

Лабораторная работа №11

Дисциплина: Операционные системы

Колчева Юлия Вячеславовна

Содержание

| | | |
|---|--------------------------------|----|
| 1 | Цель работы | 5 |
| 2 | Выполнение лабораторной работы | 6 |
| 3 | Выводы | 13 |
| 4 | Контрольные вопросы | 14 |
| 5 | Библиография | 19 |

List of Tables

List of Figures

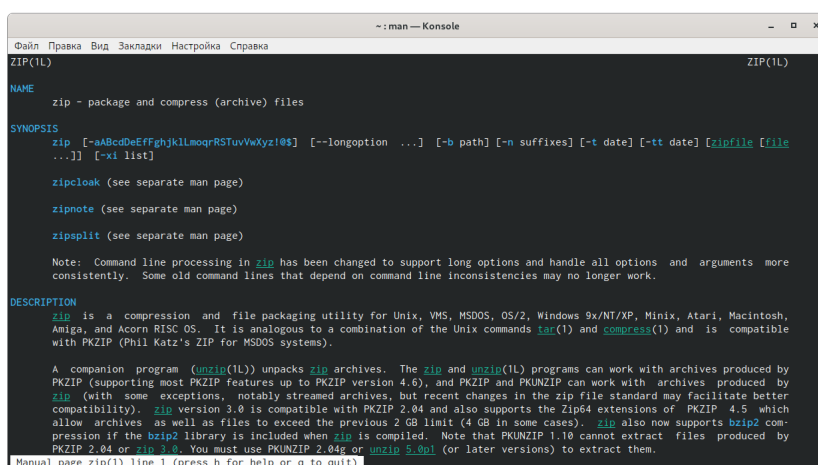
| | | |
|------|----------------------------|----|
| 2.1 | zip | 6 |
| 2.2 | bzip2 | 6 |
| 2.3 | tar | 7 |
| 2.4 | Первый скрипт | 8 |
| 2.5 | Проверка работы1 | 8 |
| 2.6 | Второй скрипт | 9 |
| 2.7 | Проверка работы2 | 10 |
| 2.8 | Третий скрипт | 10 |
| 2.9 | Проверка работы3 | 11 |
| 2.10 | Четвертый скрипт | 12 |
| 2.11 | Проверка работы4 | 12 |

