

Отчет по лабораторной работе №12

Дисциплина: Операционные системы

Калистратова Ксения Евгеньевна

Содержание

| | |
|--------------------------------------|----|
| Цель работы | 1 |
| Задачи..... | 1 |
| Выполнение лабораторной работы | 1 |
| Контрольные вопросы | 8 |
| Выводы | 10 |
| Библиография..... | 10 |

Цель работы

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

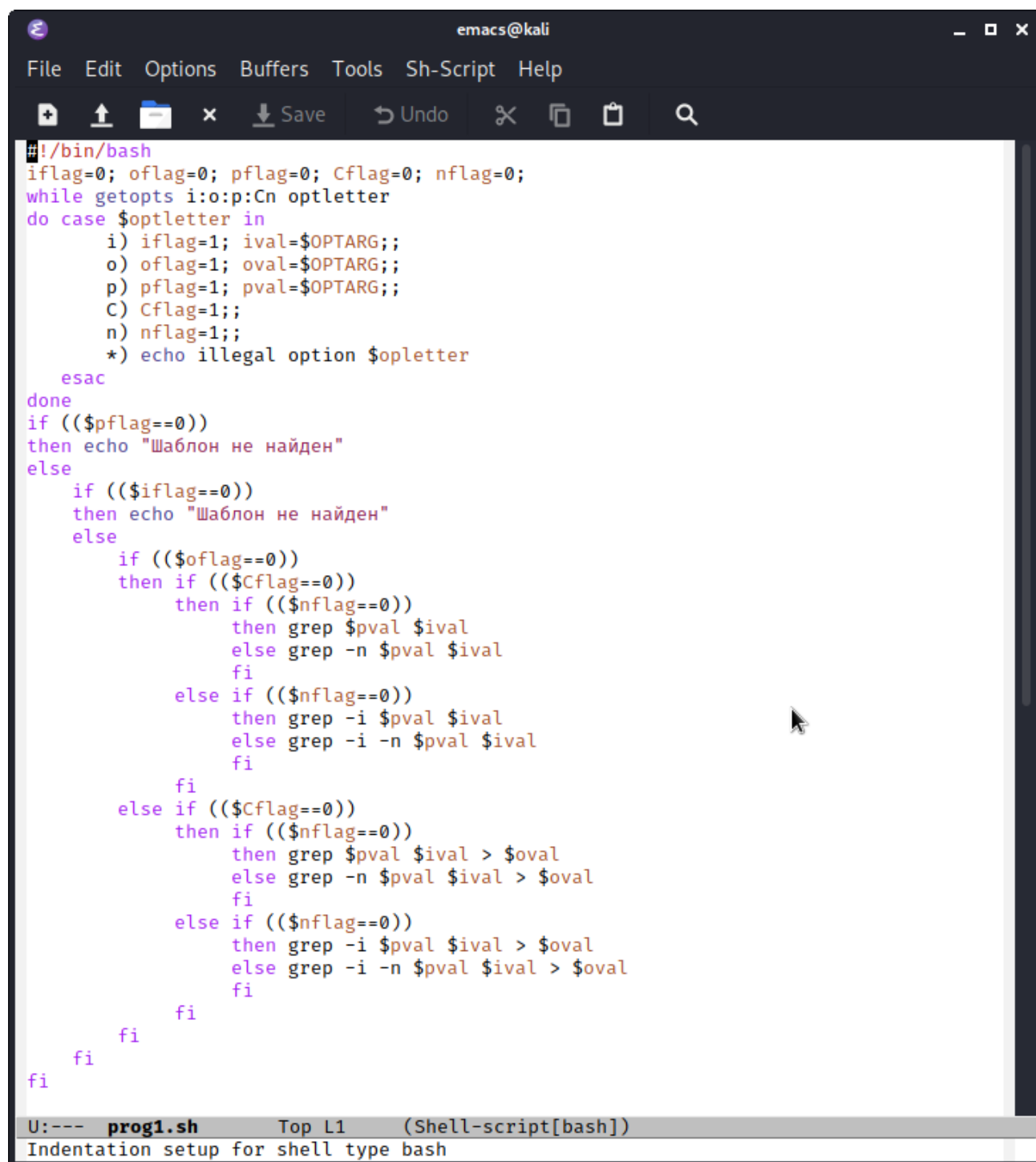
Задачи

1. Познакомиться с логическими управляющими конструкций и циклов.
2. В ходе работы написать 4 командных файла.
3. Выполнить отчет.

