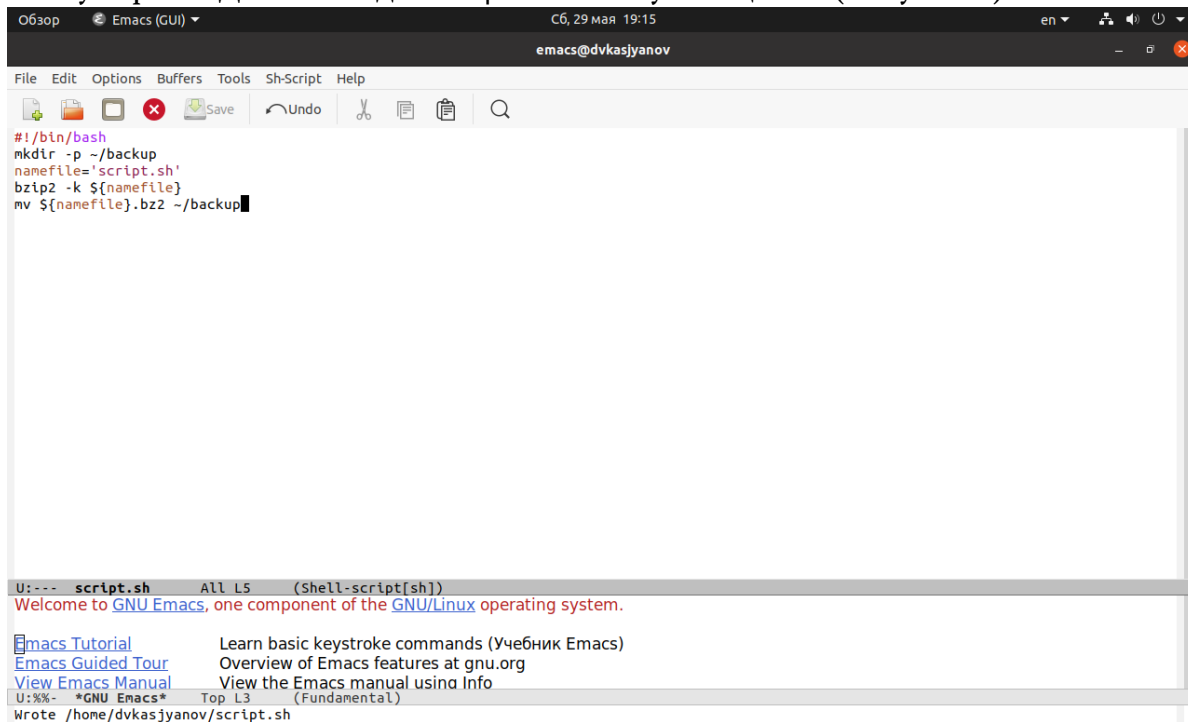
Создаю файл **script.sh**, в котором будет скрипт. Запущу его с помощью редак-

тора **emacs** в фоновом режиме (Рисунок 7):

```
dvkasjyanov@dvkasjyanov:~$ touch script.sh
dvkasjyanov@dvkasjyanov:~$ emacs script.sh &
```

(Рисунок 7)

Пишу скрипт. Для команды **bzip2** использую опцию **k** (Рисунок 8):



(Рисунок 8)

Запускаю командный файл (Рисунок 9). Проверяю корректность работы программы (Рисунок 10):

```
dvkasjyanov@dvkasjyanov:~$ bash script.sh
dvkasjyanov@dvkasjyanov:~$
dvkasjyanov@dvkasjyanov:~$ ls
1st.sh      3rd.sh~   conf.txt   ex3.sh     file.txt.save  logfile.txt  print      script.sh~  Документы  Шаблоны
1st.sh~    4th.sh   conf.txt.save  ex3.sh~    lab07.sh      main.cpp    '#PROGRAMMING.sh#'  ski.plases  Загрузки
2nd.sh     4th.sh~  date.txt      ex.sh      lab07.sh~    may        PROGRAMMING.sh      snap        Изображения
2nd.sh~    abc1     Desktop      ex.sh~     lab08        monthly    PROGRAMMING.sh~    text.txt    Музыка
'#3rd.sh#' australia ex2.sh       feathers    l.log        my_os      reports          work        Общедоступные
3rd.sh     backup   ex2.sh~      file.txt    logfile      play       script.sh          Видео      'Рабочий стол'
```

