

Лабораторная работа №11

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

Тулеуов Мадн

Содержание

1	Цель работы	5
2	Задачи	6
3	Ход работы	8
4	Вывод	14
5	Контрольные вопросы.	15

Список таблиц

Список иллюстраций

3.1	Код 1 скрипта	8
3.2	Работа скрипта	8
3.3	Код 2 скрипта	9
3.4	Код C++ файла	9
3.5	Работа скрипта	10
3.6	Код 3 скрипта	10
3.7	Создание файлов	11
3.8	Результат создания	11
3.9	Удаление файлов	11
3.10	Результат удаления	12
3.11	Код 4 скрипта	12
3.12	Проверка	13

1 Цель работы

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

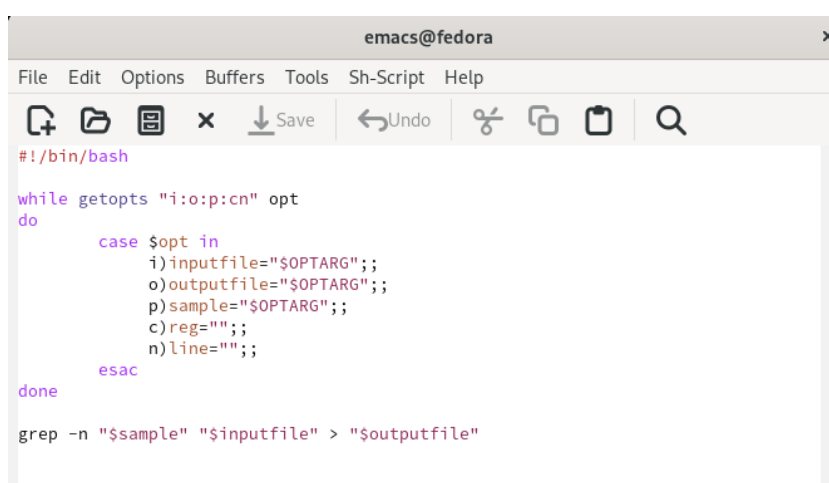
2 Задачи

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 ∞ (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 Ход работы

1. Написал скрипт, который читает данные из указанного файла, записывает их в другой, учитывая введенные опции. (рис. 3.1)

The image shows a screenshot of an Emacs editor window titled 'emacs@fedora'. The window has a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. Below the menu bar is a toolbar with icons for file operations and editing. The main text area contains a shell script with the following code:

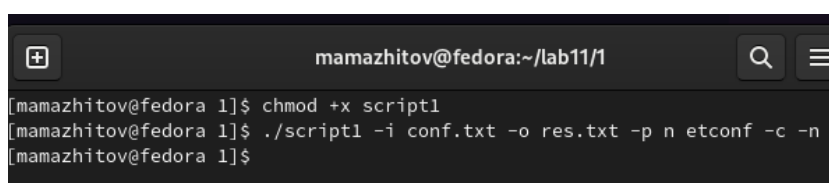
```
#!/bin/bash

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. 3.1: Код 1 скрипта

Запустил скрипт. (рис. 3.2)

The image shows a screenshot of a terminal window titled 'mamazhitov@fedora:~/lab11/1'. The terminal displays the following commands and their output:

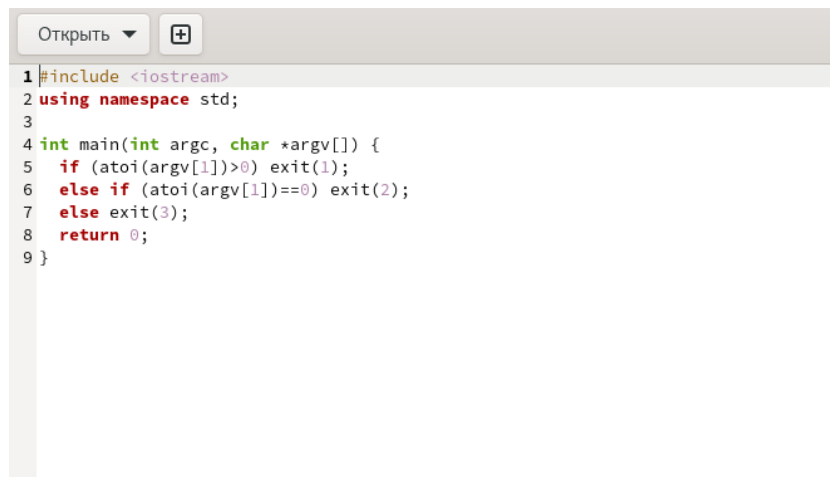
```
[mamazhitov@fedora 1]$ chmod +x script1
[mamazhitov@fedora 1]$ ./script1 -i conf.txt -o res.txt -p n etconf -c -n
[mamazhitov@fedora 1]$
```

Рис. 3.2: Работа скрипта

Проверка. (рис. ??)

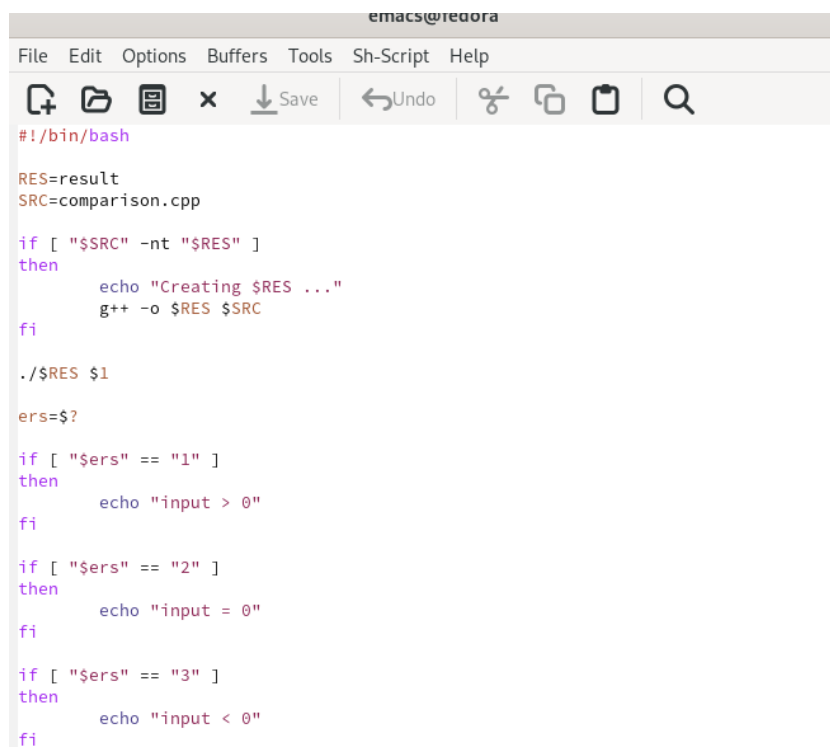
Проверка работы скрипта

2. Написал командный файл и программу на языке C++, которые получают на входе число и выводит больше, меньше или равно "0".(рис. 3.3, 3.4)



```
1#include <iostream>
2using namespace std;
3
4int main(int argc, char *argv[]) {
5    if (atoi(argv[1])>0) exit(1);
6    else if (atoi(argv[1])==0) exit(2);
7    else exit(3);
8    return 0;
9}
```

Рис. 3.3: Код 2 скрипта



```
emacs@redora
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
```

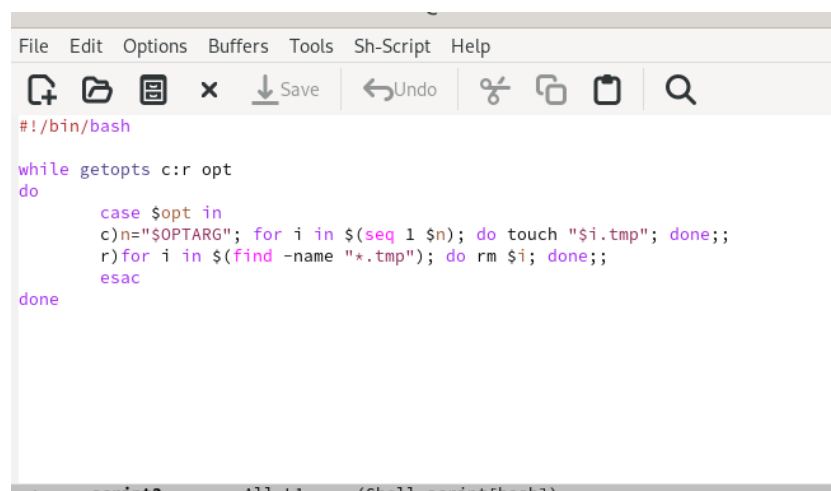
Рис. 3.4: Код C++ файла

Запустил скрипт.(рис. 3.5)