Выполнение лабораторной работы

- 1) Используя команды `getopts` и `grep`, написала командный файл, который анализирует командную строку с ключами:
 - `-i input file` — прочитать данные из указанного файла;
 - `-o out put file` — вывести данные в указанный файл;
 - `-p 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 `-p`.

Для данной задачи я создала файл prog1.sh и написала соответствующие скрипты. (рис. 1)



```
#!/bin/bash
iflag=0; oflag=0; pflag=0; Cflag=0; nflag=0;
while getopts i:o:p:Cn 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 "Шаблон не найден"
else
    if (($iflag==0))
    then echo "Шаблон не найден"
    else
        if (($oflag==0))
        then if (($Cflag==0))
            then if (($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
fi
fi
```

U:--- prog1.sh Top L1 (Shell-script[bash])
Indentation setup for shell type bash

Figure 1: Первый скрипт

Далее я проверила работу написанного скрипта, используя различные опции (например, команда «./prog1.sh -i a1.txt -o a2.txt -p capital -C -n»), предварительно добавив право на исполнение файла (команда «chmod +x prog1.sh») и создав 2 файла,

которые необходимы для выполнения программы: a1.txt и a2.txt. Скрипт работает корректно. (рис. 2):

Сфе./tou

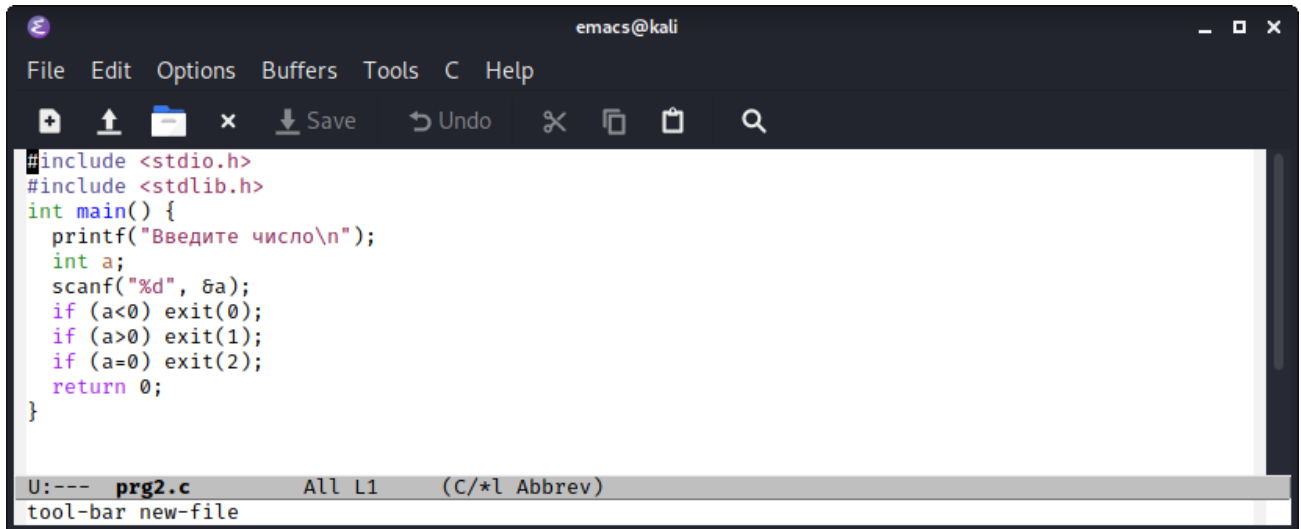


```
kekalistratova@kekalistratova: ~  
Файл Действия Правка Вид Справка  
Шаблон не найден  
(kekalistratova@kekalistratova)-[~]  
$ touch prog1.sh  
(kekalistratova@kekalistratova)-[~]  
$ emacs &  
[1] 1512  
(kekalistratova@kekalistratova)-[~]  
$ touch a1.txt a2.txt  
(kekalistratova@kekalistratova)-[~]  
$ chmod +x prog1.sh  
(kekalistratova@kekalistratova)-[~]  
$ cat a1.txt  
A crime fiction is a story that focuses on a criminal act and on a following in\  
vestigation1.  
Usually done from a point of view of either a detective or their assistant, cri\  
me fiction spans over many types of media.  
Usually it takes the form of either a novel or a movie  
(kekalistratova@kekalistratova)-[~]  
$ ./prog1.sh -i a1.txt -o a2.txt -p a -C -n  
(kekalistratova@kekalistratova)-[~]  
$ cat a2.txt  
1:A crime fiction is a story that focuses on a criminal act and on a following in\  
2:vestigation1.  
3:Usually done from a point of view of either a detective or their assistant, cri\  
4:me fiction spans over many types of media.  
5:Usually it takes the form of either a novel or a movie  
(kekalistratova@kekalistratova)-[~]  
$ ./prog1.sh -i a1.txt -o a2.txt -p a -n  
(kekalistratova@kekalistratova)-[~]  
$ cat a2.txt  
1:A crime fiction is a story that focuses on a criminal act and on a following in\  
2:vestigation1.  
3:Usually done from a point of view of either a detective or their assistant, cri\  
4:me fiction spans over many types of media.  
5:Usually it takes the form of either a novel or a movie  
(kekalistratova@kekalistratova)-[~]  
$ ./prog1.sh -i a1.txt -C -n  
Шаблон не найден  
(kekalistratova@kekalistratova)-[~]  
$ ./prog1.sh -o a2.txt -p crime -C -n  
Файл не найден
```

