

**Лабораторная работа No 11.**  
**Программирование в командном**  
**процессоре ОС UNIX. Ветвления и циклы**

Горайнова Алёна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>4</b>	<b>Выводы</b>	<b>13</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>14</b>

# Список иллюстраций

3.1	Код . . . . .	8
3.2	. . . . .	9
3.3	. . . . .	10
3.4	Код . . . . .	10
3.5	Код . . . . .	11
3.6	. . . . .	11
3.7	Код . . . . .	12
3.8	. . . . .	12

## **Список таблиц**

# 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

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` — указать шаблон для поиска;
  - `С` — различать большие и малые буквы;
  - `п` — 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 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).
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 Выполнение лабораторной работы

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;
- `ршаблон` — указать шаблон для поиска;
- `С` — различать большие и малые буквы;
- `п` — выдавать номера строк, а затем ищет в указанном файле нужные строки, определяемые ключом `-р`. (рис. ??, ??)

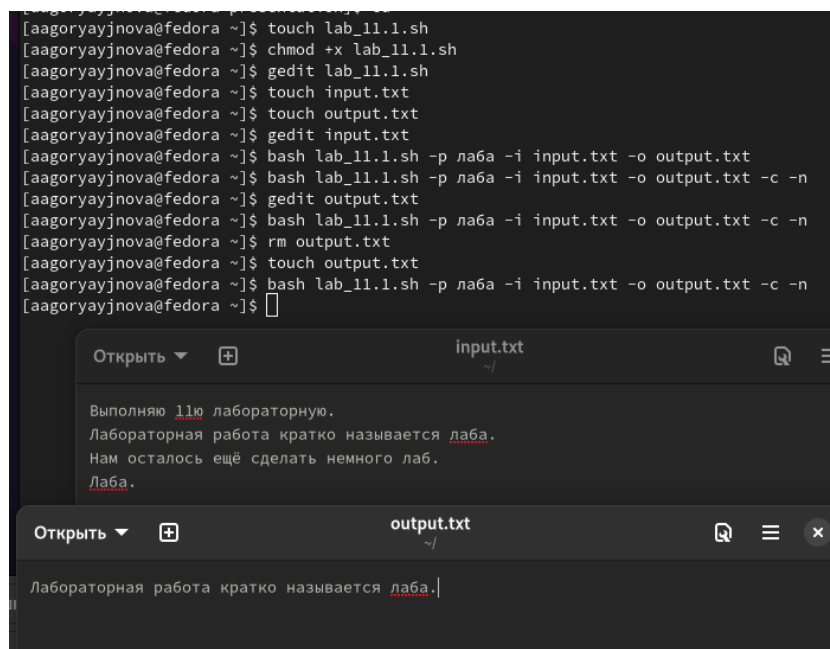


```
lab_11.1.sh
~/
1 #! /bin/bash
2
3 while getopts i:o:p:cn optletter
4 do
5     case $optletter in
6         i) iflag=1; ival=$OPTARG;;
7         o) oflag=1; oval=$OPTARG;;
8         p) pflag=1; pval=$OPTARG;;
9         c) cflag=1;;
10        n) nflag=1;;
11        *) echo Illegal option $optletter;;
12    esac
13 done
14
15 if ! test $cflag
16 then
17     cf=-i
18 fi
19
20 if test $nlag
21 then
22     nf=-n
23 fi
24
25 grep $scf $inf $pval $ival >> $oval
```

Рис. 3.1: Код



```
[aagoryayjnova@fedora ~]$ touch lab_11.1.sh
[aagoryayjnova@fedora ~]$ chmod +x lab_11.1.sh
[aagoryayjnova@fedora ~]$ gedit lab_11.1.sh
[aagoryayjnova@fedora ~]$ touch input.txt
[aagoryayjnova@fedora ~]$ touch output.txt
[aagoryayjnova@fedora ~]$ gedit input.txt
[aagoryayjnova@fedora ~]$ bash lab_11.1.sh -p лаба -i input.txt -o output.txt
[aagoryayjnova@fedora ~]$ bash lab_11.1.sh -p лаба -i input.txt -o output.txt -c -n
[aagoryayjnova@fedora ~]$ gedit output.txt
[aagoryayjnova@fedora ~]$ bash lab_11.1.sh -p лаба -i input.txt -o output.txt -c -n
[aagoryayjnova@fedora ~]$ rm output.txt
[aagoryayjnova@fedora ~]$ touch output.txt
[aagoryayjnova@fedora ~]$ bash lab_11.1.sh -p лаба -i input.txt -o output.txt -c -n
[aagoryayjnova@fedora ~]$
```



