

Лабораторная работа №9

**Программирование в командном процессоре ОС UNIX. Ветвления и
циклы**

Мальсагов Мухаммад Абу-Бакарович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	14
4	Контрольные вопросы	15

Список таблиц

Список иллюстраций

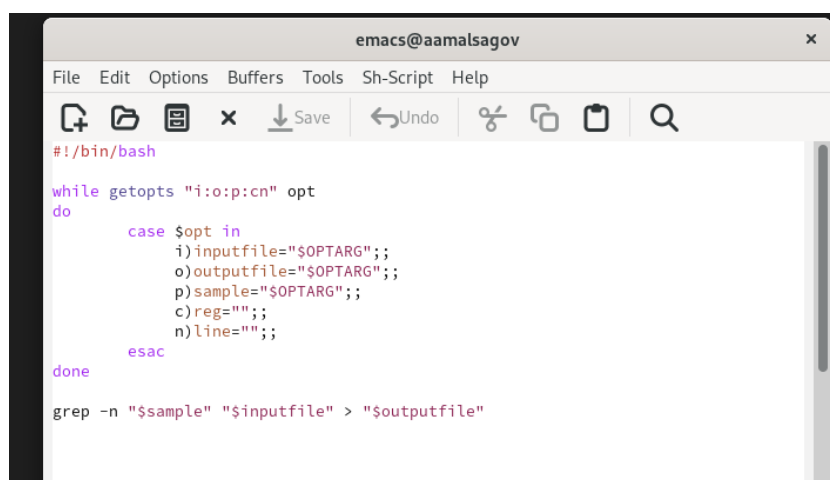
2.1	Код 1 скрипта	6
2.2	Результат работы 1 скрипта	7
2.3	Код 2 скрипта	7
2.4	Код программы на языке C++	8
2.5	Работа скрипта	9
2.6	Код 3 скрипта	10
2.7	Запуск скрипта	11
2.8	Результат работы скрипта	11
2.9	Код 4 скрипта	12
2.10	Результат	13

1 Цель работы

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

2 Выполнение лабораторной работы

1. Создал script1 и открыл его в emacs. Написал программу, которая читает данные из указанного файла, записывает их в другой файл, учитывая введенные опции. (рис. 2.1)

A screenshot of the Emacs text editor window. The title bar reads 'emacs@aamalsagov'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. Below the menu is a toolbar with icons for file operations and editing. The main text area contains a shell script with the following code:

```
#!/bin/bash

while getopts "i:o:p:cn" opt
do
    case $opt in
        i) inputfile="$OPTARG";;
        o) outputfile="$OPTARG";;
        p) sample="$OPTARG";;
        c) reg="";;
        n) line="";;
    esac
done

grep -n "$sample" "$inputfile" > "$outputfile"
```

Рис. 2.1: Код 1 скрипта

2. Проверил его работу. (рис. 2.2)

