

Отчёт по лабораторной работе №12

Дисциплина: Операционные системы

Джеффри Родригес Сантос

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Контрольные вопросы	16
5	Выводы	20

Список таблиц

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:

- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — вывести данные в указанный файл;
- `-р`шаблон — указать шаблон для поиска;
- `-C` — различать большие и малые буквы;
- `-n` — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом `-р`.

2. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено.

3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например `1.tmp`, `2.tmp`, `3.tmp`, `4.tmp` и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).

4. Написать командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы

запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).

3 Выполнение лабораторной работы

1. Используя команды `getopts` `grep`, написал командный файл, который анализирует командную строку с ключами:

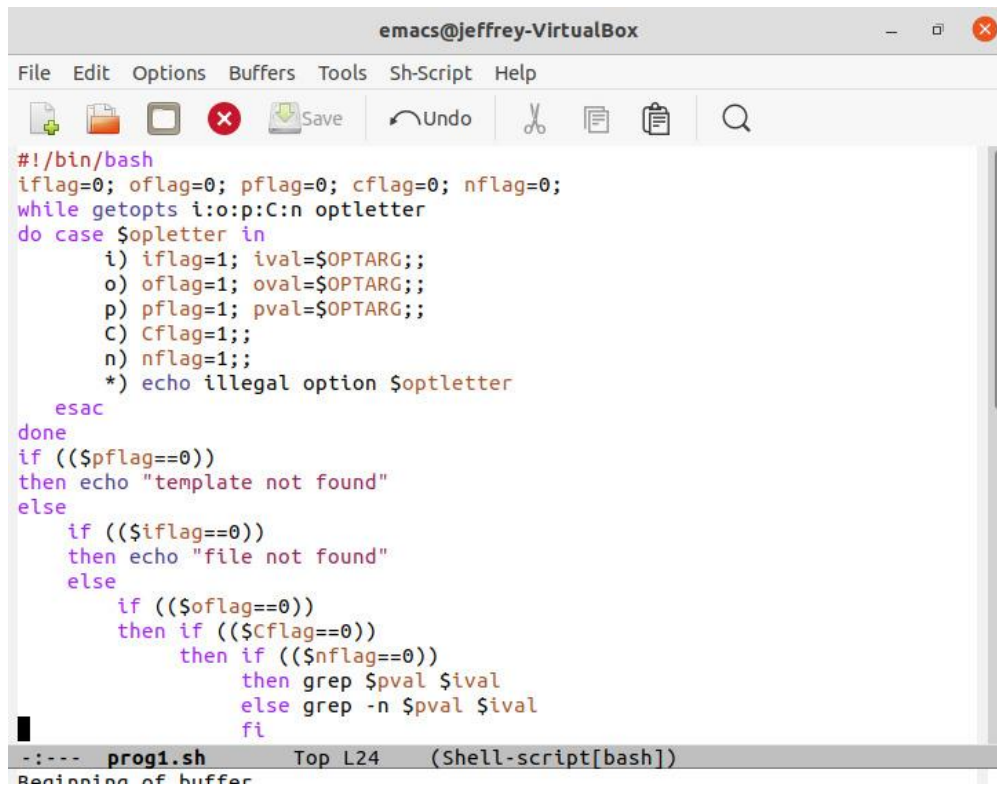
- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — вывести данные в указанный файл;
- `-р` шаблон — указать шаблон для поиска;
- `-C` — различать большие и малые буквы;
- `-n` — выдавать номера строк,

а затем ищет в указанном файле нужные строки, определяемые ключом `-р`.