```
[mamazhitov@fedora 2]$ chmod +x script2
[mamazhitov@fedora 2]$ ./script2 3
Creating result ...
input > 0
[mamazhitov@fedora 2]$ ./script2 0
input = 0
[mamazhitov@fedora 2]$ ./script2 -1
input < 0
[mamazhitov@fedora 2]$
```

Рис. 3.5: Работа скрипта

3. Открыл в emacs файл *script3* и написал программу, которая в зависимости от введенных опций либо создает определенное кол-во файлов, либо удаляет их.(рис. 3.6)

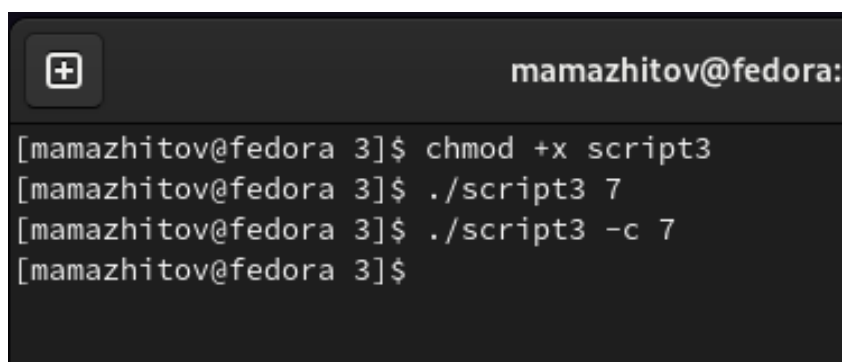


```
#!/bin/bash

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
```

Рис. 3.6: Код 3 скрипта

Запустил скрипт для создания файлов.(рис. 3.7)



```
mamazhitov@fedora:~$  
[mamazhitov@fedora 3]$ chmod +x script3  
[mamazhitov@fedora 3]$ ./script3 7  
[mamazhitov@fedora 3]$ ./script3 -c 7  
[mamazhitov@fedora 3]$
```

Рис. 3.7: Создание файлов

Результат.(рис. 3.8)

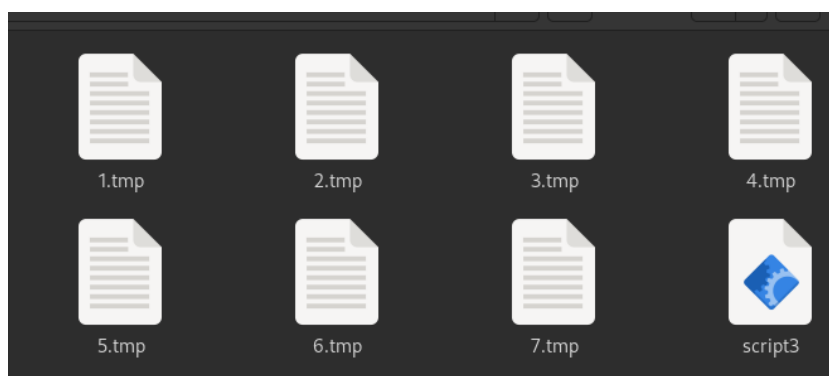
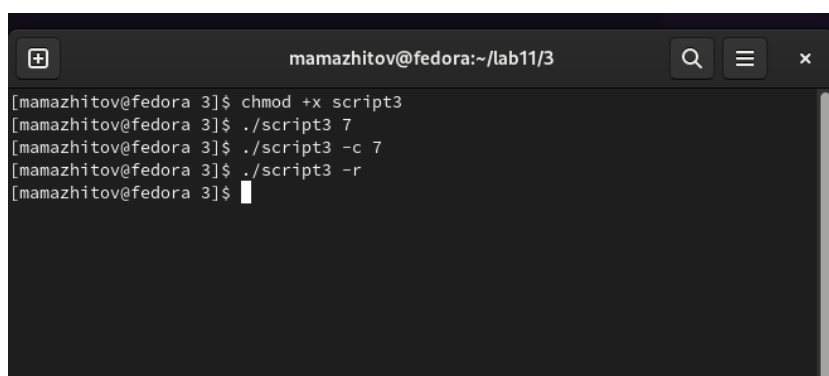


Рис. 3.8: Результат создания

Запустил скрипт для удаления файлов.(рис. 3.9)