1 Цель работы

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

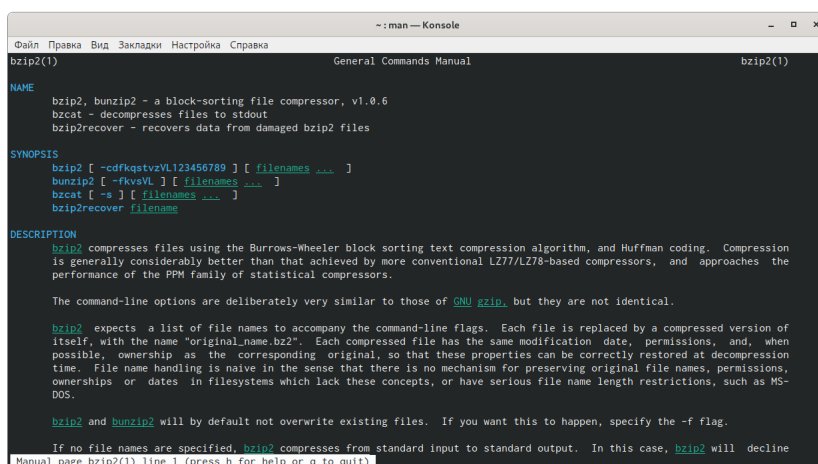
2 Выполнение лабораторной работы

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



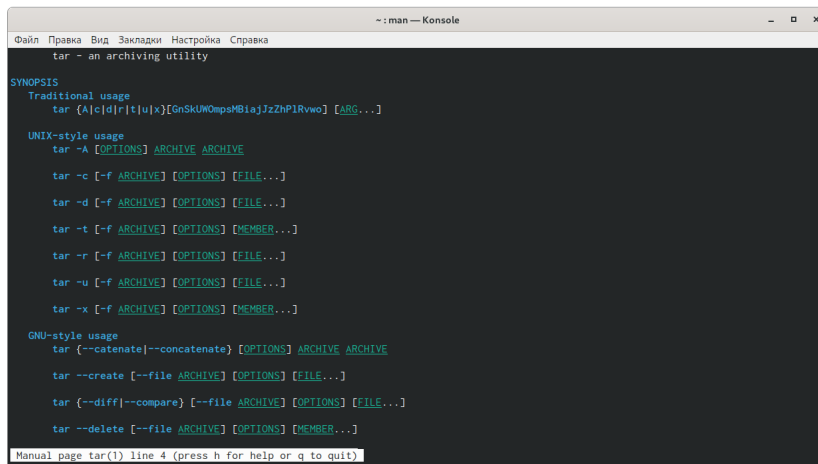
```
~: man -- Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
ZIP(1L)                                                                 ZIP(1L)
NAME
    zip - package and compress (archive) files
SYNOPSIS
    zip [-aABcdDeFFghjklLmoqrRSTuvVwXyz!0$] [--longoption ...] [-b path] [-n suffixes] [-t date] [-tt date] [zipfile [file
    ...]] [-x! 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 archives 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 com-
    pression 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 3.0b1 (or later versions) to extract them.
Manual page zip(1) line 1 (press h for help or q to quit)
```

Figure 2.1: zip



```
~: man -- Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
bzip2(1)                                                                 General Commands Manual                                                                 bzip2(1)
NAME
    bzip2, bunzip2 - a block-sorting file compressor, v1.0.6
    bzcat - decompresses files to stdout
    bzip2recover - recovers data from damaged bzip2 files
SYNOPSIS
    bzip2 [ -cdfkqstzVL123456789 ] [ 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-
    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
Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Figure 2.2: bzip2



```
~: man — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
tar - an archiving utility

SYNOPSIS
Traditional usage
tar {A|c|d|r|t|u|x}[GnSKUWmpsMBiajJzZhP1Rvwo] [ARG...]

UNIX-style usage
tar -A [OPTIONS] ARCHIVE ARCHIVE

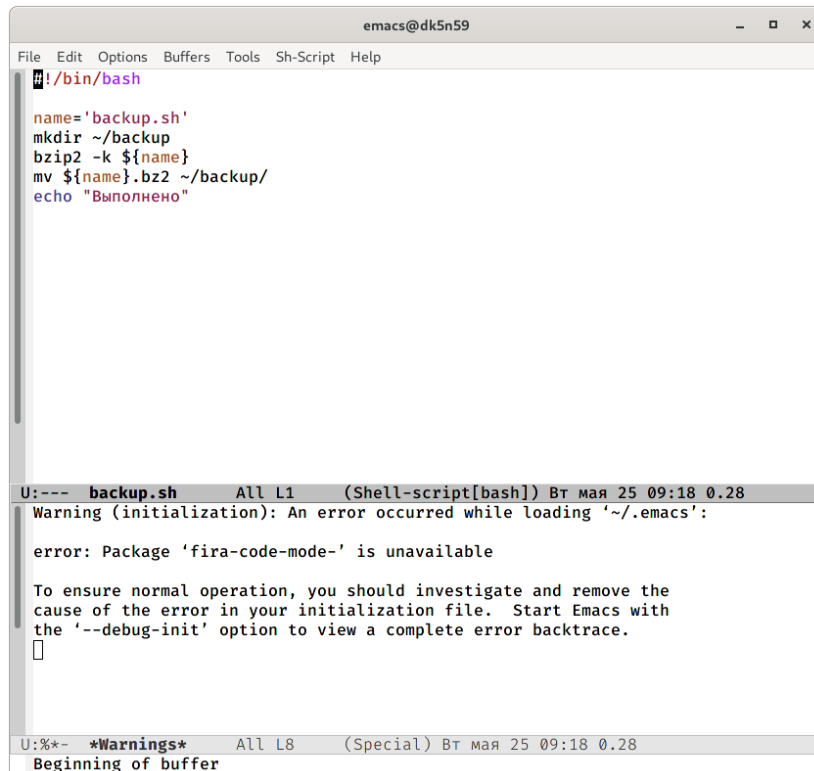
tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
tar -r [-f ARCHIVE] [OPTIONS] [FILE...]
tar -u [-f ARCHIVE] [OPTIONS] [FILE...]
tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

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...]