Для данной задачи я создал файл `prog1.sh` (рис. -fig. 3.1) и написал соответствующие скрипты (рис. -fig. 3.2) (рис. -fig. 3.3).

```
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ touch prog1.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ emacs
```

Рис. 3.1: Создал файл для первого скрипта



```
#!/bin/bash
iflag=0; oflag=0; pflag=0; cflag=0; nflag=0;
while getopts i:o:p:C:n optletter
do case $optletter in
    i) iflag=1; ival=$OPTARG;;
    o) oflag=1; oval=$OPTARG;;
    p) pflag=1; pval=$OPTARG;;
    C) cflag=1;;
    n) nflag=1;;
    *) echo illegal option $optletter
    esac
done
if (($pflag==0))
then echo "template not found"
else
    if (($iflag==0))
    then echo "file not found"
    else
        if (($oflag==0))
        then if (($cflag==0))
            then if (($nflag==0))
                then grep $pval $ival
            else grep -n $pval $ival
            fi
        fi
    fi
fi
```

--:-- prog1.sh Top L24 (Shell-script[bash])

Рис. 3.2: Написал первый скрипт



```
then if (($nflag==0))
then grep $pval $ival
else grep -n $pval $ival
fi
else if (($nflag==0))
then grep -i $pval $ival
else grep -i -n $pval $ival
fi
fi
else if (($cflag==0))
then if (($nflag==0))
then grep $pval $ival > $oval
else grep -n $pval $ival > $oval
fi
else if (($nflag==0))
then grep -i $pval $ival > $oval
else grep -i -n $pval $ival > $oval
fi
fi
fi
fi
```

--:-- prog1.sh Bot L24 (Shell-script[bash])

Рис. 3.3: Написал первый скрипт

Далее я проверил работу написанного скрипта, используя различные опции (например, команда «./prog.sh -i ~/a.txt -o ~/b.txt -p song -C -n»), предварительно добавив право на исполнение файла (команда «chmod +x .sh») и создав 2 файла, которые необходимы для выполнения программы: a.txt и b.txt (рис. -fig. 3.4).

Скрипт работает корректно.

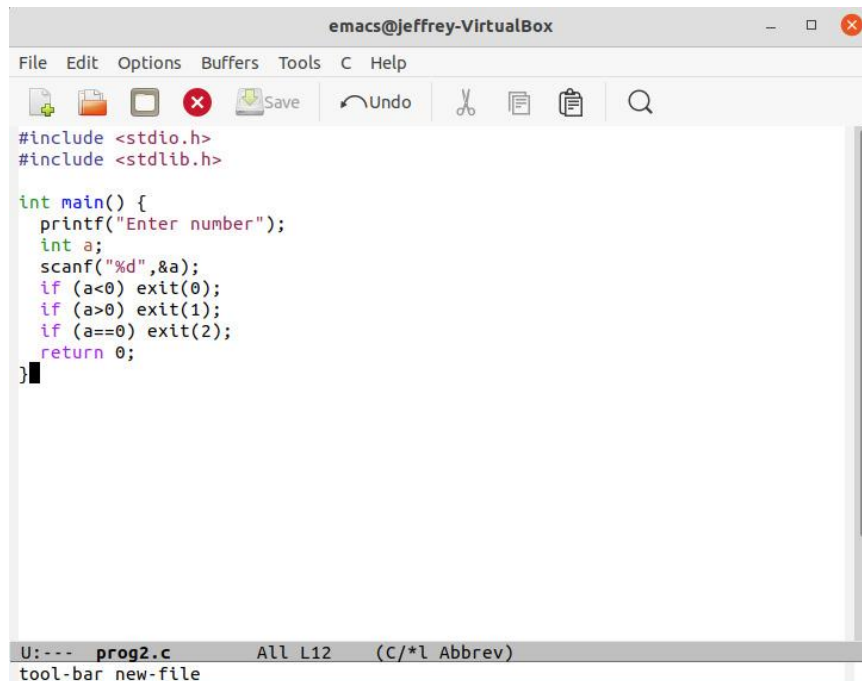
```
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ chmod +x *.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ cat ~/a.txt
cause you're right outside, let you up
lay you down, it's just us
he lost your touch, all to me , all to me
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ ./prog1.sh -i ~/a.txt -o ~/b.txt -p to me -C -n
illegal option i
illegal option o
illegal option p
template not found
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ chmod +x *.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ cat ~/a.txt
David is a song by Jeffrey
Hello
Youngster is a song by Alex Max
Show is Song by Morgana
Blur is a song by LXST CXNTURY
```