(Рисунок 9)

```
dvkasjyanov@dvkasjyanov:~$ cd backup
dvkasjyanov@dvkasjyanov:~/backup$ bunzip2 -c script.sh.bz2
#!/bin/bash
mkdir -p ~/backup
namefile='script.sh'
bzip2 -k ${namefile}
mv ${namefile}.bz2 ~/backup
dvkasjyanov@dvkasjyanov:~/backup$
```

(Рисунок 10)

2. Написать пример командного файла, обрабатывающего любое произволь-

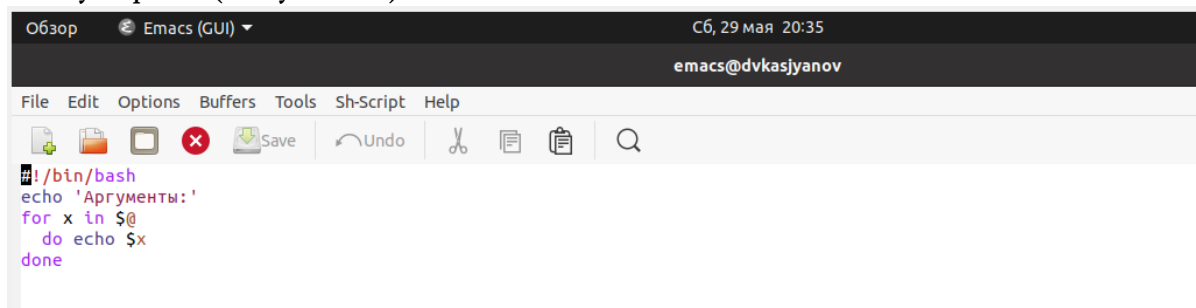
ное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.

Создаю файл **script2.sh**, в котором будет скрипт. Запущу его с помощью редактора **emacs** в фоновом режиме (Рисунок 11):

```
dvkasjyanov@dvkasjyanov:~$ touch script2.sh
dvkasjyanov@dvkasjyanov:~$ emacs script2.sh &
[1] 15593
dvkasjyanov@dvkasjyanov:~$
```

(Рисунок 11)

Пишу скрипт (Рисунок 12):



(Рисунок 12)

Проверяю работу скрипта. Запускаю командный файл, используя различное число аргументов (меньше 10, больше 10) (Рисунок 13):

```
dvkasjyanov@dvkasjyanov:~$ bash script2.sh 1 2 3 4 5 6 7
Аргументы:
1
2
3
4
5
6
7
dvkasjyanov@dvkasjyanov:~$ bash script2.sh 1 2 3 4 5 6 7 8 9 10 11 12 13
Аргументы:
1
2
3
4
5
6
7
8
9
10
11
12
13
dvkasjyanov@dvkasjyanov:~$
```

(Рисунок 13)

