

Лабораторная работа №11

НКАбд-01-22

Никита Михайлович Демидович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	10
5	Контрольные вопросы	26
6	Выводы	30
	Список литературы	31

Список иллюстраций

4.1	Создание директории lab11, файлов input.txt, output.txt и исполняемого файла programm1.sh	11
4.2	Исполняемый файл programm1.sh	12
4.3	Файл input.txt после работы исполняемого файла programm1.sh .	13
4.4	Файл output.txt после работы исполняемого файла programm1.sh .	14
4.5	Создание исполняемого файла programm2.sh	15
4.6	Исполняемый файл programm2.sh	16
4.7	Создание исполняемого файла 12.c	17
4.8	Исполняемый файл 12.c	18
4.9	Работа исполняемого файла programm2.sh	19
4.10	Создание исполняемого файла programm3.sh	20
4.11	Исполняемый файл programm3.sh	21
4.12	Работа исполняемого файла programm3.sh	22
4.13	Создание исполняемого файла programm4.sh	23
4.14	Исполняемый файл programm4.sh	24
4.15	Работа исполняемого файла programm4.sh	25

Список таблиц

1 Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX, научиться 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`шаблон — указать шаблон для поиска;
- `-С` — 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 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на

базе оболочки Korn. Рассмотрим основные элементы программирования в оболочке `bash` (в разделе “Выполнение лабораторной работы”). В других оболочках большинство команд будет совпадать с описанными ниже.

4 Выполнение лабораторной работы

На первом этапе выполнения работы я создал отдельную директорию для дальнейшей работы, командный файл `programm1.sh`, текстовый файлы `input.txt` и `output.txt` и приступил к написанию файла с командами `getopts` и `grep`, 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 `-p` (рис. [4.1]) - (рис. [4.2]):

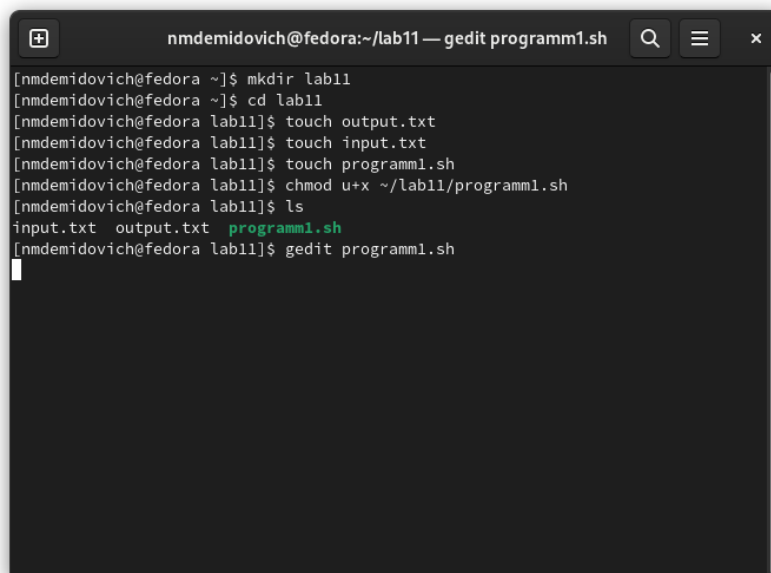
```
#!/bin/bash

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 ! test $cflag
then
    cf=-i
fi

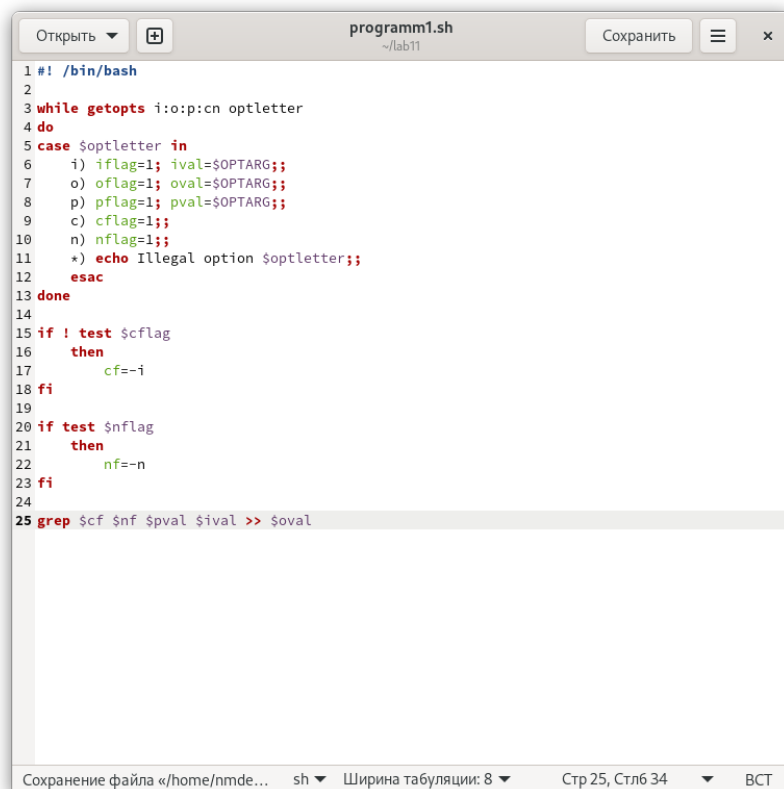
if test $nflag
then
    nf=-n
fi

grep $cf $nf $pval $ival >> $oval
```



```
nmdemidovich@fedora:~/lab11 — gedit programm1.sh
[nmdemidovich@fedora ~]$ mkdir lab11
[nmdemidovich@fedora ~]$ cd lab11
[nmdemidovich@fedora lab11]$ touch output.txt
[nmdemidovich@fedora lab11]$ touch input.txt
[nmdemidovich@fedora lab11]$ touch programm1.sh
[nmdemidovich@fedora lab11]$ chmod u+x ~/lab11/programm1.sh
[nmdemidovich@fedora lab11]$ ls
input.txt  output.txt  programm1.sh
[nmdemidovich@fedora lab11]$ gedit programm1.sh
```

