

# **Отчёт по лабораторной работе №11**

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

Хусаинова Динара Айратовна

# Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Ход работы	8
4	Контрольные вопросы	15
5	Вывод	18

## Список иллюстраций

3.1	Содержимое файла . . . . .	8
3.2	Содержимое файла . . . . .	9
3.3	Предоставляем права доступа . . . . .	9
3.4	Проверка . . . . .	9
3.5	Файл на языке Си . . . . .	10
3.6	Командный файл . . . . .	11
3.7	Проверка . . . . .	11
3.8	Командный файл . . . . .	12
3.9	Проверка . . . . .	12
3.10	Проверка . . . . .	13
3.11	Содержимое файла . . . . .	13
3.12	Проверка . . . . .	14

## List of Tables

# 1 Цель работы

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

## 2 Теоретическое введение

Циклы позволяют выполнять один и тот же участок кода необходимое количество раз. В большинстве языков программирования существует несколько типов циклов. Большинство из них поддерживаются оболочкой Bash. Мы рассмотрим их все в сегодняшней статье, но сначала поговорим какими они бывают:

`for` - позволяет перебрать все элементы из массива или использует переменную-счетчик для определения количества повторений;

`while` - цикл выполняется пока условие истинно;

`until` - цикл выполняется пока условие ложно.

Циклы Bash, это очень полезная вещь и разобраться с ними будет несложно. Bash позволяет использовать циклы как в скриптах, так и непосредственно в командной оболочке. Далее мы рассмотрим каждый из этих видов циклов.

### ЦИКЛ FOR

Цикл `for bash` применяется очень часто из-за своей специфики. Его проще всего использовать когда вы знаете сколько раз нужно повторить операцию или вам нужно просто обработать по очереди все элементы массива и вы не хотите контролировать количество повторений.

Цикл `for` имеет несколько синтаксисов и может вести себя по разному. Для перебора элементов списка удобно использовать такой синтаксис:

```
for переменная in список
do
команда1
команда2
```

done

Каждый цикл `for` независимо от типа начинается с ключевого слова `for`. Дальше все зависит от типа. В этом случае после `for` указывается имя переменной, в которую будет сохранен каждый элемент списка, затем идет ключевое слово `in` и сам список. Команды, которые нужно выполнять в цикле размещаются между словами `do` и `done`.

## 3 Ход работы

1. Используя команды `getopts` `grep`, напишем командный файл, который анализирует командную строку с ключами: – `-iinputfile` — прочитать данные из указанного файла;

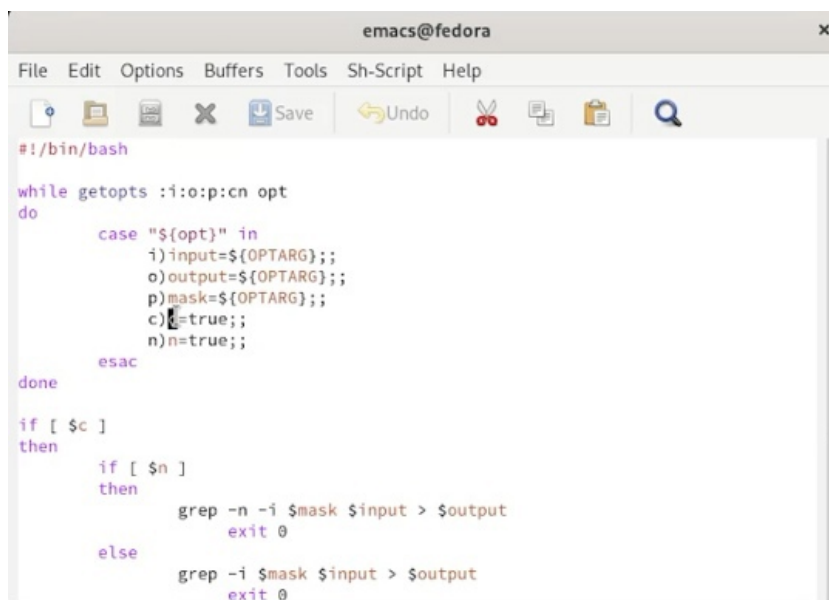
– `-ooutputfile` — вывести данные в указанный файл;