Figure 2: Проверка работы скрипта

- 2) Написана на языке Си программа, 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

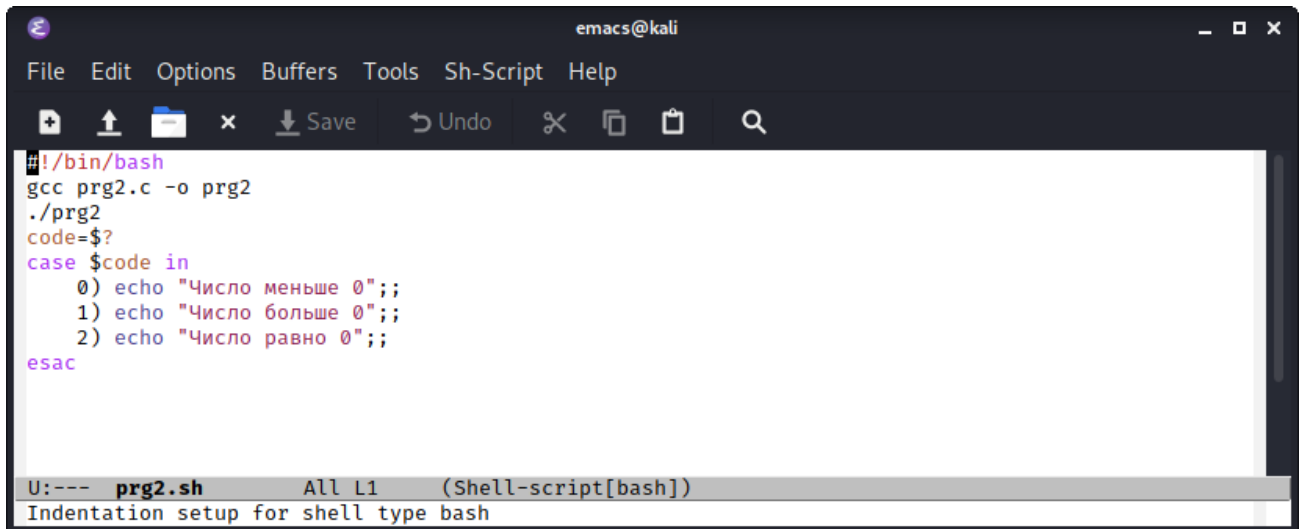
оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено. Для данной задачи я создала 2 файла: prg2.c и prg2.sh и написала соответствующие скрипты. (рис. 3) (рис. 4)



```
#include <stdio.h>
#include <stdlib.h>
int main() {
    printf("Введите число\n");
    int a;
    scanf("%d", &a);
    if (a<0) exit(0);
    if (a>0) exit(1);
    if (a=0) exit(2);
    return 0;
}
```