Manual page tar(1) line 4 (press h for help or q to quit)
```

Figure 2.3: 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 в другую директорию backup в домашнем каталоге. При написании скрипта использовала архиватор bzip2 (рис. 2.4)



```
emacs@dk5n59
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
```

U:--- backup.sh All L1 (Shell-script[bash]) Вт мая 25 09:18 0.28
Warning (initialization): An error occurred while loading '~/.emacs':

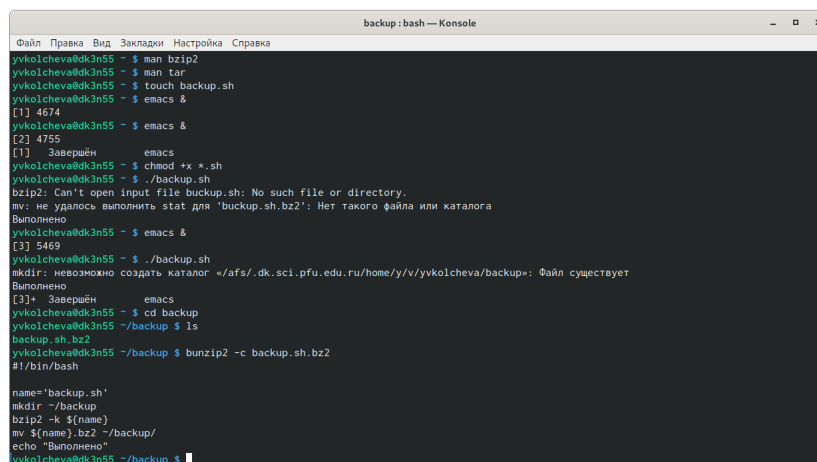
error: Package 'fira-code-mode-' is unavailable

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.
█

U:%*- *Warnings* All L8 (Special) Вт мая 25 09:18 0.28
Beginning of buffer

Figure 2.4: Первый скрипт

Проверила работу скрипта (./backup.sh), предварительно добавив для него право на выполнение (chmod+x*.sh). Проверила, появился ли каталог backup/, перейдя в него, посмотрела его содержимое и просмотрела содержимое архива (bunzip2 -c backup.sh.bz2). Скрипт работает корректно (рис. 2.5)

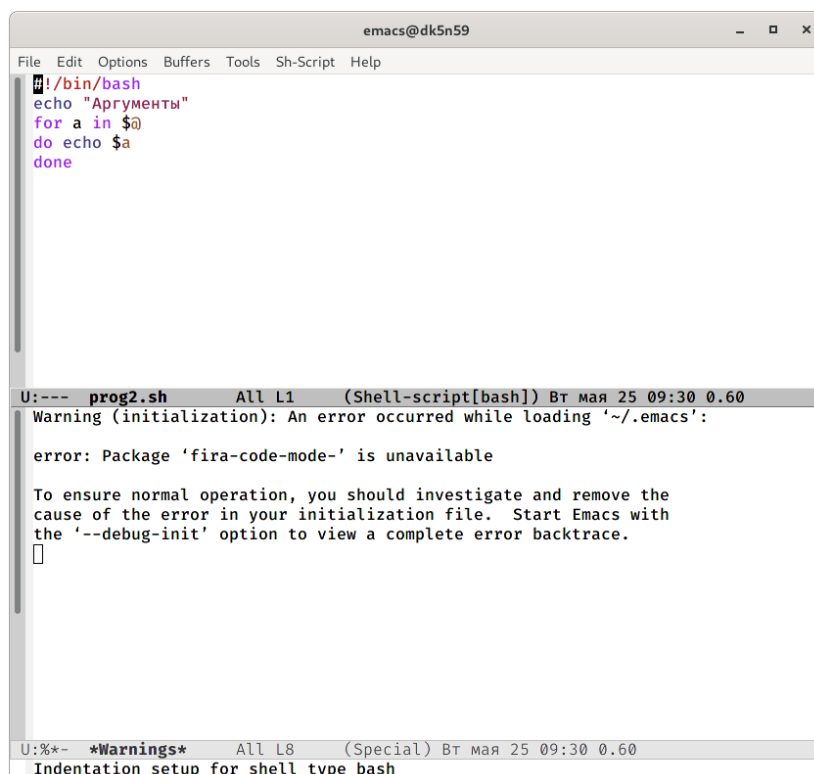