Рис. 4.1: Создание директории lab11, файлов input.txt, output.txt и исполняемого файла programm1.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 $nflag
21 then
22     nf=-n
23 fi
24
25 grep $cf $nf $pval $ival >> $oval
```

Рис. 4.2: Исполняемый файл programm1.sh

Затем я проверил работу исполняемого файла (рис. [4.3]) - (рис. [4.4]):

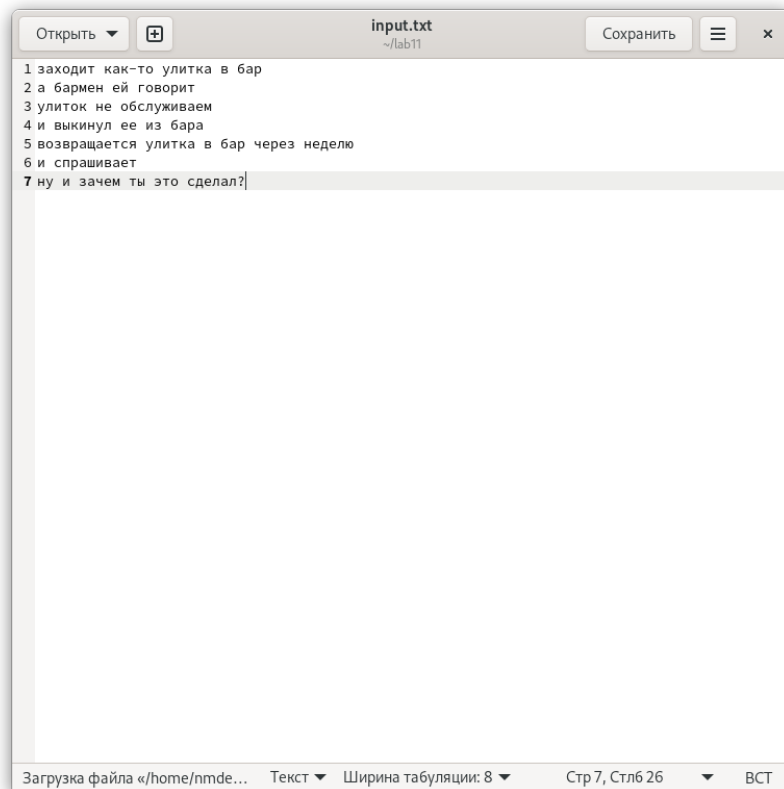


Рис. 4.3: Файл input.txt после работы исполняемого файла programm1.sh

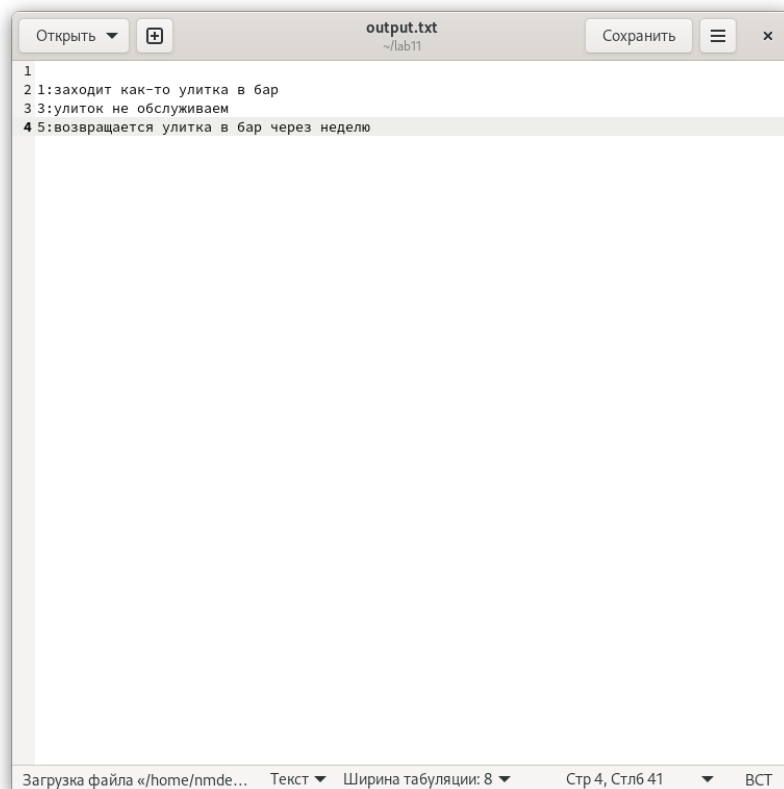


Рис. 4.4: Файл output.txt после работы исполняемого файла programm1.sh

После этого я создал исполняемый файл для второй программы:

```
#!/bin/bash
```

```
gcc -o cprog 12.c
```

```
./cprog
```

```
case $? in
```

```
0) echo "Число равно нулю";;
```

```
1) echo "Число больше нуля";;
```

```
2) echo "Число меньше нуля";;
```

```
esac
```

А также файл 12.c для программы на языке Си (рис. [4.5]) - (рис. [4.9]):

```
#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);
    }
}
```

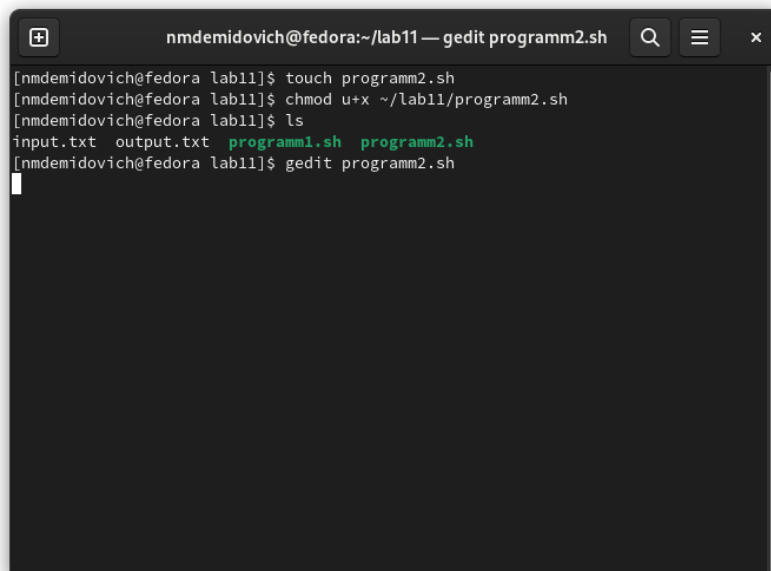
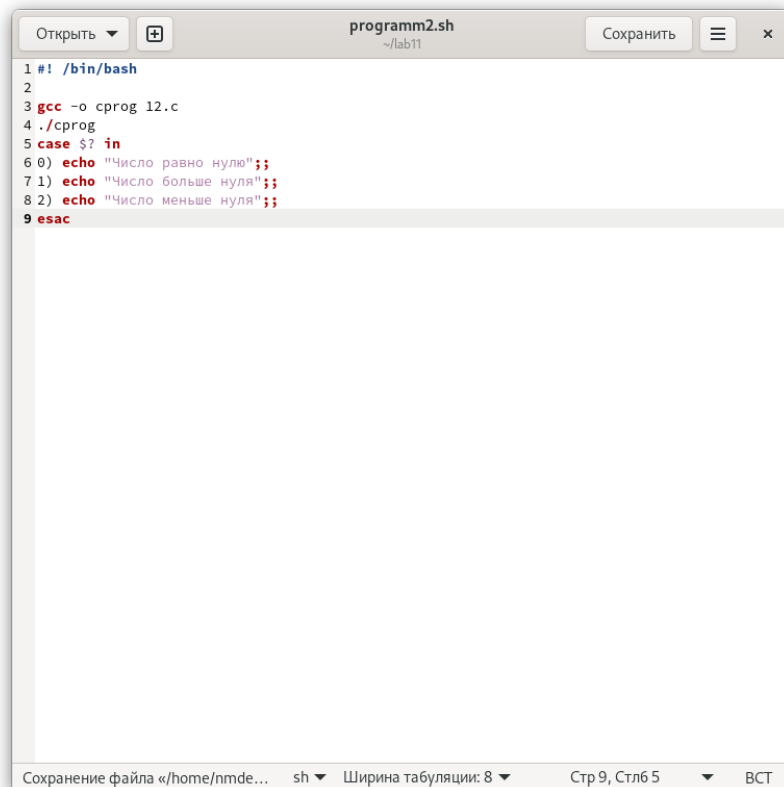


Рис. 4.5: Создание исполняемого файла programm2.sh

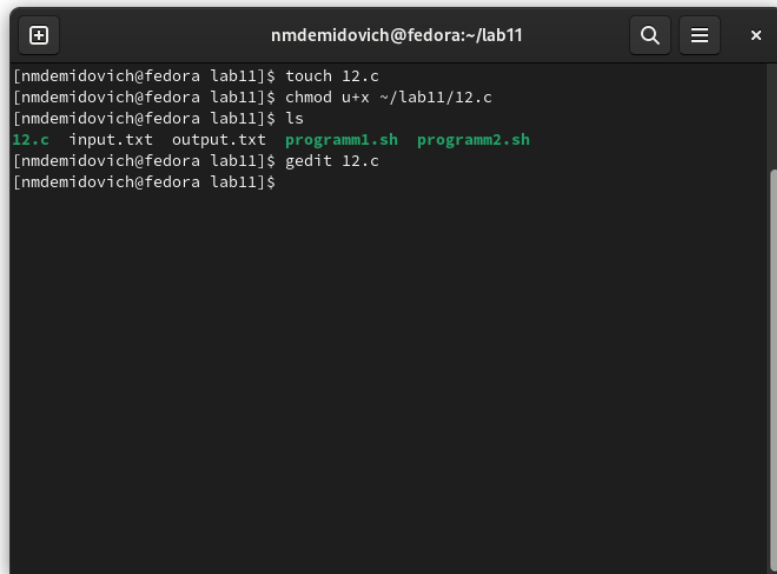


The image shows a text editor window with the title bar "programm2.sh" and a subtitle "~/.lab11". The window contains a shell script with the following lines:

```
1 #! /bin/bash
2
3 gcc -o cprog 12.c
4 ./cprog
5 case $? in
6 0) echo "Число равно нулю";;
7 1) echo "Число больше нуля";;
8 2) echo "Число меньше нуля";;
9 esac
```

The status bar at the bottom of the window displays the following information: "Сохранение файла «/home/nmde... sh", "Ширина табуляции: 8", "Стр 9, Стлб 5", and "ВСТ".

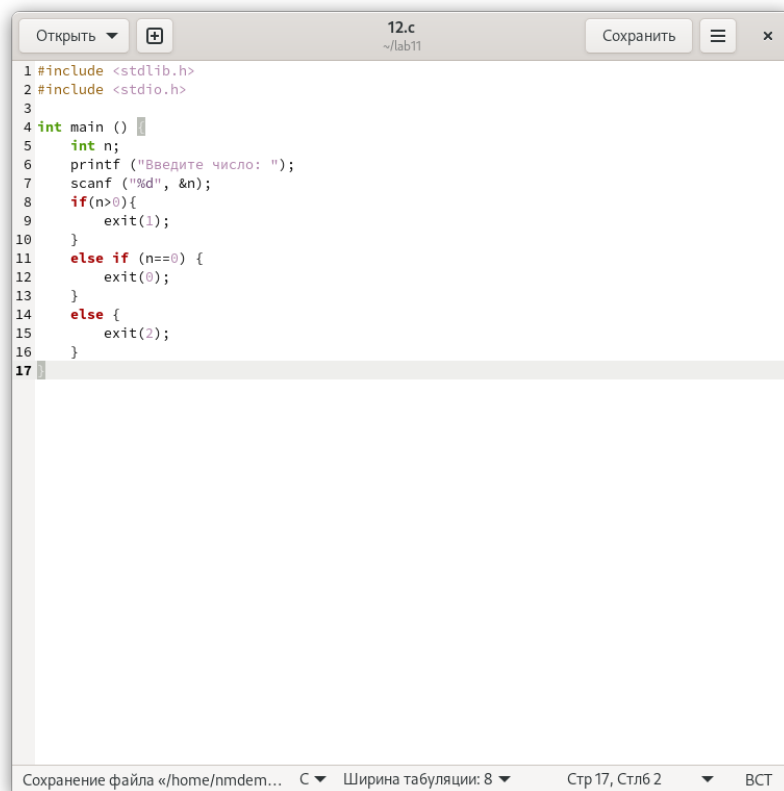
Рис. 4.6: Исполняемый файл programm2.sh

A terminal window titled 'nmdemidovich@fedora:~/lab11' with search, menu, and close icons. The terminal shows the following commands and output:

```
[nmdemidovich@fedora lab11]$ touch 12.c
[nmdemidovich@fedora lab11]$ chmod u+x ~/lab11/12.c
[nmdemidovich@fedora lab11]$ ls
12.c input.txt output.txt programm1.sh programm2.sh
[nmdemidovich@fedora lab11]$ gedit 12.c
[nmdemidovich@fedora lab11]$
```

The terminal window displays the execution of several commands to create and configure a file named 12.c. The commands are: touch 12.c, chmod u+x ~/lab11/12.c, ls, gedit 12.c, and a final prompt. The output of the ls command shows the current directory contents, including 12.c, input.txt, output.txt, programm1.sh, and programm2.sh.

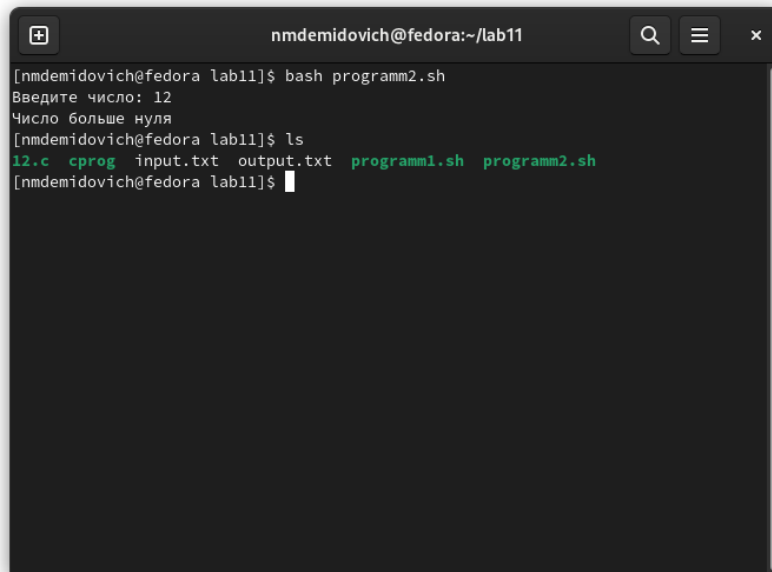
Рис. 4.7: Создание исполняемого файла 12.c



```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main () {
5     int n;
6     printf ("Введите число: ");
7     scanf ("%d", &n);
8     if (n > 0) {
9         exit(1);
10    }
11    else if (n == 0) {
12        exit(0);
13    }
14    else {
15        exit(2);
16    }
17 }
```

Сохранение файла «/home/nmdem...» С ▾ Ширина табуляции: 8 ▾ Стр 17, Стлб 2 ▾ ВСТ

Рис. 4.8: Исполняемый файл 12.c

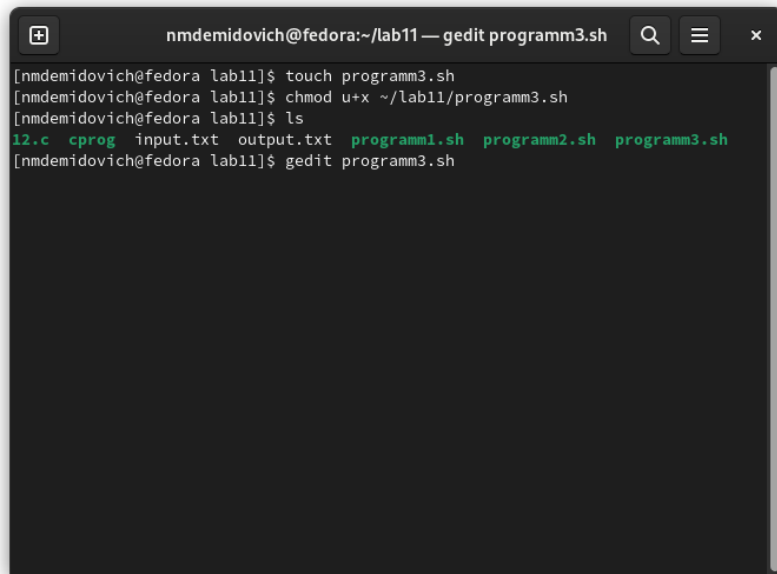
A screenshot of a terminal window titled 'nmdemidovich@fedora:~/lab11'. The terminal shows the execution of a script 'programm2.sh'. The script prompts for a number, and the user enters '12'. The script then checks if the number is greater than zero. Finally, it runs the command 'ls', which lists the files '12.c', 'cprog', 'input.txt', 'output.txt', 'programm1.sh', and 'programm2.sh' in the current directory. The terminal window has a dark background and standard window controls at the top.