U:--- prg2.c All L1 (C/*l Abbrev)
tool-bar new-file

Figure 3: Файл prg2.c



```
#!/bin/bash
gcc prg2.c -o prg2
./prg2
code=$?
case $code in
    0) echo "Число меньше 0";;
    1) echo "Число больше 0";;
    2) echo "Число равно 0";;
esac
```

U:--- prg2.sh All L1 (Shell-script[bash])
Indentation setup for shell type bash

Figure 4: Файл prg2.sh

Далее я проверила работу написанных скриптов (команда «./prg2.sh»), предварительно добавив право на исполнение файла (команда «chmod +x prg2.sh»). Скрипты работают корректно. (рис. 5)

```
kekalistratova@kali: ~/Catalog1
Файл Действия Правка Вид Справка

(kekalistratova@kali)-[~]
$ chmod +x prg2.sh

(kekalistratova@kali)-[~]
$ ./prg2.sh строку с ключами:
Введите число
8
Число больше 0

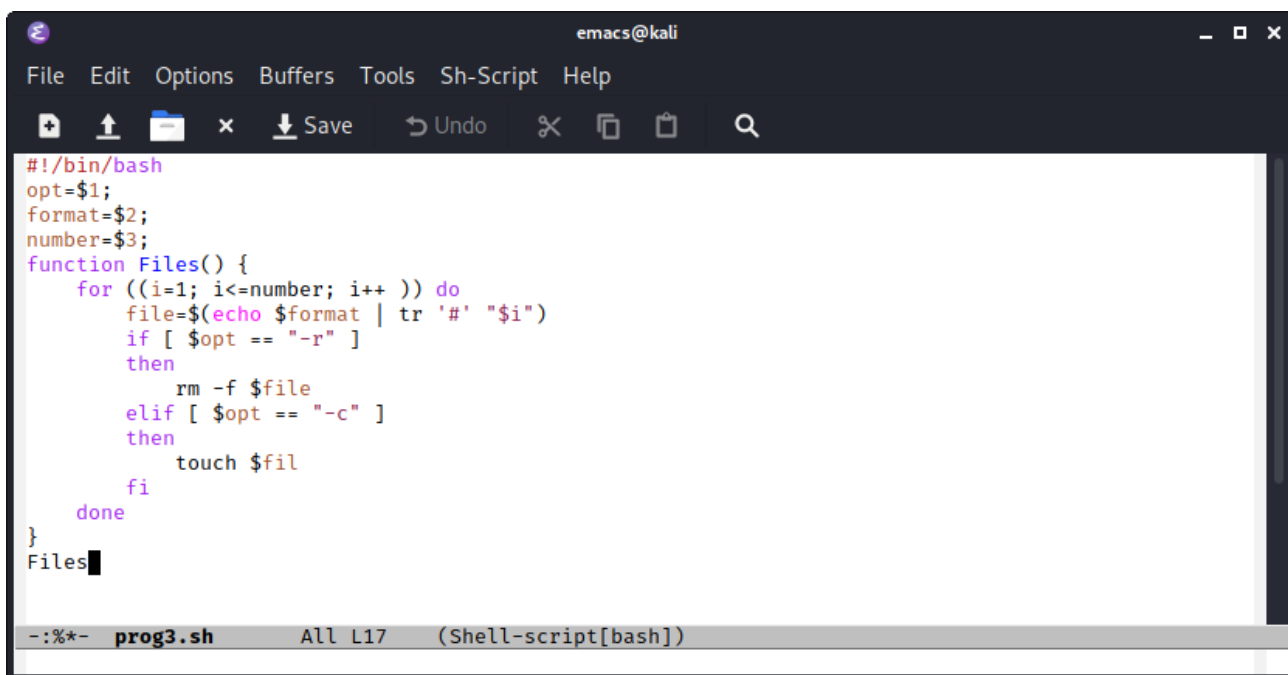
(kekalistratova@kali)-[~]
$ 0
0: command not found

(kekalistratova@kali)-[~]
$ ./prg2.sh ключом -p.
Введите число
0 рипты. (рис. -@fig:001)
Число меньше 0

(kekalistratova@kali)-[~]
$ ./prg2.sh команда «./prog.sh -i a1.txt -o a2.txt -p capital -C -n»), предварительно добавив
Введите число
-3
Число меньше 0
```