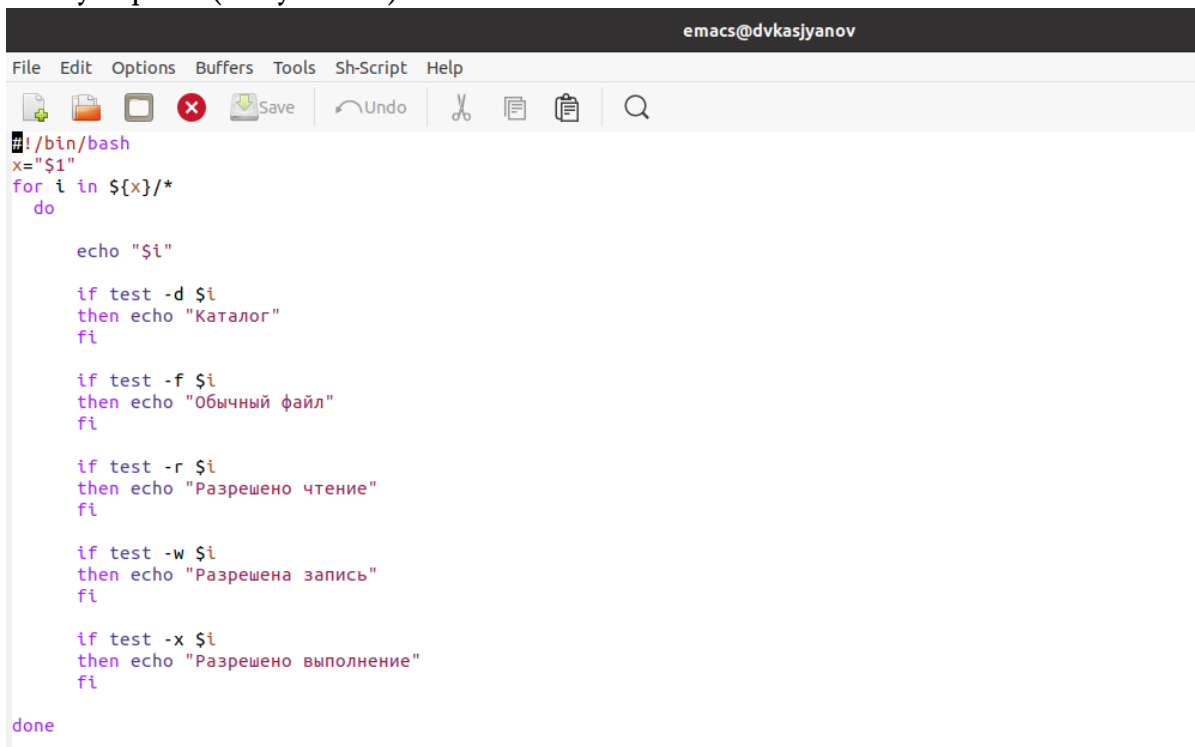
3. Написать командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

Создаю файл **script3.sh**, в котором будет скрипт. Запускаю его с помощью редактора **emacs** в фоновом режиме (Рисунок 14):

```
dvkasjyanov@dvkasjyanov:~$ touch script3.sh
dvkasjyanov@dvkasjyanov:~$ emacs script3.sh &
```

(Рисунок 14)

Пишу скрипт (Рисунок 15):



The screenshot shows the Emacs editor interface with the title bar 'emacs@dvkasjyanov'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The toolbar contains icons for file operations and editing. The main text area displays a shell script in a bash environment. The script defines a variable 'x' as '\$1', iterates over files in the directory specified by 'x', and uses 'if test' statements to check for directory, file, read, write, and execute permissions, outputting Russian labels for each. The script ends with 'done'.

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

    echo "$i"

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

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

    if test -r $i
    then echo "Разрешено чтение"
    fi

    if test -w $i
    then echo "Разрешена запись"
    fi

    if test -x $i
    then echo "Разрешено выполнение"
    fi

done
```

(Рисунок 15)

Проверяю работу скрипта (Рис. 16, 17):

```
dvkasjyanov@dvkasjyanov:~$ bash script3.sh ~
/home/dvkasjyanov/1st.sh
Обычный файл
Разрешено чтение
Разрешена запись
/home/dvkasjyanov/1st.sh~
Обычный файл
Разрешено чтение
Разрешена запись
/home/dvkasjyanov/2nd.sh
Обычный файл
Разрешено чтение
Разрешена запись
/home/dvkasjyanov/2nd.sh~
Обычный файл
Разрешено чтение
Разрешена запись
/home/dvkasjyanov/#3rd.sh#
Обычный файл
Разрешено чтение
Разрешена запись
/home/dvkasjyanov/3rd.sh
Обычный файл
Разрешено чтение
Разрешена запись
/home/dvkasjyanov/3rd.sh~
Обычный файл
Разрешено чтение
Разрешена запись
/home/dvkasjyanov/4th.sh
Обычный файл
Разрешено чтение
Разрешена запись
```

(Рисунок 16)