```
nmdemidovich@fedora lab11]$ bash programm2.sh
Введите число: 12
Число больше нуля
nmdemidovich@fedora lab11]$ ls
12.c  cprog  input.txt  output.txt  programm1.sh  programm2.sh
nmdemidovich@fedora lab11]$
```

Рис. 4.9: Работа исполняемого файла programm2.sh

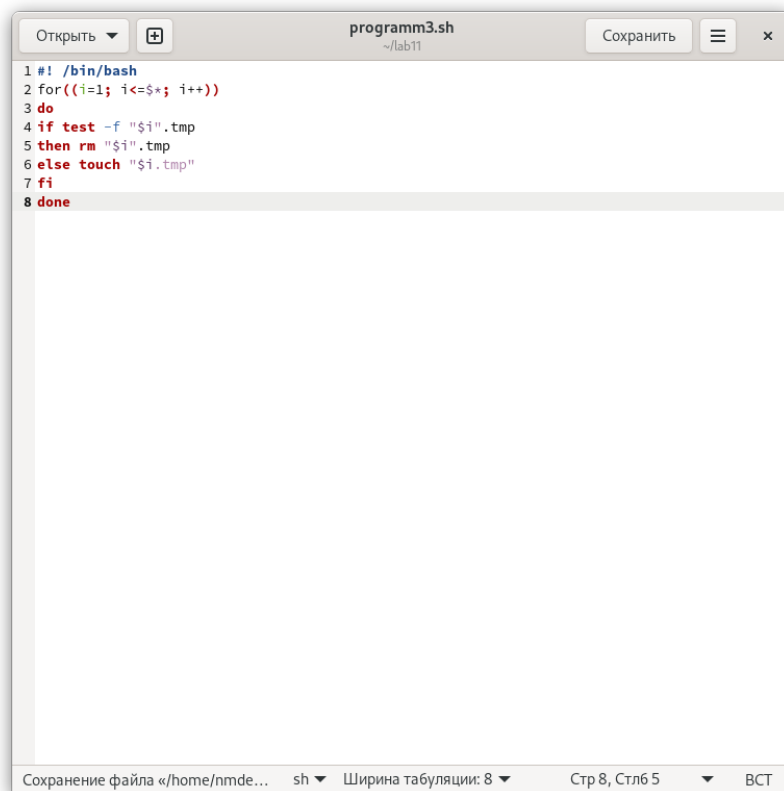
Далее я написал командный файл programm3.sh, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.) число файлов, которые необходимо создать, которое передается в аргументы командной строки (рис. [4.10]) - (рис. [4.12]):

```
#!/bin/bash
for((i=1; i<=$*; i++))
do
if test -f "$i".tmp
then rm "$i".tmp
else touch "$i.tmp"
fi
done
```

A terminal window titled 'nmdemidovich@fedora:~/lab11 — gedit programm3.sh'. The window shows a series of commands being executed in a shell. The first three commands are 'touch programm3.sh', 'chmod u+x ~/lab11/programm3.sh', and 'ls'. The output of 'ls' is displayed on the next line, showing '12.c cprog input.txt output.txt programm1.sh programm2.sh programm3.sh'. The final command is 'gedit programm3.sh', which opens the file in the gedit editor. The terminal has a dark background and green text for the prompt and command output.