Figure 5: Проверка работы скриптов

- 3) Написала 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 создала файл prog3.sh и написала соответствующий скрипт. (рис. 6)

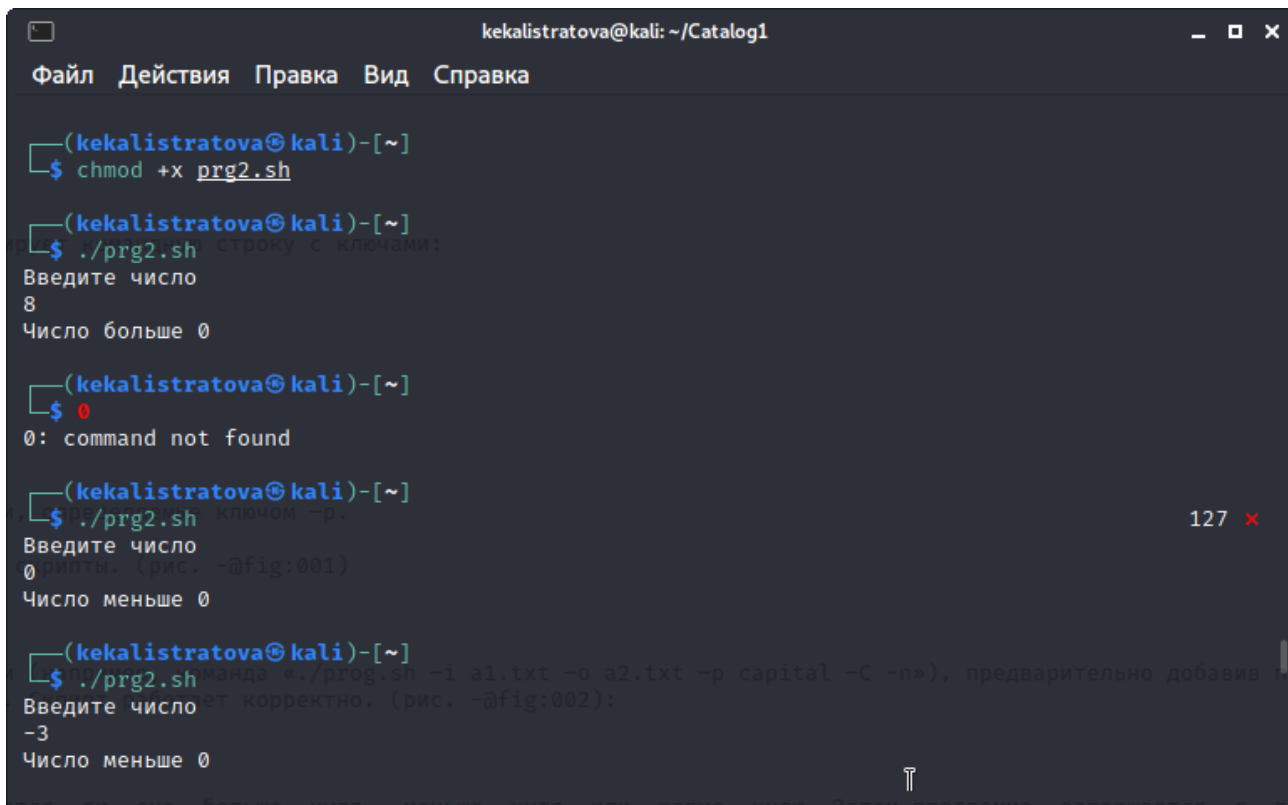


```
#!/bin/bash
opt=$1;
format=$2;
number=$3;
function Files() {
    for ((i=1; i<=number; i++ )) do
        file=$(echo $format | tr '#' "$i")
        if [ $opt == "-r" ]
        then
            rm -f $file
        elif [ $opt == "-c" ]
        then
            touch $fil
        fi
    done
}
Files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

prog3.sh All L17 (Shell-script[bash])

Figure 6: Третий скрипт

Далее я проверила работу написанного скрипта (команда «./prog3.sh»), предварительно добавив право на исполнение файла (команда «chmod +x prog3.sh»). Сначала я создала три файла (команда «./prog3.sh -c b#.txt 3»), удовлетворяющие условию задачи, а потом удалила их(команда «./prog3.sh -r b#.txt 3»). (рис. 7)