Рис. 3.4: Проверил первый скрипт

2. Написал на языке Си программу, которая вводит число и определяет, являющееся ли оно больше нула, равно нулю, меньше нуля, затем программа завершается с помощью функции exit (n), передается информация о коде завершения в оболочку. Для данной задачи я создал 2 файла: prog2.c и prog2.sh (рис. -Fig. 3.5) и написал соответствующие скрипты (рис. 3.6).

```
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ touch prog2.c prog2.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ emacs &
```

Рис. 3.5: Создал файл для второго скрипта



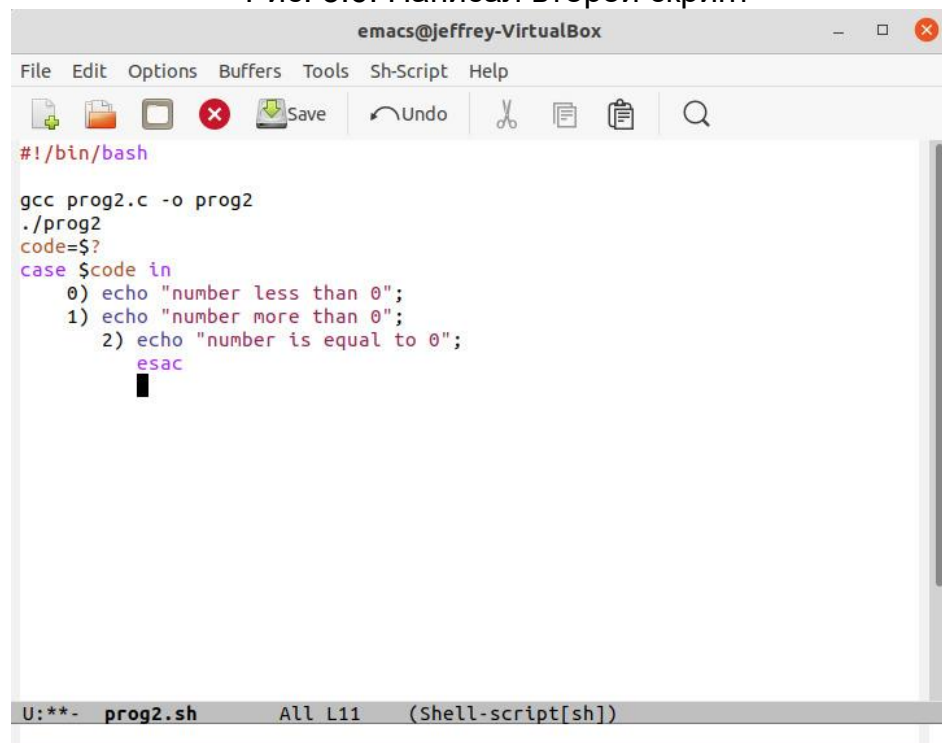
The screenshot shows the Emacs editor window titled "emacs@Jeffrey-VirtualBox". The menu bar includes File, Edit, Options, Buffers, Tools, C, and Help. The toolbar contains icons for saving, undo, and search. The code in the buffer is as follows:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("Enter number");
    int a;
    scanf("%d",&a);
    if (a<0) exit(0);
    if (a>0) exit(1);
    if (a==0) exit(2);
    return 0;
}
```

The status bar at the bottom indicates the file is "prog2.c", line 12, column 1, with the mode set to "(C/*l Abbrev)".

Рис. 3.6: Написал второй скрипт



The screenshot shows the Emacs editor window titled "emacs@Jeffrey-VirtualBox". The menu bar includes File, Edit, Options, Buffers, Tools, Sh-Script, and Help. The toolbar contains icons for saving, undo, and search. The code in the buffer is as follows:

```
#!/bin/bash

gcc prog2.c -o prog2
./prog2
code=$?
case $code in
    0) echo "number less than 0";
    1) echo "number more than 0";
    2) echo "number is equal to 0";
    esac
```

The status bar at the bottom indicates the file is "prog2.sh", line 11, column 1, with the mode set to "(Shell-script[sh])".

Рис. 3.7: Написал второй скрипт

Далее я проверил работу написанных скриптов (команда «./prog2.sh»), предварительно добавив право на исполнение файла (команда «chmod +x .sh») (рис.-fig. 3.8). Скрипты работают корректно.

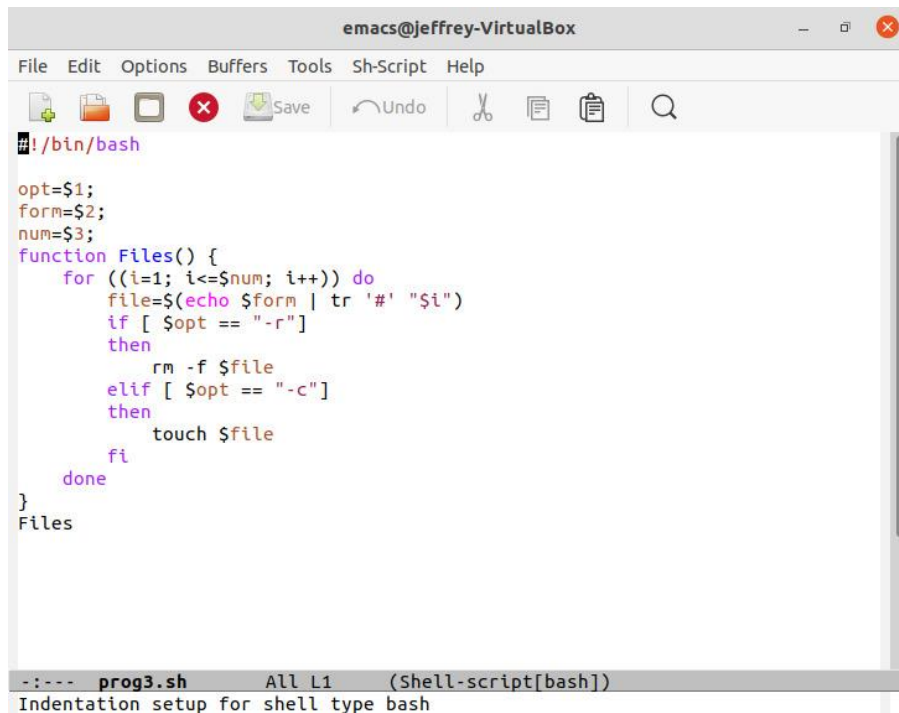
```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$ chmod +x *.sh
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$ ./prog2.sh
Enter number2
number more than 0
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$ ./prog2.sh
Enter number-2
number less than 0
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$ ./prog2.sh
Enter number0
number is equal to
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$
```

Рис. 3.8: Проверил второй скрипт

2. Написал командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т. д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют). Для данной задачи я создал файл: prog3.sh (рис. -fig. 3.9) и написал соответствующий скрипт (рис. -fig. 3.10).

```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$ touch prog3.sh
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$ emacs &
```

Рис. 3.9: Создал файл для третьего скрипта



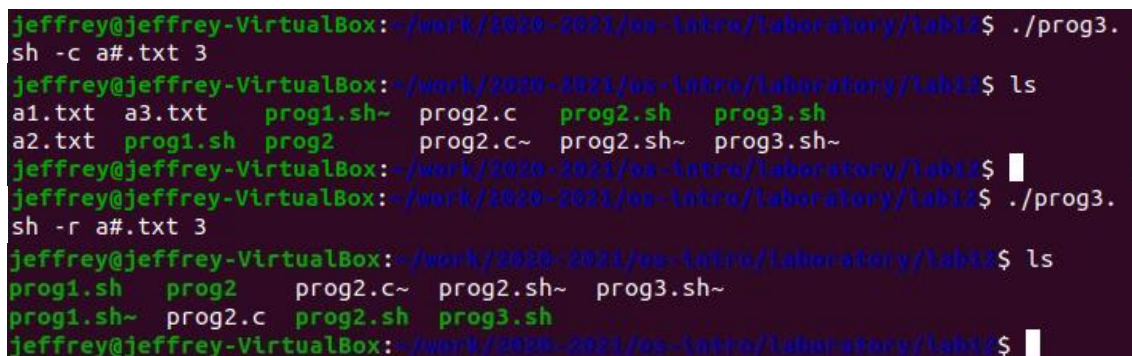
```
#!/bin/bash