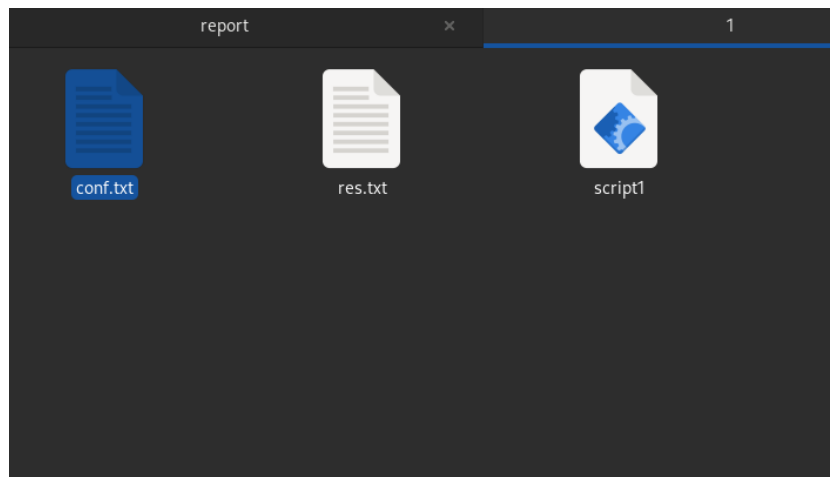


Рис. 2.2: Результат работы 1 скрипта

3. Написал командный файл и программу на языке C++, которые получают на входе число и выводят больше, равно или меньше ли оно 0. (рис. 2.3, 2.4)

```

emacs@aamalsagov
File Edit Options Buffers Tools Sh-Script Help
[Icons] Save Undo [Icons] Search

#!/bin/bash

RES=result
SRC=comparison.cpp

if [ "$SRC" -nt "$RES" ]
then
    echo "Creating $RES ..."
    g++ -o $RES $SRC
fi

./$RES $1

ers=$?

if [ "$ers" == "1" ]
then
    echo "input > 0"
fi

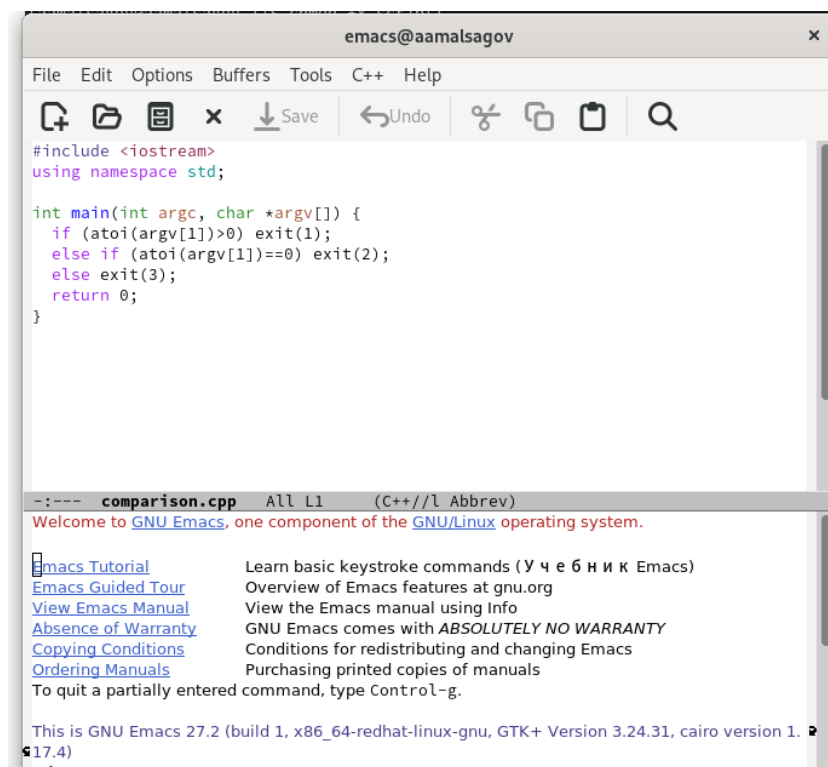
if [ "$ers" == "2" ]
then
    echo "input = 0"
fi

if [ "$ers" == "3" ]
then
    echo "input < 0"
fi

-:--- script2 All L1 (Shell-script[bash])

```

Рис. 2.3: Код 2 скрипта



```
#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
    if (atoi(argv[1])>0) exit(1);
    else if (atoi(argv[1])==0) exit(2);
    else exit(3);
    return 0;
}
```

--:--- **comparison.cpp** All L1 (C++//l Abbrev)

Welcome to [GNU Emacs](#), one component of the [GNU/Linux](#) operating system.

Emacs Tutorial	Learn basic keystroke commands (У ч е б н и к Emacs)
Emacs Guided Tour	Overview of Emacs features at gnu.org
View Emacs Manual	View the Emacs manual using Info
Absence of Warranty	GNU Emacs comes with <i>ABSOLUTELY NO WARRANTY</i>
Copying Conditions	Conditions for redistributing and changing Emacs
Ordering Manuals	Purchasing printed copies of manuals

To quit a partially entered command, type Control-g.