```
nmdemidovich@fedora lab11]$ touch programm3.sh
nmdemidovich@fedora lab11]$ chmod u+x ~/lab11/programm3.sh
nmdemidovich@fedora lab11]$ ls
12.c cprog input.txt output.txt programm1.sh programm2.sh programm3.sh
nmdemidovich@fedora lab11]$ gedit programm3.sh
```

Рис. 4.10: Создание исполняемого файла programm3.sh

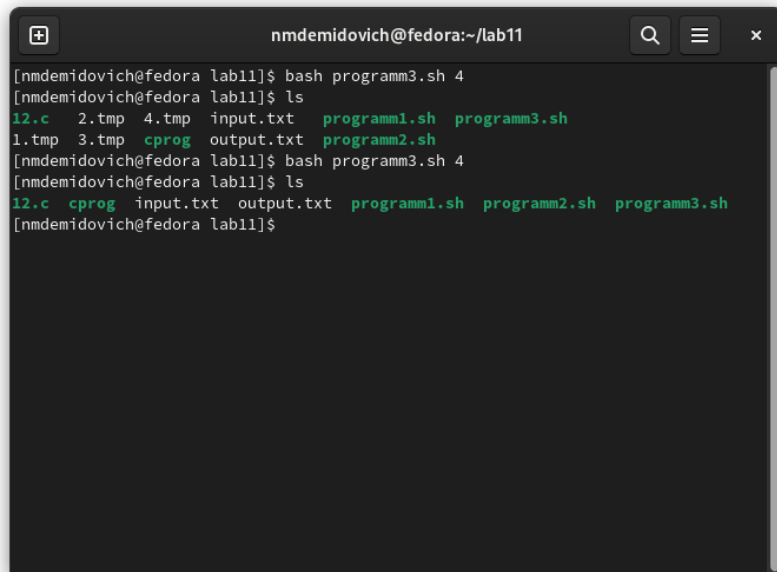


The image shows a terminal window with a title bar containing the filename 'programm3.sh' and the path '~/lab11'. The window has buttons for 'Открыть' (Open), 'Сохранить' (Save), and a close button 'x'. The main area displays the following shell script code:

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

The status bar at the bottom shows 'Сохранение файла «/home/nmde... sh', 'Ширина табуляции: 8', 'Стр 8, Стлб 5', and 'ВСТ'.

Рис. 4.11: Исполняемый файл programm3.sh

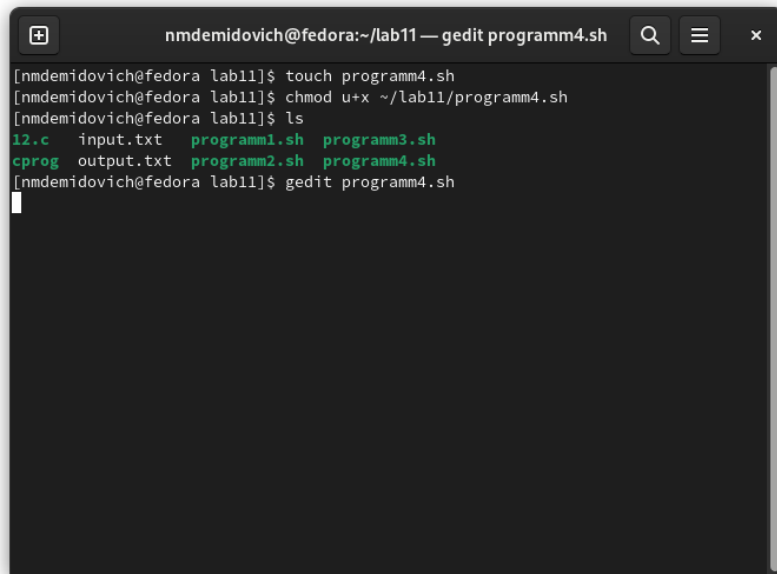
A terminal window titled 'nmdemidovich@fedora:~/lab11' with search, menu, and close icons. The terminal shows the execution of 'programm3.sh' with argument '4'. It lists files, runs 'cprog', and lists files again. The output shows files 2.tmp, 4.tmp, input.txt, programm1.sh, and programm3.sh. The second listing shows 1.tmp, 3.tmp, cprog, output.txt, and programm2.sh.

```
nmdemidovich@fedora lab11]$ bash programm3.sh 4
nmdemidovich@fedora lab11]$ ls
12.c  2.tmp  4.tmp  input.txt  programm1.sh  programm3.sh
1.tmp  3.tmp  cprog  output.txt  programm2.sh
nmdemidovich@fedora lab11]$ bash programm3.sh 4
nmdemidovich@fedora lab11]$ ls
12.c  cprog  input.txt  output.txt  programm1.sh  programm2.sh  programm3.sh
nmdemidovich@fedora lab11]$
```

Рис. 4.12: Работа исполняемого файла programm3.sh

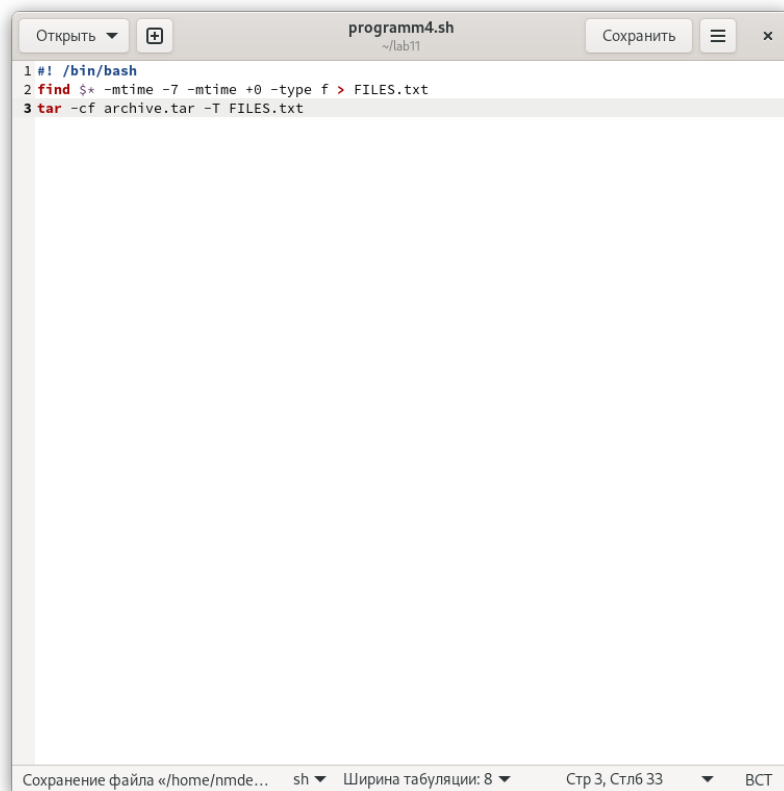
И на финальном этапе выполнения работы я создал исполняемый файл для четвертой программы. Это командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории (рис. [4.13]) - (рис. [4.15]):

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

A terminal window titled 'nmdemidovich@fedora:~/lab11 — gedit programm4.sh'. The window shows a series of commands being executed in a shell. The first command is 'touch programm4.sh', followed by 'chmod u+x ~/lab11/programm4.sh', and then 'ls'. The 'ls' command output shows a directory listing with files '12.c', 'input.txt', 'programm1.sh', 'programm3.sh', 'cprog', 'output.txt', 'programm2.sh', and 'programm4.sh'. The final command is 'gedit programm4.sh', which opens the file in the gedit editor. The terminal window has a dark background and a light-colored text. The window title bar includes a search icon, a menu icon, and a close icon.