```
backup: bash — Konsole
Файл Правка Вид Закладки Настройка Справка
yvkolcheva@dk3n55 ~$ man bzip2
yvkolcheva@dk3n55 ~$ man tar
yvkolcheva@dk3n55 ~$ touch backup.sh
yvkolcheva@dk3n55 ~$ emacs &
[1] 4674
yvkolcheva@dk3n55 ~$ emacs &
[2] 4755
[1] Завершён      emacs
yvkolcheva@dk3n55 ~$ chmod +x *.sh
yvkolcheva@dk3n55 ~$ ./backup.sh
bzip2: Can't open input file backup.sh: No such file or directory.
mv: не удалось выполнить stat для 'backup.sh.bz2': Нет такого файла или каталога
Выполнено
yvkolcheva@dk3n55 ~$ emacs &
[3] 5469
yvkolcheva@dk3n55 ~$ ./backup.sh
mkdir: невозможно создать каталог «/afs/.dk.sci.pfu.edu.ru/home/y/y/yvkolcheva/backup»: Файл существует
Выполнено
[3]+ Завершён      emacs
yvkolcheva@dk3n55 ~$ cd backup
yvkolcheva@dk3n55 ~/backup$ ls
backup.sh.bz2
yvkolcheva@dk3n55 ~/backup$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
yvkolcheva@dk3n55 ~/backup$ █
```

Figure 2.5: Проверка работы1

Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs (команды «touch prog2.sh» и «emacs &»). Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. (рис. 2.6)



```
emacs@dk5n59
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
echo "Аргументы"
for a in $@
do echo $a
done

U:--- prog2.sh All L1 (Shell-script[bash]) Вт мая 25 09:30 0.60
Warning (initialization): An error occurred while loading '~/.emacs':

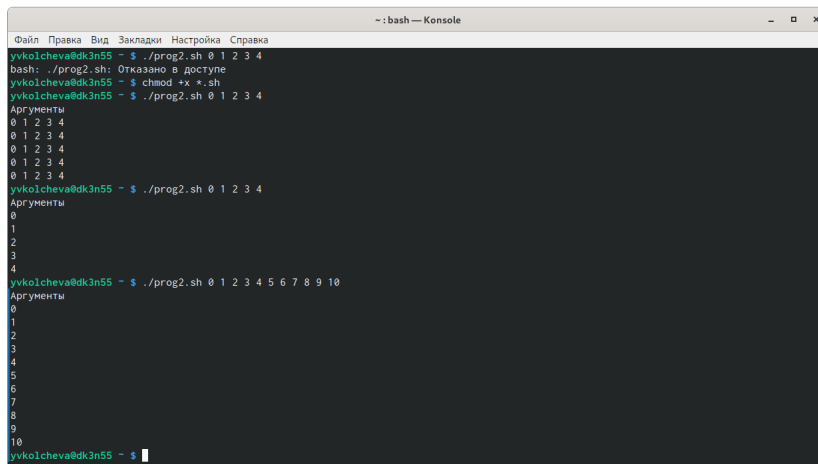
error: Package 'fira-code-mode-' is unavailable

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.