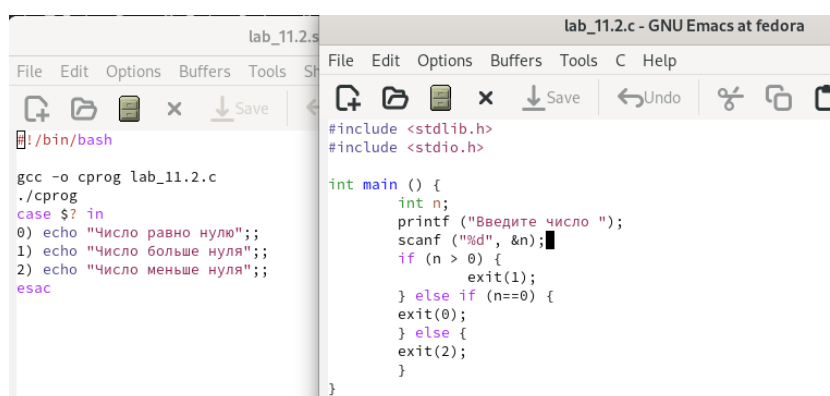
The screenshot shows a terminal window with a dark background. The user is at the prompt [aagoryayjnova@fedora ~]. They have created a script lab\_11.1.sh, made it executable, and opened it in gedit. They have also created input.txt and output.txt files. The script lab\_11.1.sh is executed with the command bash lab\_11.1.sh -p лаба -i input.txt -o output.txt. The script is then executed again with the command bash lab\_11.1.sh -p лаба -i input.txt -o output.txt -c -n. The output.txt file is then opened in gedit, showing the text: Выполняю 11ю лабораторную. Лабораторная работа кратко называется лаба. Нам осталось ещё сделать немного лаб. Лаба. The output.txt file is then removed with the command rm output.txt, and a new output.txt file is created with the command touch output.txt. Finally, the script is executed again with the command bash lab\_11.1.sh -p лаба -i input.txt -o output.txt -c -n. The output.txt file is then opened in gedit, showing the text: Лабораторная работа кратко называется лаба.

Рис. 3.2:

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. (рис. 3.3, 3.4)

```
[aagoryayjnova@fedora ~]$ gedit lab_11.2.c
[aagoryayjnova@fedora ~]$ bash lab_11.2.sh
Введите число 10
Число больше нуля
[aagoryayjnova@fedora ~]$ bash lab_11.2.sh
Введите число 0
Число равно нулю
[aagoryayjnova@fedora ~]$ bash lab_11.2.sh
Введите число -10
Число меньше нуля
[aagoryayjnova@fedora ~]$
```