– `-rшаблон` — указать шаблон для поиска;

– `-C` — различать большие и малые буквы;

– `-n` — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом `-r`(рис. 3.1,3.2).



```
#!/bin/bash

while getopts :i:o:p:cn opt
do
    case "${opt}" in
        i) input=${OPTARG};;
        o) output=${OPTARG};;
        p) mask=${OPTARG};;
        c) C=true;;
        n) n=true;;
    esac
done

if [ $C ]
then
    if [ $n ]
    then
        grep -n -i $mask $input > $output
        exit 0
    else
        grep -i $mask $input > $output
        exit 0
    fi
fi
```

Рис. 3.1: Содержимое файла



```

done
esac
if [ $c ]
then
    if [ $n ]
    then
        grep -n -i $mask $input > $output
        exit 0
    else
        grep -i $mask $input > $output
        exit 0
    fi
else
    grep -i $mask $input > $output
    exit
fi

```

Рис. 3.2: Содержимое файла

Теперь проверим его работу, сначала даем права на выполнение, а затем запускаем и осуществляем поиск комбинации букв AU(рис. 3.3,3.4).

```

dakhusainova@fedora:~
[dakhusainova@fedora ~]$ chmod +x task111

```

Рис. 3.3: Предоставляем права доступа

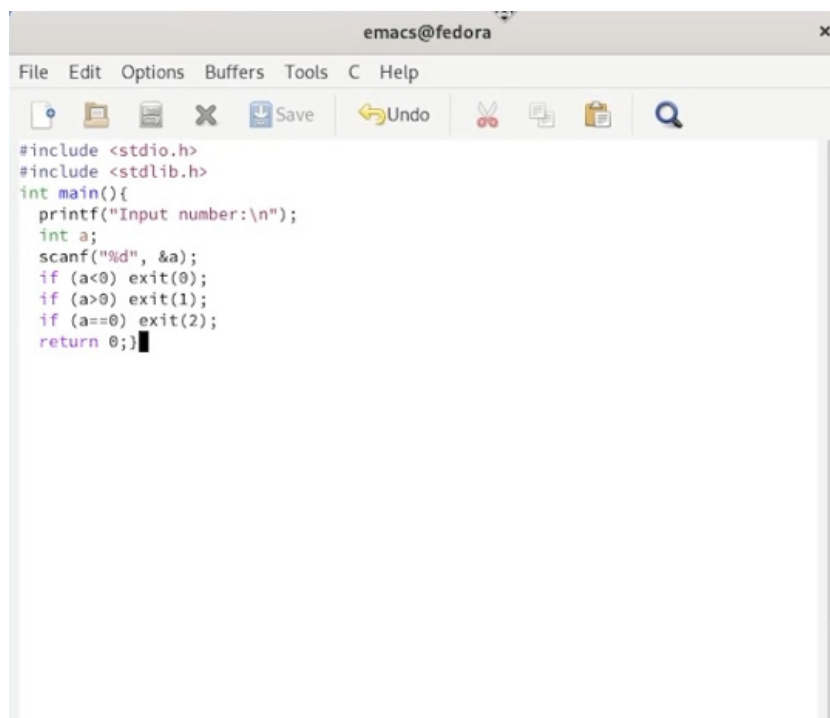
```

dakhusainova@fedora:~
[dakhusainova@fedora ~]$ ./task111 -i file.txt -o rudo.txt -p "AU" -n -c
[dakhusainova@fedora ~]$ grep -n -i "AU" file.txt
10:audit
11:authselect
37:default
113:libaudit.conf
272:australia
[dakhusainova@fedora ~]$ ./task111 -i file.txt -o rudo.txt -p "AU" -n
[dakhusainova@fedora ~]$ grep -n -i "AU" file.txt
10:audit
11:authselect
37:default
113:libaudit.conf
272:australia
[dakhusainova@fedora ~]$ grep -n "AU" file.txt
[dakhusainova@fedora ~]$ ./task111 -i file.txt -o rudo.txt -p "AU" -n -c
[dakhusainova@fedora ~]$ cat rudo.txt
10:audit
11:authselect
37:default
113:libaudit.conf
272:australia
[dakhusainova@fedora ~]$

```

Рис. 3.4: Проверка

2. Напишем на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершится с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку( рис. 3.5,3.6,3.7).