```
(kekalistratova@kali)-[~]
$ chmod +x prg2.sh

(kekalistratova@kali)-[~]
$ ./prg2.sh строку с ключами:
Введите число
8
Число больше 0

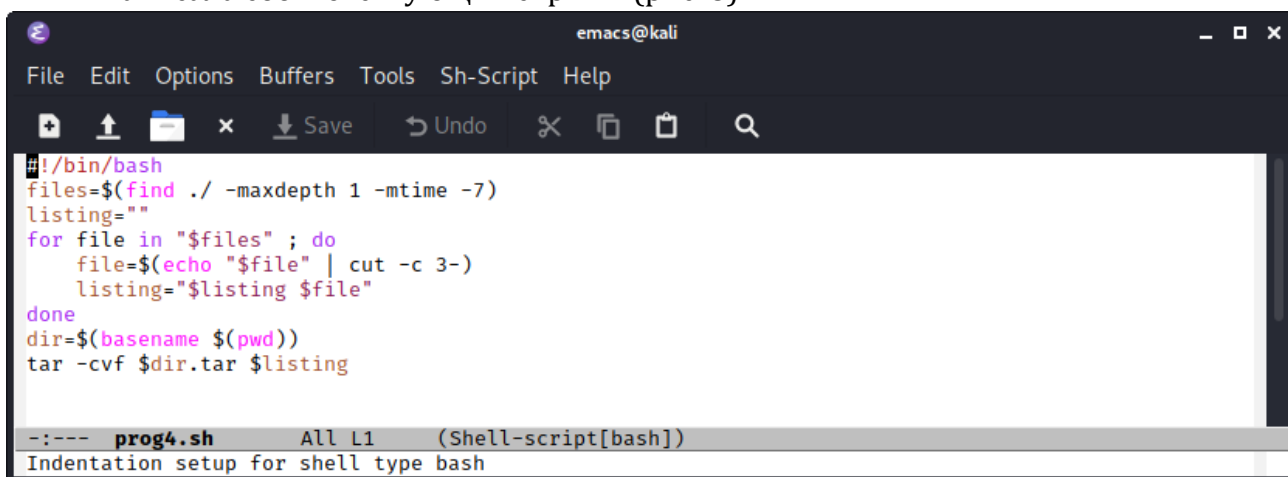
(kekalistratova@kali)-[~]
$ 0
0: command not found

