

# **Отчёт по лабораторной работе № 11**

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

Паулу Антонию Жоау

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
3.1	Написание программ . . . . .	6
<b>4</b>	<b>Выводы</b>	<b>14</b>
<b>5</b>	<b>Ответы на контрольные вопросы</b>	<b>15</b>

## Список иллюстраций

3.1	Скрипт 1	. . . . .	6
3.2	Скрипт 1	. . . . .	7
3.3	Скрипт 1	. . . . .	7
3.4	Скрипт 1	. . . . .	7
3.5	Скрипт 1	. . . . .	8
3.6	Скрипт 2	. . . . .	8
3.7	Скрипт 2	. . . . .	9
3.8	Скрипт 2	. . . . .	9
3.9	Скрипт 2	. . . . .	9
3.10	Скрипт 2	. . . . .	10
3.11	Скрипт 3	. . . . .	10
3.12	Скрипт 3	. . . . .	11
3.13	Скрипт 3	. . . . .	11
3.14	Скрипт 3	. . . . .	11
3.15	Скрипт 4	. . . . .	12
3.16	Скрипт 4	. . . . .	12
3.17	Скрипт 4	. . . . .	12
3.18	Скрипт 4	. . . . .	13
3.19	Скрипт 4	. . . . .	13

# 1 Цель работы

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

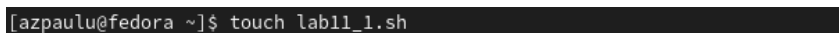
## 2 Задание

- Ознакомиться с теоретическим материалом.
- Выполнить упражнения.
- Ответить на контрольные вопросы.

## 3 Выполнение лабораторной работы

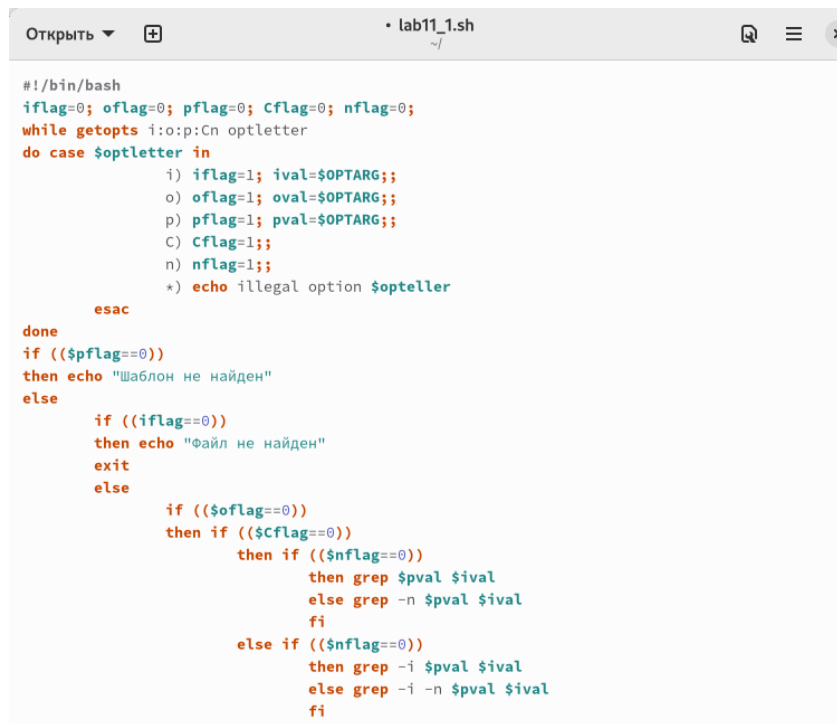
### 3.1 Написание программ

1. Используя команды `getopts` `grep`, написали командный файл, который анализирует командную строку с ключами: `-iinputfile` — прочитать данные из указанного файла; `-ooutputfile` — вывести данные в указанный файл; `-r` — 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 `-r`. Сделали файл исполняемым, проверили его работу. (рис. [3.1]), (рис. [3.2]), (рис. [3.3]), (рис. [3.4]), (рис. [3.5])