This is GNU Emacs 27.2 (build 1, x86_64-redhat-linux-gnu, GTK+ Version 3.24.31, cairo version 1.17.4)

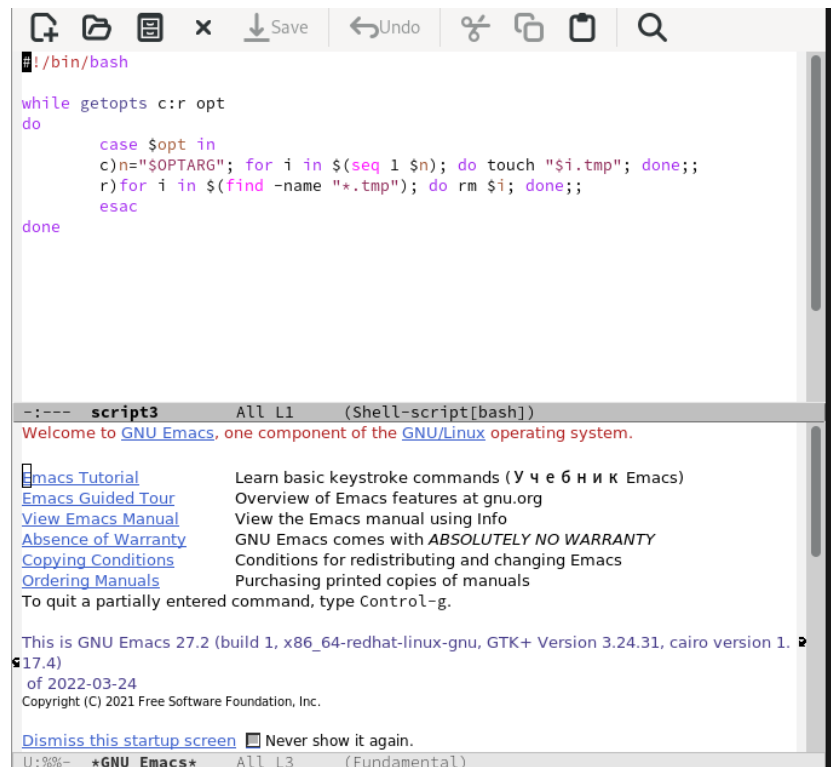
Рис. 2.4: Код программы на языке C++

4. Запустил скрипт.(рис. 2.5)


```
[aamalsagov@aamalsagov 1]$ chmod +x script1
[aamalsagov@aamalsagov 1]$ ./script1 -i conf.txt -o res.txt -p n etconf -c -n
[aamalsagov@aamalsagov 1]$ cd ..
[aamalsagov@aamalsagov scripts]$ cd 2
[aamalsagov@aamalsagov 2]$ emacs script2
[aamalsagov@aamalsagov 2]$ emacs comparison.cpp
[aamalsagov@aamalsagov 2]$ chmod +x script2
[aamalsagov@aamalsagov 2]$ ./script2 3
Creating result ...
input > 0
[aamalsagov@aamalsagov 2]$ ./script2 0
input = 0
[aamalsagov@aamalsagov 2]$ ./script2 -1
input < 0
[aamalsagov@aamalsagov 2]$
```

Рис. 2.5: Работа скрипта

5. Создал script3 и открыл его в emacs. Написал программу, которая в зависимости от введенных опций либо создает определенное кол-во файлов, либо удаляет их всех.(рис. 2.6)



The image shows a screenshot of the Emacs editor interface. The top part of the window displays a shell script in a buffer named 'script3'. The script is a while loop that iterates over command-line options and performs actions based on them. The bottom part of the window shows the Emacs startup screen, which includes a welcome message, a list of links for tutorials and manuals, and version information.

```
#!/bin/bash

while getopts c:r opt
do
    case $opt in
        c)n="$OPTARG"; for i in $(seq 1 $n); do touch "$i.tmp"; done;;
        r)for i in $(find -name "*.tmp"); do rm $i; done;;
        esac
    done
```

--- script3 All L1 (Shell-script[bash])

Welcome to GNU Emacs, one component of the GNU/Linux operating system.

[Emacs Tutorial](#) Learn basic keystroke commands (У ч е б н и к Emacs)
[Emacs Guided Tour](#) Overview of Emacs features at gnu.org
[View Emacs Manual](#) View the Emacs manual using Info
[Absence of Warranty](#) GNU Emacs comes with *ABSOLUTELY NO WARRANTY*
[Copying Conditions](#) Conditions for redistributing and changing Emacs
[Ordering Manuals](#) Purchasing printed copies of manuals

To quit a partially entered command, type Control-g.

This is GNU Emacs 27.2 (build 1, x86_64-redhat-linux-gnu, GTK+ Version 3.24.31, cairo version 1.17.4)
of 2022-03-24
Copyright (C) 2021 Free Software Foundation, Inc.