U:%*- *Warnings* All L8 (Special) Вт мая 25 09:30 0.60
Indentation setup for shell type bash
```

Figure 2.6: Второй скрипт

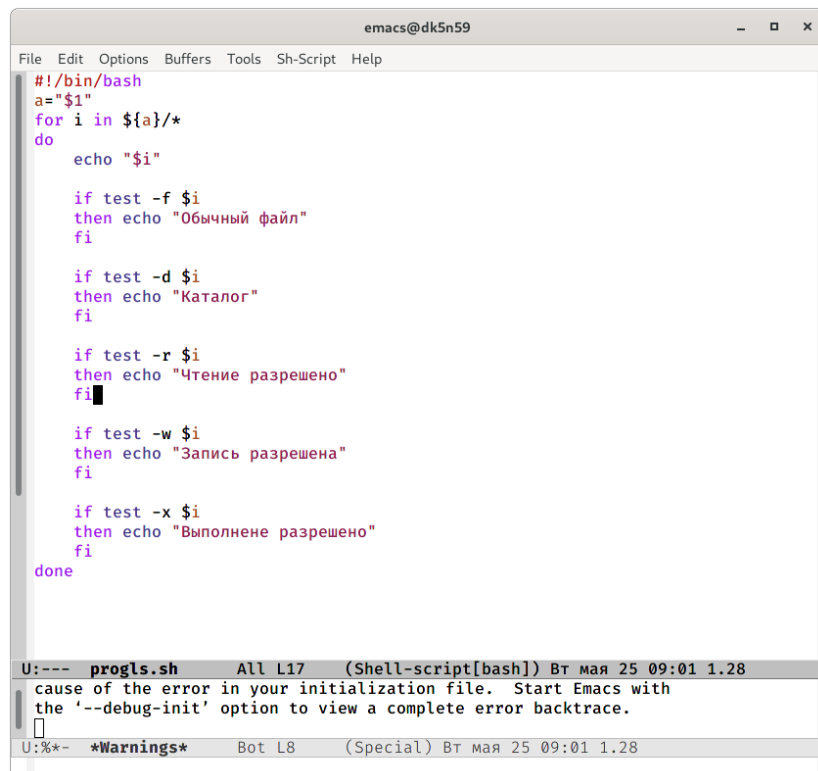
Проверила работу написанного скрипта (./prog2.sh 0 1 2 3 4 и ./prog2.sh 0 1 2 3 45 6 7 8 9 10), предварительно добавив для него право на выполнение (команда «chmod+x*.sh»). Скрипт работает корректно. (рис. 2.7)



```
~: bash — Konsole
yukoicheva@dk3n55 ~$ ./prog2.sh 0 1 2 3 4
bash: ./prog2.sh: Отказано в доступе
yukoicheva@dk3n55 ~$ chmod +x *.sh
yukoicheva@dk3n55 ~$ ./prog2.sh 0 1 2 3 4
Аргументы
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
yukoicheva@dk3n55 ~$ ./prog2.sh 0 1 2 3 4
Аргументы
0
1
2
3
4
yukoicheva@dk3n55 ~$ ./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
yukoicheva@dk3n55 ~$
```

Figure 2.7: Проверка работы2

Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs (touch progl.s.sh и emacs&) Написала командный файл-аналог команды ls. (рис. 2.8)



```
emacs@dk5n59
File Edit Options Buffers Tools Sh-Script Help
#!/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
done

U:--- progl.s.sh All L17 (Shell-script[bash]) Вт мая 25 09:01 1.28
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
U:%*- *Warnings* Bot L8 (Special) Вт мая 25 09:01 1.28
```

Figure 2.8: Третий скрипт

Далее проверила работу скрипта(./proglis.sh ~), предварительно добавив для него право на выполнение (команда «chmod+x*.sh»). Скрипт работает корректно. (рис. 2.9)