Рис. 3.3:



```
lab_11.2.s
File Edit Options Buffers Tools Sh
[bin/bash

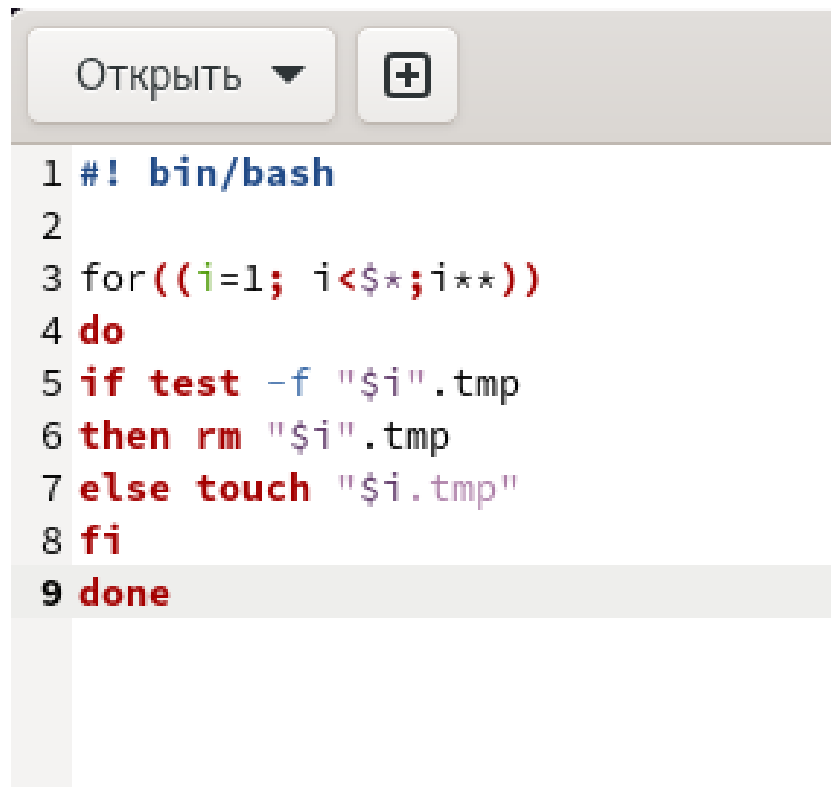
gcc -o cprog lab_11.2.c
./cprog
case $? in
0) echo "Число равно нулю";;
1) echo "Число больше нуля";;
2) echo "Число меньше нуля";;
esac

lab_11.2.c - GNU Emacs at fedora
File Edit Options Buffers Tools C Help
#include <stdlib.h>
#include <stdio.h>

int main () {
    int n;
    printf ("Введите число ");
    scanf ("%d", &n);
    if (n > 0) {
        exit(1);
    } else if (n==0) {
        exit(0);
    } else {
        exit(2);
    }
}
```

Рис. 3.4: Код

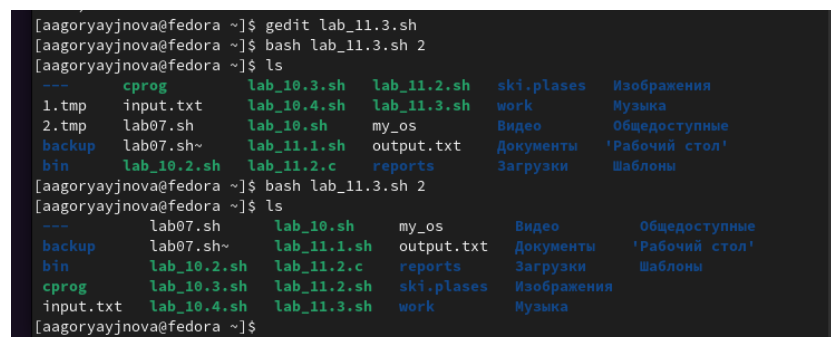
3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют). (рис. 3.5, 3.6)



The screenshot shows a code editor window with a toolbar at the top containing a button labeled "Открыть" (Open) with a dropdown arrow and a button with a plus sign. The code is a shell script with the following lines:

```
1 #! bin/bash
2
3 for((i=1; i<$*;i**))
4 do
5 if test -f "$i".tmp
6 then rm "$i".tmp
7 else touch "$i.tmp"
8 fi
9 done
```