opt=$1;
form=$2;
num=$3;
function Files() {
  for ((i=1; i<=$num; i++)) do
    file=$(echo $form | tr '#' "$i")
    if [ $opt == "-r" ]
    then
      rm -f $file
    elif [ $opt == "-c" ]
    then
      touch $file
    fi
  done
}
Files
```

--- prog3.sh All L1 (Shell-script[bash])
Indentation setup for shell type bash

Рис. 3.10: Написал третий скрипт

Далее я проверил работу написанного скрипта (команда «./files.sh»), предварительно добавив право на исполнение файла (команда «chmod +x .sh»). Сначала я создал три файла (команда «./files.sh -c a#.txt 3»), удовлетворяющие условию задачи, а потом удалил их (команда «./files.sh -r a#.txt 3») (рис. -fig. 3.11). Скрипт работает корректно.



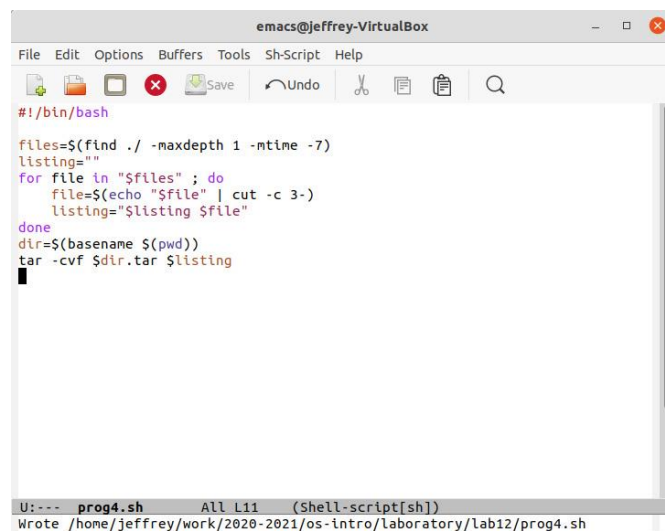
```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/oa-intro/laboratory/lab12$ ./prog3.sh -c a#.txt 3
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/oa-intro/laboratory/lab12$ ls
a1.txt  a3.txt  prog1.sh~  prog2.c  prog2.sh  prog3.sh
a2.txt  prog1.sh  prog2     prog2.c~  prog2.sh~  prog3.sh~
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/oa-intro/laboratory/lab12$ ./prog3.sh -r a#.txt 3
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/oa-intro/laboratory/lab12$ ls
prog1.sh  prog2  prog2.c~  prog2.sh~  prog3.sh~
prog1.sh~  prog2.c  prog2.sh  prog3.sh
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/oa-intro/laboratory/lab12$
```

Рис. 3.11: Проверил третий скрипт

3. Написал командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировала его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). Для данной задачи я создал файл: prog4.sh (рис. -fig. 3.12) и написал соответствующий скрипт (рис. -fig. 3.13).

```
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$ touch prog4.sh
jeffrey@jeffrey-VirtualBox: ~/work/2020-2021/os-intro/laboratory/lab12$ emacs &
```

Рис. 3.12: Создал файл для четвёртого скрипта



```
emacs@jeffrey-VirtualBox
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash

files=$(find ./ -maxdepth 1 -mtime -7)
listing=""
for file in "$files" ; do
    file=$(echo "$file" | cut -c 3-)
    listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing
```

U:-- prog4.sh All L11 (Shell-script[sh])
Wrote /home/jeffrey/work/2020-2021/os-intro/laboratory/lab12/prog4.sh

Рис. 3.13: Написал четвёртый скрипт

Далее я проверил работу написанного скрипта (команды «`sudo ~/work/2020-2021/os-intro/laboratory/lab12/prog4.sh`» и «`tar -tf lab12.tar`»), предварительно добавив право на исполнение файла (команда «`chmod +x .sh`») (рис. -fig. 3.14) (рис.-fig. 3.15). Скрипт работает корректно.