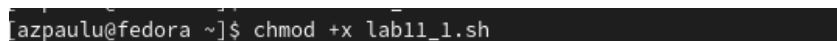
```
[azpaulu@fedora ~]$ touch lab11_1.sh
```

Рис. 3.1: Скрипт 1



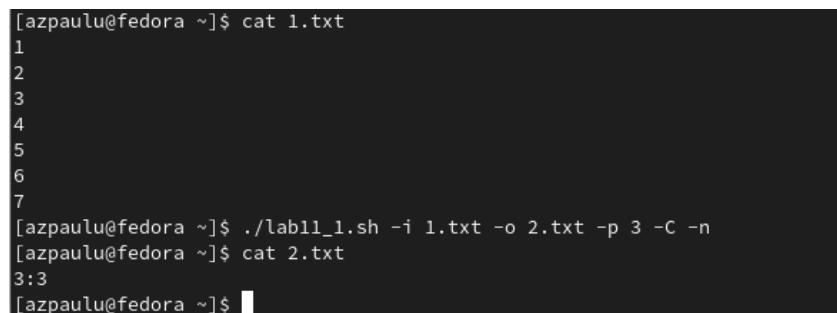
```
#!/bin/bash
iflag=0; oflag=0; pflag=0; Cflag=0; nflag=0;
while getopts i:o:p:Cn optletter
do case $optletter in
    i) iflag=1; ival=$OPTARG;;
    o) oflag=1; oval=$OPTARG;;
    p) pflag=1; pval=$OPTARG;;
    C) Cflag=1;;
    n) nflag=1;;
    *) echo illegal option $optletter
       esac
done
if (($pflag==0))
then echo "Шаблон не найден"
else
    if (($iflag==0))
    then echo "Файл не найден"
    exit
    else
        if (($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
        fi
    fi
fi
```

Рис. 3.2: Скрипт 1



```
[azpaulu@fedora ~]$ chmod +x lab11_1.sh
```

Рис. 3.3: Скрипт 1



```
[azpaulu@fedora ~]$ cat 1.txt
1
2
3
4
5
6
7
[azpaulu@fedora ~]$ ./lab11_1.sh -i 1.txt -o 2.txt -p 3 -C -n
[azpaulu@fedora ~]$ cat 2.txt
3:3
[azpaulu@fedora ~]$
```

Рис. 3.4: Скрипт 1

```
318
[azpaulu@fedora ~]$ ./lab11_1.sh -i 1.txt -o -p 3 -C -n
Шаблон не найден
[azpaulu@fedora ~]$ ./lab11_1.sh -o 2.txt -p 3 -C -n
Файл не найден
[azpaulu@fedora ~]$
```