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

Рис. 3.5: Файл на языке Си

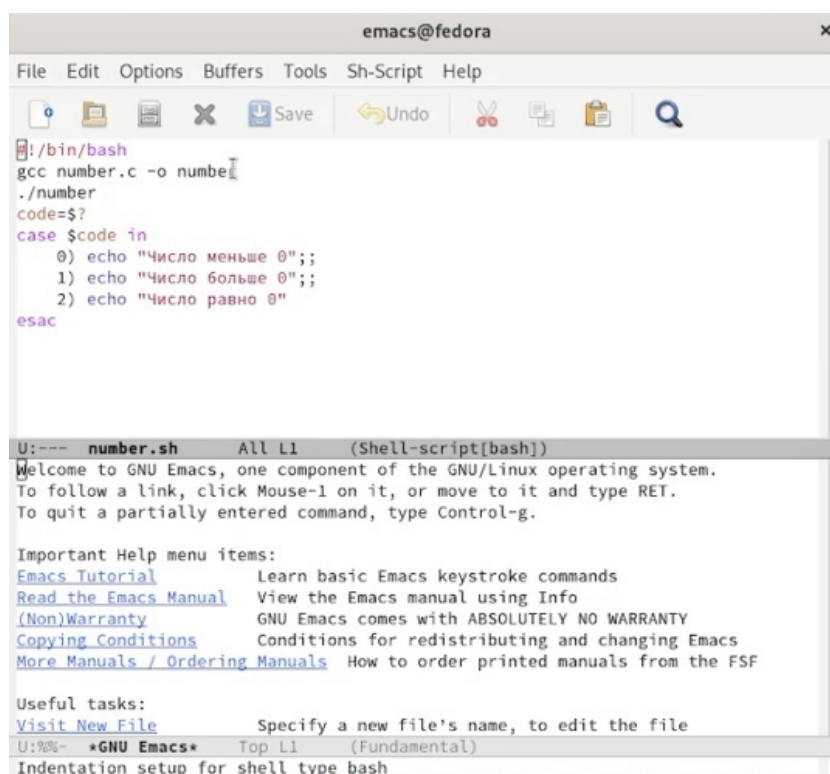


Рис. 3.6: Командный файл

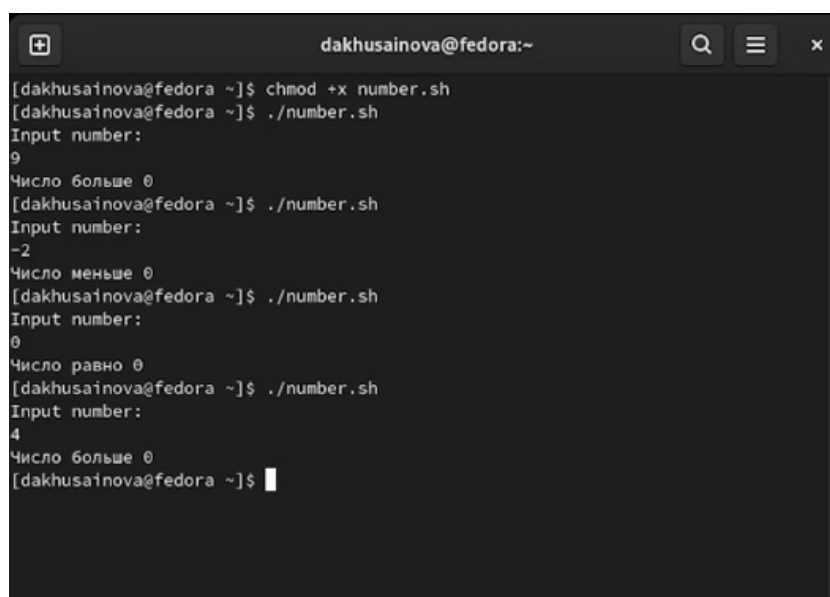
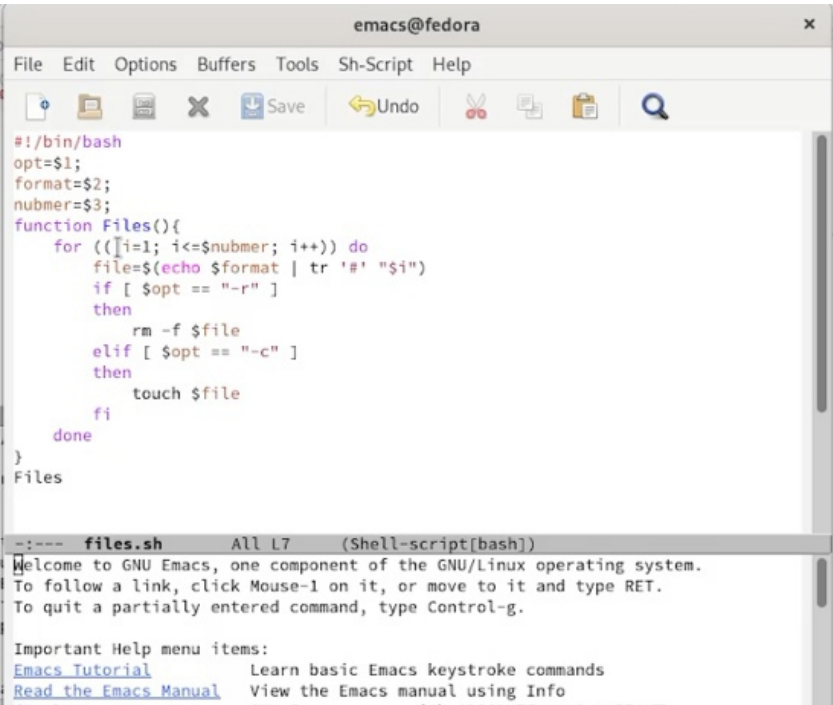


Рис. 3.7: Проверка

3. Напишем командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до  $\infty$  (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)(рис. 3.8,3.9,3.10).

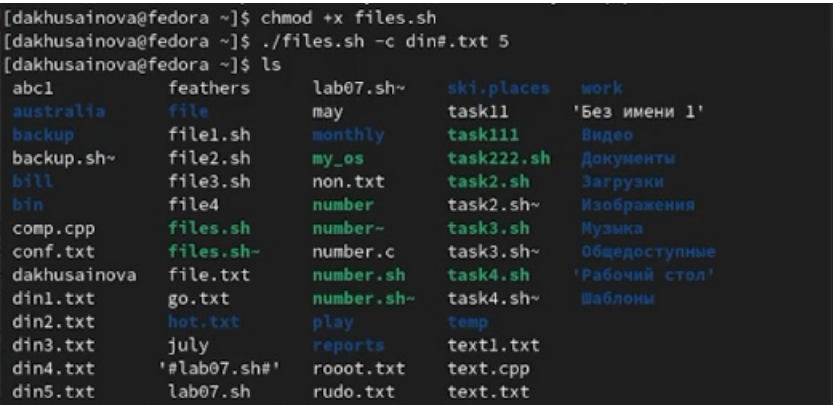


The screenshot shows the Emacs editor window titled 'emacs@fedora'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The toolbar contains icons for opening, saving, undo, redo, and search. The main text area contains a shell script:

```
#!/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 $file
    fi
  done
}
Files
```

Below the script, the status bar shows '--- files.sh All L7 (Shell-script[bash])'. The bottom panel displays the Emacs welcome message and help menu items.