```
dvkasjyanov@dvkasjyanov:~$ ls -l
итого 192
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:57 1.doc
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:58 1.jpg
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:57 1.pdf
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 366 мая 22 18:44 1st.sh
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 22 18:40 1st.sh~
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:57 1.txt
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:57 2.doc
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:58 2.jpg
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 213 мая 22 18:44 2nd.sh
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 22 18:40 2nd.sh~
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:57 2.pdf
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:57 3.doc
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:58 3.jpg
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:57 3.pdf
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 496 мая 22 18:52 '#3rd.sh#'
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 497 мая 22 18:44 3rd.sh
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 22 18:40 3rd.sh~
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:57 4.doc
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:58 4.jpg
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 416 мая 22 18:44 4th.sh
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 22 18:40 4th.sh~
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 19:58 5.jpg
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 17 21:30 abc1
drwxr--r-- 2 dvkasjyanov dvkasjyanov 4096 мая 17 21:39 australia
drwxrwxr-x 2 dvkasjyanov dvkasjyanov 4096 мая 29 19:15 backup
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 446 мая 21 11:46 conf.txt
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 454 мая 21 12:19 conf.txt.save
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 33 мая 18 21:40 date.txt
drwxr-xr-x 4 dvkasjyanov dvkasjyanov 4096 мая 17 01:16 Desktop
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 97 мая 29 18:46 ex2.sh
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 18:43 ex2.sh~
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 45 мая 29 18:54 ex3.sh
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 29 18:49 ex3.sh~
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 124 мая 29 18:35 ex.sh
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 122 мая 29 18:33 ex.sh~
-rw-rw-r-- 1 dvkasjyanov dvkasjyanov 0 мая 17 21:39 feathers
```

(Рисунок 17)

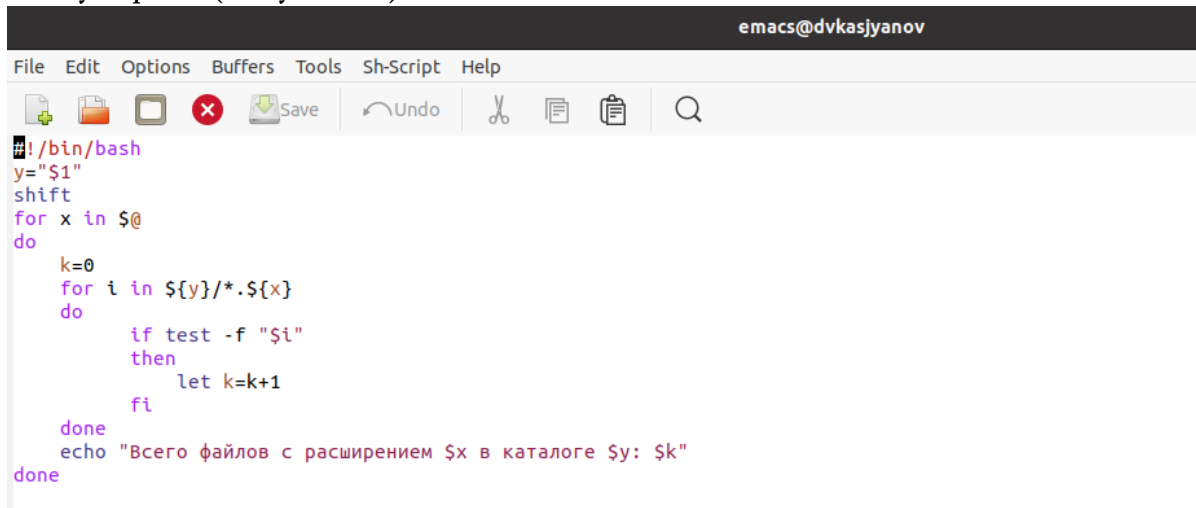
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (**.txt**, **.doc**, **.jpg**, **.pdf** и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

Создаю файл **script4.sh**, в котором будет скрипт. Запущу его с помощью редактора **emacs** в фоновом режиме (Рисунок 18):