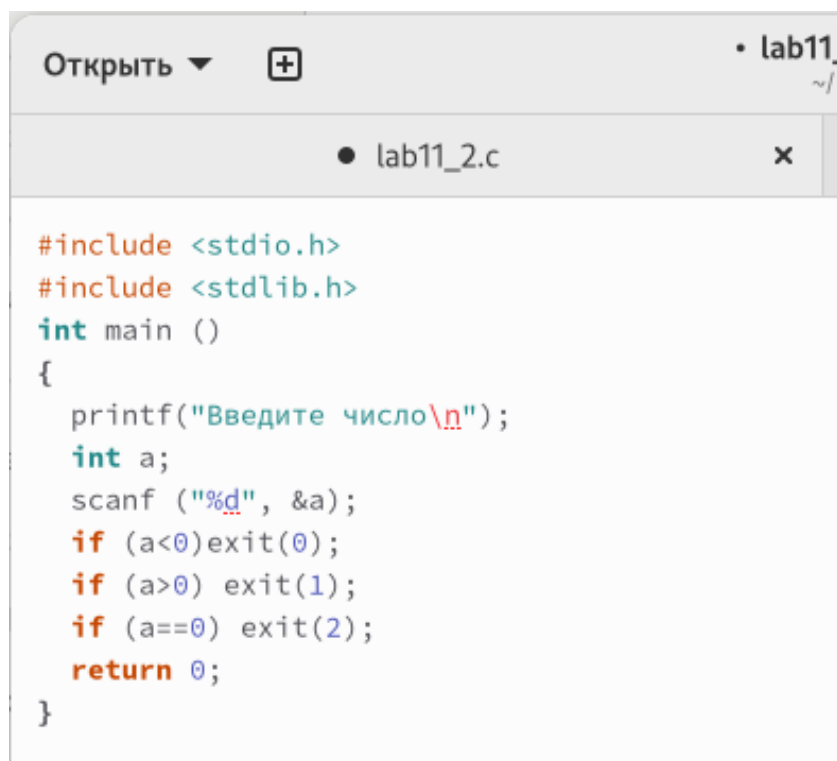
Рис. 3.5: Скрипт 1

2. Создали два файла. Написали 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 вызывает эту программу и, проанализировав с помощью команды `$?`, выдаёт сообщение о том, какое число было введено. Сделали файл исполняемым и проверили его работу. (рис. [3.6]), (рис. [3.7]), (рис. [3.8]), (рис. [3.9]), (рис. [3.10])

```
Файл не найден
[azpaulu@fedora ~]$ touch lab11_2.sh
[azpaulu@fedora ~]$ touch lab11_2.c
```

Рис. 3.6: Скрипт 2





The screenshot shows a code editor window with a tab labeled 'lab11\_2.c'. The code is a C program that prompts the user to enter a number and then checks if it is less than, greater than, or equal to zero. The code is as follows:

```
#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;
}
```

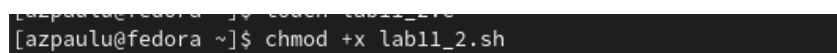
Рис. 3.7: Скрипт 2



The screenshot shows a code editor window with two tabs: 'lab11\_2.c' and 'lab11\_2.sh'. The 'lab11\_2.sh' tab is active, showing a shell script that compiles the C program and runs it, then checks the exit code and prints a message based on the result. The script is as follows:

```
#!/bin/bash
gcc lab11_2.c -o lab11_2
./lab11_2
code=$?
case $code in
    0) echo "Число меньше 0";;
    1) echo "Число больше 0";;
    2) echo "Число равно 0";;
esac
```

Рис. 3.8: Скрипт 2



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

```
[azpaulu@fedora ~]$ chmod +x lab11_2.sh
```

Рис. 3.9: Скрипт 2

```

[azpaulu@fedora ~]$ ./lab11_2.sh
Введите число
1
Число больше 0
[azpaulu@fedora ~]$ ./lab11_2.sh
Введите число
-9
Число меньше 0
[azpaulu@fedora ~]$ ./lab11_2.sh
Введите число
0
Число равно 0

```

Рис. 3.10: Скрипт 2

3. Написали 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). Дали файлу права на исполнение и проверили работу. (рис. [3.11]), (рис. [3.12]), (рис. [3.13]), (рис. [3.14])

```

[azpaulu@fedora ~]$ touch lab11_3.sh

```

Рис. 3.11: Скрипт 3

```
Открыть ▾ + lab11_3.sh
~/

#!/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
}
```