Рис. 3.8: Командный файл



The screenshot shows a terminal window with the following commands and output:

```
[dakhusainova@fedora ~]$ chmod +x files.sh
[dakhusainova@fedora ~]$ ./files.sh -c din#.txt 5
[dakhusainova@fedora ~]$ ls
```

abc1	feathers	lab07.sh~	ski.places	work
australia	file	may	task11	'Без имени 1'
backup	file1.sh	monthly	task111	Видео
backup.sh~	file2.sh	my_os	task222.sh	Документы
bill	file3.sh	non.txt	task2.sh	Загрузки
bin	file4	number	task2.sh~	Изображения
comp.cpp	files.sh	number~	task3.sh	Музыка
conf.txt	files.sh~	number.c	task3.sh~	Общедоступные
dakhusainova	file.txt	number.sh	task4.sh	'Рабочий стол'
din1.txt	go.txt	number.sh~	task4.sh~	Шаблоны
din2.txt	hot.txt	play	temp	
din3.txt	july	reports	text1.txt	
din4.txt	'#lab07.sh#'	rooot.txt	text.cpp	
din5.txt	lab07.sh	rudo.txt	text.txt	

Рис. 3.9: Проверка

```
[dakhusainova@fedora ~]$ ./files.sh -r din#.txt 5
[dakhusainova@fedora ~]$ ls
abc1      file3.sh    my_os      task111     'Без имени 1'
australia file4        non.txt    task222.sh  Видео
backup    files.sh    number     task2.sh    Документы
backup.sh~ files.sh~    number~    task2.sh~   Загрузки
bill      file.txt    number.c   task3.sh    Изображения
bin       go.txt      number.sh  task3.sh~   Музыка
comp.cpp  hot.txt     number.sh~ task4.sh     Общедоступные
conf.txt  july        play       task4.sh~   'Рабочий стол'
dakhusainova '#lab07.sh#' reports     temp        Шаблоны
feathers  lab07.sh    rooot.txt  text1.txt
file      lab07.sh~   rudo.txt   text.cpp
file1.sh  may         ski.places text.txt
file2.sh  monthly    task11     work
```

Рис. 3.10: Проверка

4. Напишем 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)(рис. 3.11,3.12).

```
#!/bin/bash

while getopts :d: opt; do
    case $opt in
        d)dir="$OPTARG";;
    esac
done
find $dir -mtime -7 -mtime +0 -type f > fortar.txt
tar -cvf archive.tar -T fortar.txt
```

Рис. 3.11: Содержимое файла

```
[dakhusainova@fedora ~]$ ls
abc1      file4      my_os      task111     text.txt
archive.tar files.sh    non.txt    task222.sh  tmp
australia files.sh~   number     task2.sh    'Без имени 1'
backup    file.txt   number~    task2.sh~   Видео
backup.sh~ fortar.txt number.c    task3.sh    Документы
bill      go.txt     number.sh   task3.sh~   Загрузки
bin       hot.txt    number.sh~  task444.sh  Изображения
Catalog1  july       play        task444.sh~ Музыка
dakhusainova '#lab07.sh#' reports     task4.sh    Общедоступные
file      lab07.sh   rooot.txt   task4.sh~   'Рабочий стол'
file1.sh  lab07.sh~ rudo.txt    temp        Шаблоны
file2.sh  may        ski.places  text1.txt
file3.sh  monthly    task11      text.cpp
[dakhusainova@fedora ~]$ mkdir tmp
```

Рис. 3.12: Проверка

## 4 Контрольные вопросы

### 1. Каково предназначение команды getopts?

Getopts-это встроенная команда оболочки Unix для анализа аргументов командной строки. Он предназначен для обработки аргументов командной строки, которые следуют рекомендациям синтаксиса утилиты POSIX, основанным на интерфейсе C getopt.