```
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ chmod +x
prog4.sh
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ ls -l
total 44
-rwxrwxr-x 1 jeffrey jeffrey 921 мая 29 20:32 prog1.sh
-rwxrwxr-x 1 jeffrey jeffrey 120 мая 29 20:17 prog1.sh~
-rwxrwxr-x 1 jeffrey jeffrey 16840 мая 29 21:02 prog2
-rw-rw-r-- 1 jeffrey jeffrey 181 мая 29 20:51 prog2.c
-rw-rw-r-- 1 jeffrey jeffrey 0 мая 29 20:45 prog2.c~
-rwxrwxr-x 1 jeffrey jeffrey 192 мая 29 21:01 prog2.sh
-rw-rw-r-- 1 jeffrey jeffrey 0 мая 29 20:45 prog2.sh~
-rwxrwxr-x 1 jeffrey jeffrey 234 мая 29 21:14 prog3.sh
-rw-rw-r-- 1 jeffrey jeffrey 0 мая 29 21:03 prog3.sh~
-rwxrwxr-x 1 jeffrey jeffrey 210 мая 29 21:24 prog4.sh
-rw-rw-r-- 1 jeffrey jeffrey 0 мая 29 21:20 prog4.sh~
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ sudo ~/w
ork/2020-2021/os-intro/laboratory/lab12/prog4.sh
[sudo] password for jeffrey:
prog3.sh
prog4.sh
prog1.sh~
prog2.c~
prog4.sh~
prog2.sh
prog2.sh~
prog2.c
prog3.sh~
prog1.sh
prog2
[1]+ Done emacs
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$
```

Рис. 3.14: Проверил четвёртый скрипт

```
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$ tar -tf
lab12.tar
prog3.sh
prog4.sh
prog1.sh~
prog2.c~
prog4.sh~
prog2.sh
prog2.sh~
prog2.c
prog3.sh~
prog1.sh
prog2
[1]+ Done emacs
jeffrey@jeffrey-VirtualBox:~/work/2020-2021/os-intro/laboratory/lab12$
```

Рис. 3.15: Проверил четвёртый скрипт

4 Контрольные вопросы

1. Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды, следующий:

`getopts option-string variable [arg ...]`

Флаги — это опции командной строки, обычно помеченные знаком минус; например, для команды `ls` флагом может являться `-F`.

Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, то она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введённые данные с помощью оператора `case`.

Функция `getopts` включает две специальные переменные среды — `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента.

Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введённых пользователем данных.

2. При перечислении имён файлов текущего каталога можно использовать следующие символы:

- `-` — соответствует произвольной, в том числе и пустой строке;

- `?` – соответствует любому одинарному символу;
- `[c1-c2]` – соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`. Например,
- `echo *` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`;
- `ls *.c` – выведет все файлы с последними двумя символами, совпадающими с `.c`.
- `echo prog.?` – выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.`.
- `[a-z]` – соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.

3. Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некоторого условия. Для решения подобных задач язык программирования `bash` предоставляет возможность использовать такие управляющие конструкции, как `for`, `case`, `if` и `while`. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда.

Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных

файлах. Единственная функция этой команды заключается в выработке кода завершения.

4. Два несложных способа позволяют вам прерывать циклы в оболочке bash. Команда `break` завершает выполнение цикла, а команда `continue` завершает данную итерацию блока операторов.

Команда `break` полезна для завершения цикла `while` в ситуациях, когда условие перестаёт быть правильным.

Команда `continue` используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить проверять данный блок на других условных выражениях.

5. Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования bash: это команда `true`, которая всегда возвращает код завершения, равный нулю (т. е. истина), и команда `false`, которая всегда возвращает код завершения, не равный нулю (т. е. ложь).

Примеры бесконечных циклов:

```
while true
do echo hello andy
done
until false
do echo hello mike
done
```

6. Строка `if test -f man.s/i.s, mans/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).
7. Выполнение оператора цикла `while` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой

код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `do`, после чего осуществляется безусловный переход на начало оператора цикла `while`. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, возвратит ненулевой код завершения (ложь).

При замене в операторе цикла `while` служебного слова `while` на `until` условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла `while` и оператор цикла `until` идентичны.

5 Выводы

В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX и научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.