[Dismiss this startup screen](#) ☐ Never show it again.

U:%%- *GNU Emacs* All L3 (Fundamental)

Рис. 2.6: Код 3 скрипта

6. Проверил работу скрипта. (рис. 2.7, 2.8)

```
[aamalsagov@aamalsagov scripts]$ cd 1
[aamalsagov@aamalsagov 1]$ emacs script1
[aamalsagov@aamalsagov 1]$ chmod +x script1
[aamalsagov@aamalsagov 1]$ ./script1 -i conf.txt -o res.txt -p n etconf -c -n
[aamalsagov@aamalsagov 1]$ cd ..
[aamalsagov@aamalsagov scripts]$ cd 2
[aamalsagov@aamalsagov 2]$ emacs script2
[aamalsagov@aamalsagov 2]$ emacs comparison.cpp
[aamalsagov@aamalsagov 2]$ chmod +x script2
[aamalsagov@aamalsagov 2]$ ./script2 3
Creating result ...
input > 0
[aamalsagov@aamalsagov 2]$ ./script2 0
input = 0
[aamalsagov@aamalsagov 2]$ ./script2 -1
input < 0
[aamalsagov@aamalsagov 2]$ cd ..
[aamalsagov@aamalsagov scripts]$ cd 3
[aamalsagov@aamalsagov 3]$ emacs script3
[aamalsagov@aamalsagov 3]$ chmod +x script3
[aamalsagov@aamalsagov 3]$ ./script3 -c 7
[aamalsagov@aamalsagov 3]$ ./script3 -r
[aamalsagov@aamalsagov 3]$
```

Рис. 2.7: Запуск скрипта

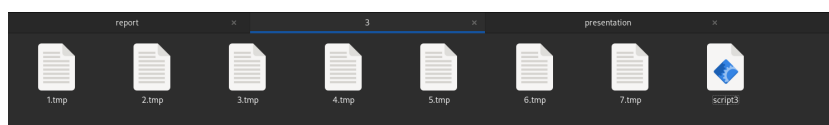


Рис. 2.8: Результат работы скрипта

7. Создал script4 и открыл его в emacs. Написал программу, которая с помощью команды tar заковывает в архив все файлы в указанной директории. Модифицировал его так, чтобы заковывались только те файлы, которые были изменены менее недели тому назад.(рис. 2.9)

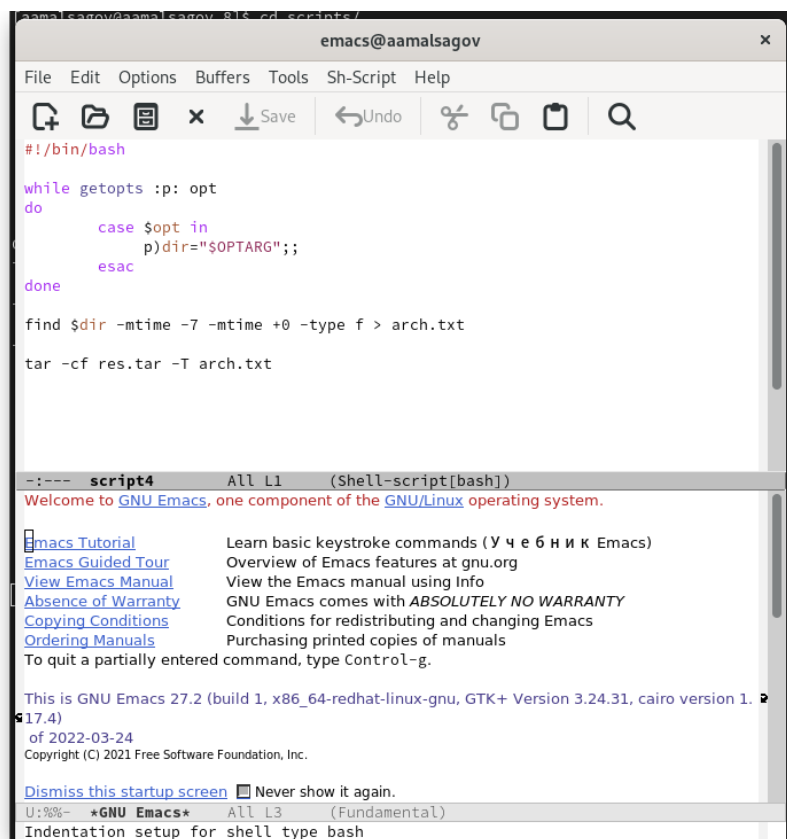


Рис. 2.9: Код 4 скрипта

8. Запустил скрипт.(рис. 2.10)