Рис. 3.5: Код

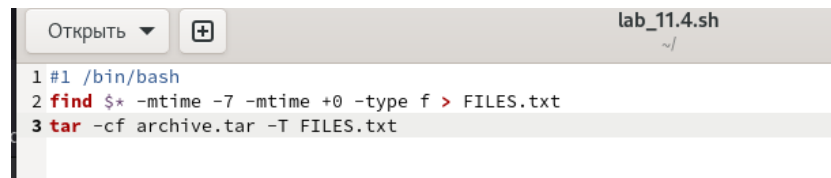


The screenshot shows a terminal window with the following commands and output:

```
[aagoryayjnova@fedora ~]$ gedit lab_11.3.sh
[aagoryayjnova@fedora ~]$ bash lab_11.3.sh 2
[aagoryayjnova@fedora ~]$ ls
----
1.tmp      cprog      lab_10.3.sh  lab_11.2.sh  ski.plases  Изображения
2.tmp      lab07.sh   lab_10.4.sh  lab_11.3.sh  work        Музыка
backup     lab07.sh~  lab_10.sh    my_os        Видео       Общедоступные
lab07.sh~  lab_11.1.sh output.txt   Документы   'Рабочий стол'
bin        lab_10.2.sh lab_11.2.c   reports      Загрузки    Шаблоны
[aagoryayjnova@fedora ~]$ bash lab_11.3.sh 2
[aagoryayjnova@fedora ~]$ ls
----
lab07.sh   lab_10.sh    my_os        Видео       Общедоступные
backup     lab07.sh~   lab_11.1.sh  output.txt  Документы   'Рабочий стол'
bin        lab_10.2.sh lab_11.2.c   reports      Загрузки    Шаблоны
cprog      lab_10.3.sh lab_11.2.sh  ski.plases  Изображения
input.txt  lab_10.4.sh lab_11.3.sh  work        Музыка
[aagoryayjnova@fedora ~]$
```

Рис. 3.6:

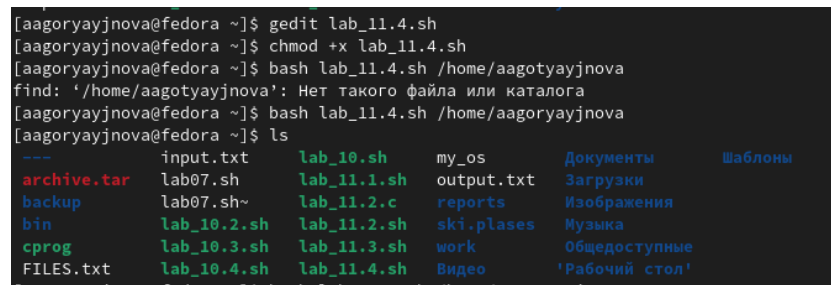
4. Написать командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`). (рис. 3.7, 3.8)



The screenshot shows a code editor window titled "lab\_11.4.sh" with a toolbar containing "Открыть" (Open) and a "+" icon. The editor contains the following code:

```
1 #! /bin/bash
2 find $* -mtime -7 -mtime +0 -type f > FILES.txt
3 tar -cf archive.tar -T FILES.txt
```

Рис. 3.7: Код



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

```
[aagoryayjnova@fedora ~]$ gedit lab_11.4.sh
[aagoryayjnova@fedora ~]$ chmod +x lab_11.4.sh
[aagoryayjnova@fedora ~]$ bash lab_11.4.sh /home/aagoryayjnova
find: '/home/aagoryayjnova': Нет такого файла или каталога
[aagoryayjnova@fedora ~]$ bash lab_11.4.sh /home/aagoryayjnova
[aagoryayjnova@fedora ~]$ ls
```

The output of the `ls` command is displayed in a colorized format:

---	input.txt	lab_10.sh	my_os	Документы	Шаблоны
archive.tar	lab07.sh	lab_11.1.sh	output.txt	Загрузки	
backup	lab07.sh~	lab_11.2.c	reports	Изображения	
bin	lab_10.2.sh	lab_11.2.sh	ski.places	Музыка	
cprog	lab_10.3.sh	lab_11.3.sh	work	Общедоступные	
FILES.txt	lab_10.4.sh	lab_11.4.sh	Видео	'Рабочий стол'	

Рис. 3.8:

## 4 Выводы

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

## 5 Контрольные вопросы

### 1. Каково предназначение команды getopt?

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 -ifile_in.txt -ofile_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) oflag = 1; oval =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 равно `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, следова-

тельно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

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

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. Какие операторы управления действиями вы знаете?

Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости отрезультатов 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. Какие операторы используются для прерывания цикла?

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

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).

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

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

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. :::  
{#refs} :::