(kekalistratova@kali)-[~]
$ ./prg2.sh ключом -r.
Введите число
0
Число меньше 0

(kekalistratova@kali)-[~]
$ ./prg2.sh
Введите число
-3
Число меньше 0
```

Figure 7: Проверка работы скрипта

- 4) Написала 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 создала файл prog4.sh и написала соответствующий скрипт. (рис. 8)

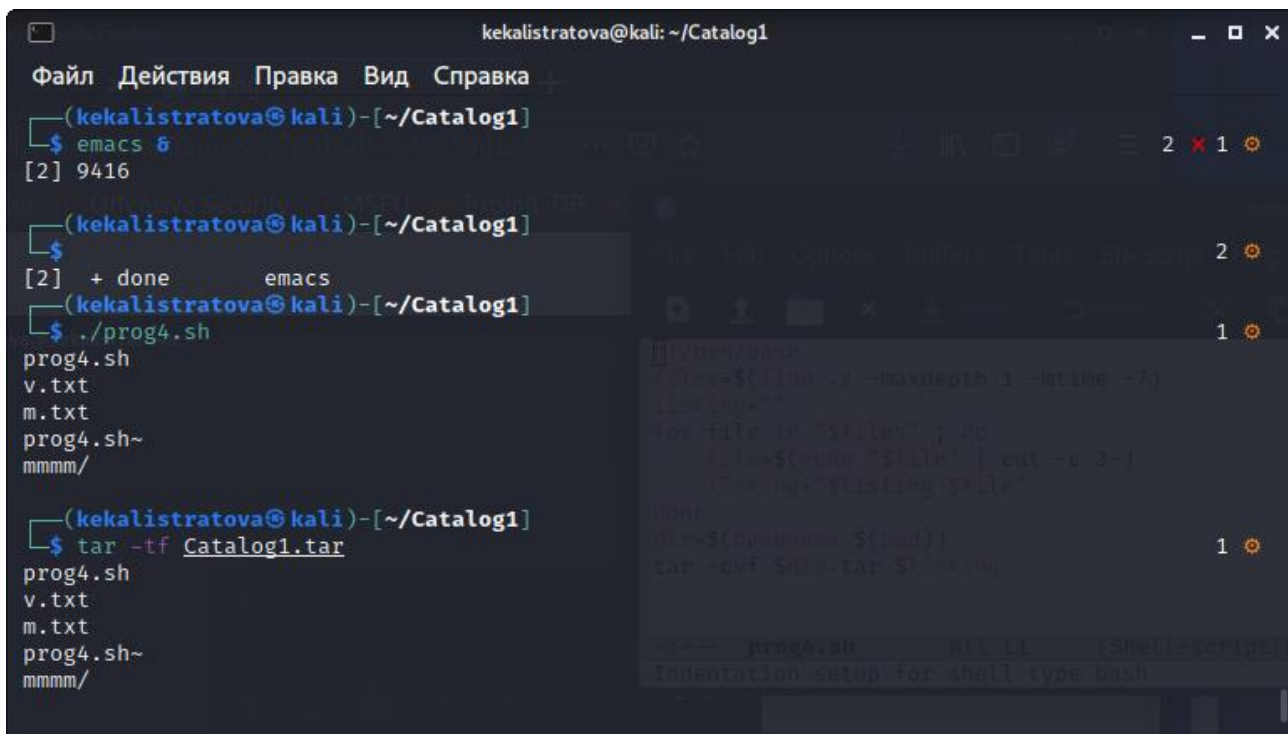


```
#!/bin/bash
files=$(find ./ -maxdepth 1 -mtime -7)
listing=""
for file in "$files" ; do
    file=$(echo "$file" | cut -c 3-)
    listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing
```

-- prog4.sh All L1 (Shell-script[bash])
Indentation setup for shell type bash

Figure 8: Четвертый скрипт

Далее я проверила работу написанного скрипта (команды «./prog4.sh»и «tar -tf catalog.tar»), предварительно добавив право на исполнение файла (команда «chmod +x prog4.sh») и создав отдельный catalog с несколькими файлами. Скрипт работает корректно. (рис. 9)



```
kekalistratova@kali: ~/Catalog1
(kekalistratova@kali)-[~/Catalog1]
$ emacs &
[2] 9416
(kekalistratova@kali)-[~/Catalog1]
$ + done emacs
(kekalistratova@kali)-[~/Catalog1]
$ ./prog4.sh
prog4.sh
v.txt
m.txt
prog4.sh~
mmm/

(kekalistratova@kali)-[~/Catalog1]
$ tar -tf Catalog1.tar
prog4.sh
v.txt
m.txt
prog4.sh~
mmm/
```

Figure 9: Проверка работы скрипта

Контрольные вопросы

- 1) Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, 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 введенные 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 введенных пользователем данных.

- 2) При перечислении имён файлов текущего каталога можно использовать следующие символы:
 1. `*` – соответствует произвольной, в том числе и пустой строке;
 2. `?` – соответствует любому одинарному символу;
 3. `[c1-c2]` – соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`.

Например,

- 1.1. `echo*` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`;
 - 1.2. `ls*.c` – выведет все файлы с последними двумя символами, совпадающими с `.c`.
 - 1.3. `echo prog.?` – выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.`.
 - 1.4. `[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, неравный 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 mans/i.s` проверяет, существует ли файл `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.

Выводы

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

Библиография

1. https://esystem.rudn.ru/pluginfile.php/1142093/mod_resource/content/3/009-lab_shell_prog_2.pdf
2. Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: <http://lib.rudn.ru/MegaPro2/Download/MObject/6118>.
3. Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немнюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ЕТ 60)
4. Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ЕТ 50)