```
nmdemidovich@fedora lab11]$ touch programm4.sh
nmdemidovich@fedora lab11]$ chmod u+x ~/lab11/programm4.sh
nmdemidovich@fedora lab11]$ ls
12.c  input.txt  programm1.sh  programm3.sh
cprog output.txt  programm2.sh  programm4.sh
nmdemidovich@fedora lab11]$ gedit programm4.sh
```

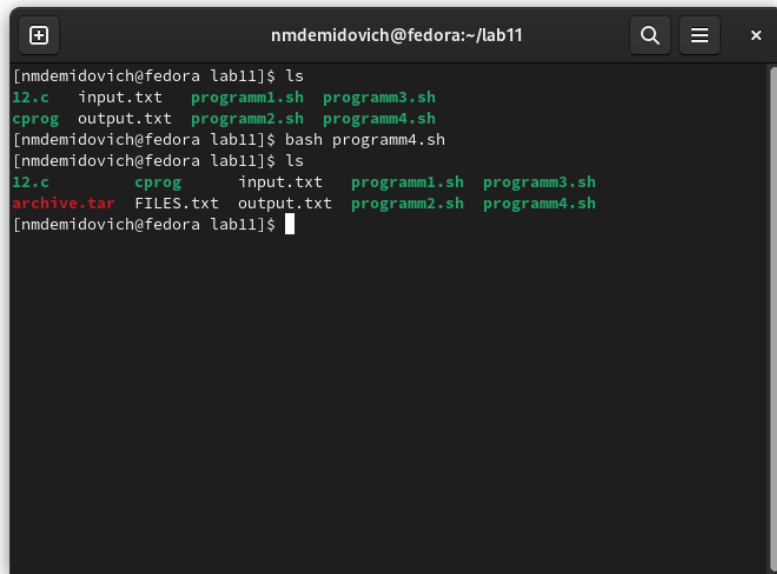
Рис. 4.13: Создание исполняемого файла programm4.sh



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

Сохранение файла «/home/nmde... sh ▾ Ширина табуляции: 8 ▾ Стр 3, Стлб 33 ▾ ВСТ

Рис. 4.14: Исполняемый файл programm4.sh

A terminal window titled 'nmdemidovich@fedora:~/lab11' with search and menu icons. It shows a sequence of commands and their outputs. The first 'ls' command lists files with green permissions. The second 'ls' command, after running 'bash programm4.sh', shows additional files like 'archive.tar' and 'FILES.txt' in red, indicating they are new or modified. The window has a dark background and a vertical scrollbar on the right.

```
[nmdemidovich@fedora lab11]$ ls
12.c  input.txt  programm1.sh  programm3.sh
cprog output.txt  programm2.sh  programm4.sh
[nmdemidovich@fedora lab11]$ bash programm4.sh
[nmdemidovich@fedora lab11]$ ls
12.c      cprog      input.txt  programm1.sh  programm3.sh
archive.tar  FILES.txt  output.txt  programm2.sh  programm4.sh
[nmdemidovich@fedora lab11]$
```

Рис. 4.15: Работа исполняемого файла programm4.sh

5 Контрольные вопросы

1. Каково предназначение команды `getopts`?

Осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable`. 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 $optletter in
o) iflag = 1; oval =OPTARG;;
i) iflag=1; ival=$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 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, 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`, встреченная 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.

6 Выводы

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

Список литературы

Лабораторная работа №11 (Архитектура ОС)