```

~: bash — Konsole
Файл Правка Вид Закладки Настройка Справка
yvkolcheva@dk5n59 ~$ chmod +x *.sh
yvkolcheva@dk5n59 ~$ ls
abc1  06.txt  file.txt  lab05.asm  labor2  proglis.sh~  text.txt  Загрузки
adc1  example1.txt  GNUstep  lab06  laboratorn8  program.asm  tmp  Изображения
asdfg  'example2.txt#'  kl  lab06.asm  may  program.lst  var  Музыка
asdfg.asm  'example2.txt'  kl.cpp  lab07  monthly  public  work  Общедоступные
australia  'example3.txt#'  kl.o  lab07.asm  my_os  public.html  yvkolch2  Презентации ОС
backup  'example3.txt'  lab02  lab07.sh  play  report7.pdf  yvkolch2.cpp  Презентация lab2.pdf
backup.sh  'example4.txt#'  lab03a  lab2  prog2.sh  reports  yvkolch2.o  'Рабочий стол'
backup.sh~  'example4.txt'  lab03b  lab2.asm  prog2.sh~  repots  Видео
conf.txt  feathers  lab05  lab.asm  proglis.sh  ski.places  Документы
yvkolcheva@dk5n59 ~$ ./proglis.sh ~
/afs/.dk.sc1.pfu.edu.ru/home/y/v/yvkolcheva/abc1
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sc1.pfu.edu.ru/home/y/v/yvkolcheva/adc1
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sc1.pfu.edu.ru/home/y/v/yvkolcheva/asdfg
Обычный файл
Чтение разрешено
Запись разрешена
Выполнение разрешено
/afs/.dk.sc1.pfu.edu.ru/home/y/v/yvkolcheva/asdfg.asm
Обычный файл
Чтение разрешено
Запись разрешена
/afs/.dk.sc1.pfu.edu.ru/home/y/v/yvkolcheva/australia
Каталог

```

Figure 2.9: Проверка работы3

Для четвертого скрипта также создала файл (touch format.sh) и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs&»). 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. 2.10)

```

#!/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 расширением $a"
done

```

U:--- **format.sh** All L15 (Shell-script[sh]) Вт мая 25 09:10 1.46
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.
U:%*- ***Warnings*** All L8 (Special) Вт мая 25 09:10 1.46
Wrote /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/format.sh

Figure 2.10: Четвертый скрипт

Проверила работу написанного скрипта (команда «./format.sh ~ pdf sh txt doc»), предварительно добавив для него право на выполнение (chmod+x*.sh), а также создав дополнительные файлы с разными расширениями. Скрипт работает корректно. (рис. 2.11)

```

./format.sh: строка 19: test: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/Рабочий: ожидается бинарный оператор
./format.sh: строка 23: test: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/Рабочий: ожидается бинарный оператор
/afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/Шаблоны
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
yvkolcheva@dk5n59 ~ $ touch Format.sh
yvkolcheva@dk5n59 ~ $ emacs &
[1] 3981
yvkolcheva@dk5n59 ~ $ chmod +x *.sh
yvkolcheva@dk5n59 ~ $ touch file.pdf file2.doc
yvkolcheva@dk5n59 ~ $ ./format.sh ~ pdf sh txt doc

./format.sh: строка 15: 3 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva с расширением pdf: Нет такого файла
или каталога
./format.sh: строка 15: 5 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva с расширением sh: Нет такого файла
или каталога
./format.sh: строка 15: 8 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva с расширением txt: Нет такого файла
или каталога
./format.sh: строка 15: 1 файл содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva с расширением doc: Нет такого файла
или каталога
yvkolcheva@dk5n59 ~ $ ./format.sh ~ pdf sh txt doc
3 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva с расширением pdf
5 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva с расширением sh
8 файлов содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva с расширением txt
1 файл содержится в каталоге /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva с расширением doc
yvkolcheva@dk5n59 ~ $

```

Figure 2.11: Проверка работы4

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: оболочка Борна (Bourne shell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; C-оболочка (или csh) – надстройка на оболочке Борна, использующая подобный синтаксис команд с возможностью сохранения истории выполнения команд; оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C 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/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 `.`, «*`mvafile{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` флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «`set-AstatesDelawareMichigan"NewJersey"`» 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 month day`» В переменные `month` и `day` будут считаны соответствующие значения, введённые с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить 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 последо-

вательность 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 (`newline`). `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 `*`, `-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+хмия_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*

5 Библиография

Лекция Кудрявцева <https://esystem.rudn.ru/mod/url/view.php?id=718563>

Лабораторная работа №11 <https://esystem.rudn.ru/mod/resource/view.php?id=718604>