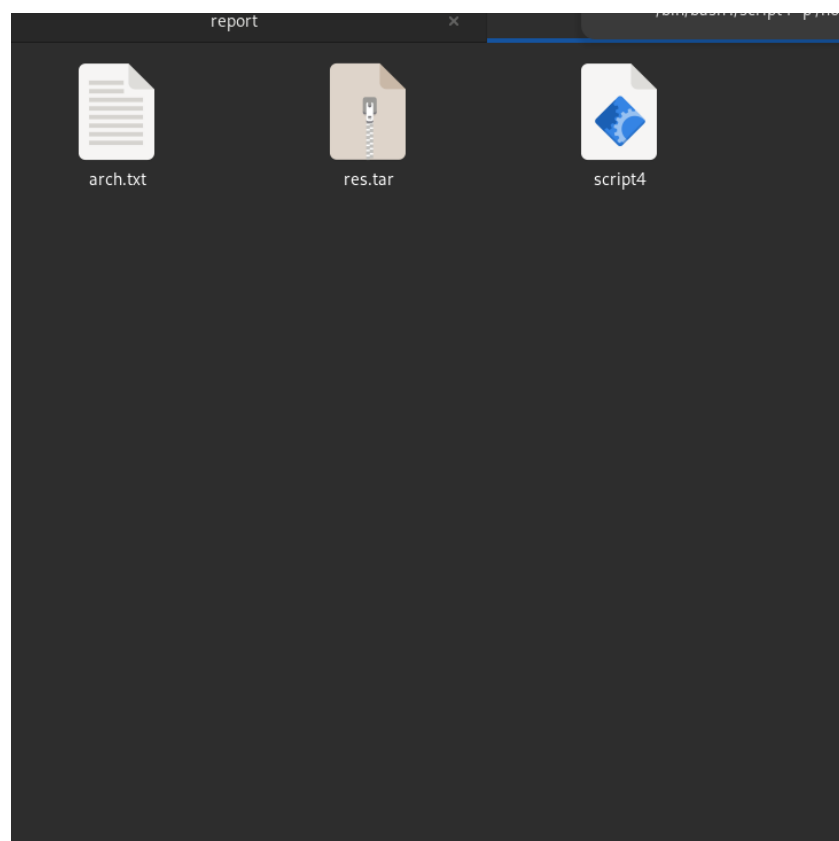


Рис. 2.10: Результат

3 Выводы

Мы научились писать более сложные команды файлы.

4 Контрольные вопросы

1. Весьма необходимой при программировании является команда `getopts`, которая 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, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`. Предположим, необходимо распознать командную строку следующего формата: `testprog -infile_in.txt -outfile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае:

```
while getopts o:i:Ltr optletter do
  case $optletter in
    o) iflag=1; ival=$OPTARG;;
    i) iflag=1; ival=$OPTARG;;
    L) Lflag=1;;
    t) tflag=1;;
    r) rflag=1;;
    *) echo Illegal option $optletter; esac
done
```

 Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равна `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является

числовым индексом на упомянутый аргумент. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

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

- `-` — соответствует произвольной, в том числе и пустой строке;
- `?` — соответствует любому одному символу;
- `[c1-c1]` — соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`.
- `echo *` — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`;
- `ls *.c` — выведет все файлы с последними двумя символами, равными `.c`.
- `echo prog.?` — выдаст 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 Вам 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 дела являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда.

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 перестает быть правильным. Пример бесконечного цикла `while`, с прерыванием в момент, когда файл перестает существовать: `while true do if [! -f $file] then break fi sleep 10 done`
5. Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения.
6. Введенная строка означает условие существования файла `mans/i.$s`
7. Если речь идет о 2-х параллельных действиях, то это `while`. когда мы показываем, что сначала делается 1-е действие. потом оно заканчивается при наступлении 2-го действия, применяем `until`.