```
dvkasjyanov@dvkasjyanov:~$ touch script4.sh
dvkasjyanov@dvkasjyanov:~$ emacs script4.sh &
```

(Рисунок 18)

Пишу скрипт (Рисунок 19):



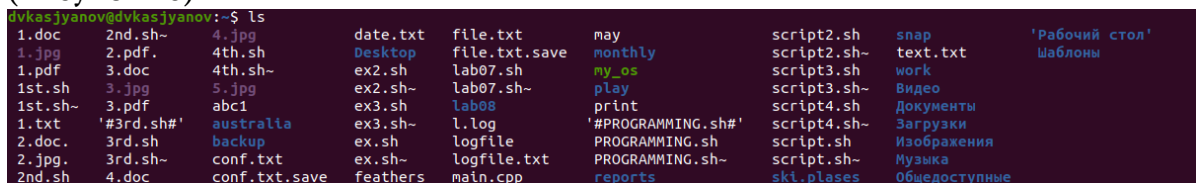
```
emacs@dvkasjyanov
File Edit Options Buffers Tools Sh-Script Help
[Icons] Save Undo [Icons]
#!/bin/bash
y="$1"
shift
for x in $@
do
    k=0
    for i in ${y}/*.${x}
    do
        if test -f "$i"
        then
            let k=k+1
        fi
    done
    echo "Всего файлов с расширением $x в каталоге $y: $k"
done
```

(Рисунок 19)

Проверяю работу скрипта (Рис. 20, 21):

```
dvkasjyanov@dvkasjyanov:~$ bash script4.sh ~ pdf doc txt jpg
Всего файлов с расширением pdf в каталоге /home/dvkasjyanov: 2
Всего файлов с расширением doc в каталоге /home/dvkasjyanov: 3
Всего файлов с расширением txt в каталоге /home/dvkasjyanov: 6
Всего файлов с расширением jpg в каталоге /home/dvkasjyanov: 4
```

(Рисунок 20)



```
dvkasjyanov@dvkasjyanov:~$ ls
1.doc      2nd.sh~   4.jpg      date.txt   file.txt    may          script2.sh  snap      'Рабочий стол'
1.jpg      2.pdf.    4th.sh     Desktop    file.txt.save  monthly      script2.sh~ text.txt   Шаблоны
1.pdf      3.doc     4th.sh~    ex2.sh     lab07.sh     my_os        script3.sh  work
1st.sh     3.jpg     5.jpg     ex2.sh~    lab07.sh~    play        script3.sh~ Видео
1st.sh~    3.pdf     abc1       ex3.sh     lab08        print       script4.sh  Документы
1.txt      '3rd.sh#' australia  ex3.sh~    l.log       '#PROGRAMMING.sh#' script4.sh~ Загрузки
2.doc      3rd.sh    backup     ex3.sh     logfile      PROGRAMMING.sh script4.sh  Изображения
2.jpg      3rd.sh~   conf.txt   ex.sh      logfile.txt  PROGRAMMING.sh~ script.sh   Музыка
2nd.sh     4.doc     conf.txt.save feathers    main.cpp     reports      script.sh~ Общедоступные
ski.places
```

(Рисунок 21)

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, содержащая базовый, но при этом полный набор функций;
 - **С-оболочка (или csh)** – надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
 - **оболочка Корна (или ksh)** – напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
 - **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, 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 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.
 - **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 >.
 - **HOME**: имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
 - **IFS**: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line).
 - **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).

- **TERM:** тип используемого терминала.
- **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 `$` является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании гделибо в командном файле комбинации символов `$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`.

4 Выводы

Я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.

5 Библиография

Лабораторная работа №11 - “Программирование в командном процессоре ОС UNIX. Командные файлы”