Рис. 3.12: Скрипт 3

```
[azpaulu@fedora ~]$ chmod +x lab11_3.sh
```

Рис. 3.13: Скрипт 3

```
[azpaulu@fedora ~]$ ./lab11_3.sh -c 1.txt 5
[azpaulu@fedora ~]$ ls
backup      lab10_3.sh~  lab11_2.c   Документы   'Рабочий стол'
lab09.sh    lab10_4.sh  lab11_2.sh  Загрузки    Шаблоны
lab10_1.sh  lab10_4.sh~ lab11_3.sh  Изображения
lab10_2.sh  lab11_1.sh  work        Музыка
lab10_3.sh  lab11_2     Видео       Общедоступные
[azpaulu@fedora ~]$ ./lab11_3.sh -r 1.txt 5
[azpaulu@fedora ~]$ ls
backup      lab10_3.sh~  lab11_2.c   Документы   'Рабочий стол'
lab09.sh    lab10_4.sh  lab11_2.sh  Загрузки    Шаблоны
lab10_1.sh  lab10_4.sh~ lab11_3.sh  Изображения
lab10_2.sh  lab11_1.sh  work        Музыка
lab10_3.sh  lab11_2     Видео       Общедоступные
```

Рис. 3.14: Скрипт 3

4. Написали командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся

в виде аргумента командной строки. Дали файлу права на исполнение и проверили работу. (рис. [3.14]), (рис. [3.15]), (рис. [3.16])

```
[azpaulu@fedora ~]$ touch lab11_4.sh
```

Рис. 3.15: Скрипт 4

A screenshot of a code editor window titled 'lab11\_4.sh'. The editor shows a bash script that finds files in the current directory and its subdirectories (up to a depth of 1), filters them by modification time (within the last 7 days), and then creates a tar archive of the files. The script uses 'find', 'echo', 'cut', and 'tar' commands. The file path shown in the title bar is '~/'.

```
#!/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
```

Рис. 3.16: Скрипт 4

```
[azpaulu@fedora ~]$ chmod +x lab11_4.sh
```

Рис. 3.17: Скрипт 4

```
[azpaulu@fedora ~]$ ./lab11_4.sh
.cache/
.cache/mesa_shader_cache/
.cache/mesa_shader_cache/index
.cache/mesa_shader_cache/45/
.cache/mesa_shader_cache/45/087bb9d067b72e4ae4410072007a4333b3d011
.cache/mesa_shader_cache/a5/
.cache/mesa_shader_cache/a5/0e6f48985dfbdc0389e2f2e103b3a846b6aaba
.cache/mesa_shader_cache/a5/7e22bca5650285dd9e4527949759cc403a2c8e
.cache/mesa_shader_cache/a5/f11f6e003779cf3a7cff9f3e2ddda9dcbf82ed
.cache/mesa_shader_cache/58/
.cache/mesa_shader_cache/58/016bb235fd37eeb3d521b9985fff9edcc72611
.cache/mesa_shader_cache/58/7de943071c254470ddcdef5a3de32fc4c549db
.cache/mesa_shader_cache/ff/
.cache/mesa_shader_cache/ff/5ddb2f79d46c5b4eba6c6afb2b3bdd48f69748
.cache/mesa_shader_cache/8f/
.cache/mesa_shader_cache/8f/61378b62853479f9c30e43660d434fb5f6b74c
.cache/mesa_shader_cache/b6/
.cache/mesa_shader_cache/b6/372c0597390ad489841891b52cdacd39cdde99
.cache/mesa_shader_cache/b6/453b42ee3ef41a713530005660f214997ac9f0
```

Рис. 3.18: Скрипт 4

```
[azpaulu@fedora ~]$ ls
azpaulu.tar  lab10_3.sh  lab11_2  work  Музыка
backup      lab10_3.sh~ lab11_2.c Видео  Общедоступные
lab09.sh    lab10_4.sh  lab11_2.sh Документы  'Рабочий стол'
lab10_1.sh  lab10_4.sh~ lab11_3.sh Загрузки  Шаблоны
lab10_2.sh  lab11_1.sh  lab11_4.sh Изображения
```

Рис. 3.19: Скрипт 4

## 4 Выводы

В ходе выполнения лабораторной работы были изучены основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 5 Ответы на 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 mans/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.