### 2. Какое отношение метасимволы имеют к генерации имён файлов?

При генерации имен используют метасимволы: звездочка - произвольная (возможно пустая) последовательность символов; "?" - один произвольный символ; [...] любой из символов, указанных в скобках перечислением и/или с указанием диапазона; cat f\* выдаст все файлы каталога, начинающиеся с "f"; cat f выдаст все файлы, содержащие "f"; cat program.? выдаст файлы данного каталога с однобуквенными расширениями, скажем "program.c" и "program.o", но не выдаст "program.com"; cat [a-d]\* выдаст файлы, которые начинаются с "a", "b", "c", "d". Аналогичный эффект дадут и команды "cat [abcd]" и "cat [bdac]".

### 3. Какие операторы управления действиями вы знаете?

Точка с запятой (;)

Амперсанд (&)

Символ доллара со знаком вопроса (\$?)

Двойной амперсанд (&&)

Двойная вертикальная черта (||)

Комбинирование операторов && и ||

Знак фунта (#)

#### 4. Какие операторы используются для прерывания цикла?

Утверждения прерывания и продолжения. Операторы `break` и `continue` могут использоваться для управления выполнением цикла `for`. Заявление о перерыве. Чтобы использовать оператор `break`, пользователи должны указать конкретное условие, при выполнении которого цикл будет прерван.

#### 5. Для чего нужны команды `false` и `true`?

В Unix-подобных операционных системах, `true` и `false` являются командами, единственной функцией которых является всегда возвращаться с заданным статусом выхода. Программисты и сценарии часто используют статус выхода команды для оценки успеха (нулевой статус выхода) или отказа (ненулевое значение) команды. `true` и `false` команды, представляют собой логические значения из командного успеха, потому что истинные возвращает 0, и ложное возвращение 1.

#### 6. Что означает строка `if test -f mans/i.$s`, встреченная в командном файле?

Конструкция условного оператора в слегка упрощенном виде выглядит так: `if list1 then list2 else list3 fi` где `list1`, `list2` и `list3` — это последовательности команд, разделенные запятыми и оканчивающиеся точкой с запятой или символом новой строки. Кроме того, эти последовательности могут быть заключены в фигурные скобки: `{list}`. Оператор `if` проверяет значение, возвращаемое командами из `list1`. Если в этом списке несколько команд, то проверяется значение, возвращаемое последней командой списка. Если это значение равно 0, то будут выполняться команды из `list2`; если это значение не нулевое, будут выполнены команды из `list3`. Значение, возвращаемой таким составным оператором `if`, совпадает со значением, выдаваемым последней командой выполняемой последовательности. Полный формат команды `if` имеет вид: `if list then list [ elif list then list ] ... [ else list ] fi` (здесь квадратные скобки означают только необязательность присутствия в операторе того, что в них содержится). В качестве выражения, которое стоит сразу после `if` или `elif`, часто используется команда `test`, которая может обозначаться также квадратными скобками `[ ]`. Команда `test` выполняет вычисление некоторого выражения и возвращает значение 0, если выражение истинно, и 1 в



противном случае. Выражение передается программе `test` как аргумент. Вместо того, чтобы писать `test expression`, можно заключить выражение в квадратные скобки: `[ expression ]`. Заметьте, что `test` и `[` — это два имени одной и той же программы, а не какое-то магическое преобразование, выполняемое оболочкой `bash` (только синтаксис `[` требует, чтобы была поставлена закрывающая скобка). Заметьте также, что вместо `test` в конструкции `if` может быть использована любая программа.

#### 7. Объясните различия между конструкциями `while` и `until`.

Оператор `while` выполняет тело цикла, пока какое-то условие истинно, т.е. выражение или команда возвращают нулевой код. Оператор `until` наоборот, выполняет тело цикла, пока условие ложно, т.е. код возврата выражения или команды отличен от нуля. Простой пример цикла `while`: `count=0 while [ $count -lt 5 ]; do (count++) echo $count done` 1 2 3 4 5.

## 5 Вывод

Мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.