```
mamazhitov@fedora:~/lab11/3$  
[mamazhitov@fedora 3]$ chmod +x script3  
[mamazhitov@fedora 3]$ ./script3 7  
[mamazhitov@fedora 3]$ ./script3 -c 7  
[mamazhitov@fedora 3]$ ./script3 -r  
[mamazhitov@fedora 3]$
```

Рис. 3.9: Удаление файлов

Результат.(рис. 3.10)

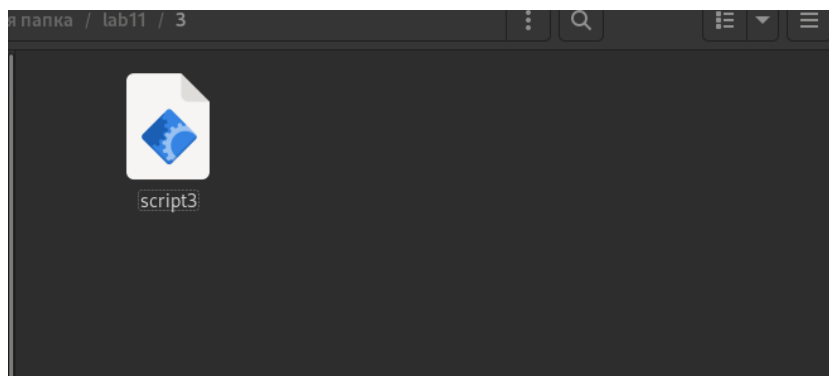


Рис. 3.10: Резльтат удаления

4. Открыл в emacs файл *script4*. Написал программу, которая с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировал его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (рис. 3.11)

```
mamazhitov@fedora:~/lab11/4
[mamazhitov@fedora 4]$ chmod +x script4
[mamazhitov@fedora 4]$ ./script4 /home/mamazhitov
[mamazhitov@fedora 4]$ ./script4 -p /home/mamazhitov
find: '/home/mamazhitov/.local/share/Trash/expunged/1244508069/games': Отказано
в доступе
tar: Удаляется начальный '/' из имен объектов
tar: Удаляются начальные '/' из целей жестких ссылок
[mamazhitov@fedora 4]$
```

Рис. 3.11: Код 4 скрипта

Проверил работу скрипта. (рис. 3.12)

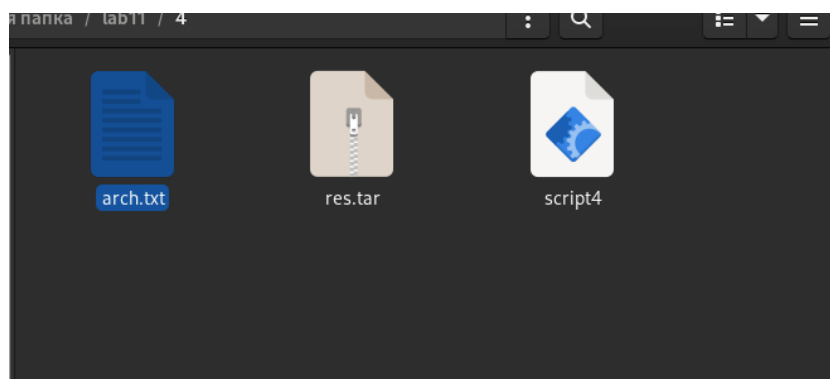


Рис. 3.12: Проверка

4 Вывод

Мы научились писать более сложные командные файлы.

5 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`.