

# **Лабораторная работа №10**

**Дисциплина - операционные системы**

Волгин Иван Алексеевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
<b>5</b>	<b>Выводы</b>	<b>19</b>

## Список иллюстраций

4.1	Создаю нужные файлы . . . . .	11
4.2	Код программы . . . . .	12
4.3	Выполнение программы . . . . .	13
4.4	Создаю нужные файлы . . . . .	14
4.5	Код программы . . . . .	14
4.6	Код программы . . . . .	15
4.7	Выполнение программы . . . . .	15
4.8	Код программы . . . . .	16
4.9	Выполнение программы . . . . .	17
4.10	Код программы . . . . .	18
4.11	Выполнение программы . . . . .	18

## 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; `-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`.
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  $\infty$  (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).
4. Написать командный файл, который с помощью команды `tag` 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 оболочке 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 Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

- Оболочка `bash` поддерживает встроенные арифметические функции. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (`term`), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате типа `radix#number`, где `radix` (основание системы счисления) — любое число не более 26. Для большинства команд используются следующие основания систем исчисления: 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%). Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7. Команда `let` также расширяет другие выражения `let`, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда `let` не ограничена простыми арифметическими выражениями. Табл. 10.1 показывает полный набор `let`-операций. Подобно `C` оболочка `bash` может присваивать переменной любое значение, а произвольное выражение само имеет значение, которое может использоваться. При этом «ноль» воспринимается как «ложь», а любое другое значение выражения — как «истина». Для облегчения программирования можно записывать условия оболочки `bash` в двойные скобки — (( )).

Арифметические операторы оболочки `bash`

Оператор	Синтаксис	Результат
!	<code>!exp</code>	Если <code>exp</code> равно 0, то возвращает 1; иначе 0
!=	<code>exp1 != exp2</code>	Если <code>exp1</code> не равно <code>exp2</code> , то возвращает 1; иначе 0
%	<code>exp1 % exp2</code>	Возвращает остаток от деления <code>exp1</code> на <code>exp2</code>
%=	<code>var=%exp</code>	Присваивает остаток от деления <code>var</code> на <code>exp</code> переменной <code>var</code>
&	<code>exp1 &amp; exp2</code>	Возвращает побитовое AND выражений <code>exp1</code> и <code>exp2</code>



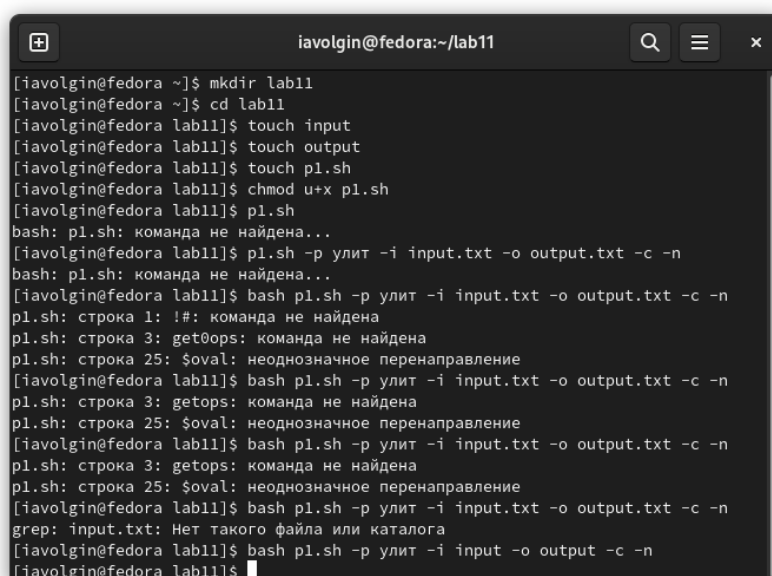
$\text{expr1} \& \text{expr2}$  Если и  $\text{expr1}$  и  $\text{expr2}$  не равны нулю, то возвращает 1; иначе 0  
 $\text{var} \&= \text{expr}$  Присваивает переменной  $\text{var}$  побитовое AND  $\text{var}$  и  $\text{expr}$   
 $\text{var} * \text{expr1} * \text{expr2}$  Умножает  $\text{expr1}$  на  $\text{expr2}$   
 $\text{var} = \text{expr}$  Умножает  $\text{expr}$  на значение переменной  $\text{var}$  и присваивает результат переменной  $\text{var}$   
 $\text{var} + \text{expr1} + \text{expr2}$  Складывает  $\text{expr1}$  и  $\text{expr2}$   
 $\text{var} += \text{expr}$  Складывает  $\text{expr}$  со значением переменной  $\text{var}$  и результат присваивает переменной  $\text{var}$   
 $-\text{expr}$  Операция отрицания  $\text{expr}$  (унарный минус)  
 $\text{expr1} - \text{expr2}$  Вычитает  $\text{expr2}$  из  $\text{expr1}$   
 $\text{var} -= \text{expr}$  Вычитает  $\text{expr}$  из значения переменной  $\text{var}$  и присваивает результат переменной  $\text{var}$   
 $\text{var} / \text{expr} / \text{expr2}$  Делит  $\text{expr1}$  на  $\text{expr2}$   
 $\text{var} /= \text{expr}$  Делит значение переменной  $\text{var}$  на  $\text{expr}$  и присваивает результат переменной  $\text{var}$   
 $\text{expr1} < \text{expr2}$  Если  $\text{expr1}$  меньше, чем  $\text{expr2}$ , то возвращает 1, иначе возвращает 0  
 $\text{expr1} \ll \text{expr2}$  Сдвигает  $\text{expr1}$  влево на  $\text{expr2}$  бит  
 $\text{var} \ll \text{expr}$  Побитовый сдвиг влево значения переменной  $\text{var}$  на  $\text{expr}$  бит  
 $\text{expr1} \leq \text{expr2}$  Если  $\text{expr1}$  меньше или равно  $\text{expr2}$ , то возвращает 1; иначе возвращает 0  
 $\text{var} = \text{expr}$  Присваивает значение  $\text{expr}$  переменной  $\text{var}$   
 $\text{expr1} == \text{expr2}$  Если  $\text{expr1}$  равно  $\text{expr2}$ , то возвращает 1; иначе возвращает 0  
 $\text{expr1} > \text{expr2}$  1, если  $\text{expr1}$  больше, чем  $\text{expr2}$ ; иначе 0  
 $\text{var} \geq \text{expr1} \geq \text{expr2}$  1, если  $\text{expr1}$  больше или равно  $\text{expr2}$ ; иначе 0  
 $\text{expr} \gg \text{expr2}$  Сдвигает  $\text{expr1}$  вправо на  $\text{expr2}$  бит  
 $\text{var} \gg \text{expr}$  Побитовый сдвиг вправо значения переменной  $\text{var}$  на  $\text{expr}$  бит  
 $\text{expr1} \wedge \text{expr2}$  Исключающее OR выражений  $\text{expr1}$  и  $\text{expr2}$   
 $\text{var} \wedge \text{expr}$  Присваивает переменной  $\text{var}$  побитовое XOR  $\text{var}$  и  $\text{expr}$   
 $\text{expr1} | \text{expr2}$  Побитовое OR выражений  $\text{expr1}$  и  $\text{expr2}$   
 $\text{var} |= \text{expr}$  Присваивает переменной  $\text{var}$  результат операции XOR  $\text{var}$  и  $\text{expr}$   
 $\text{expr1} || \text{expr2}$  1, если или  $\text{expr1}$  или  $\text{expr2}$  являются ненулевыми значениями; иначе 0  
 $\sim \text{expr}$  Побитовое дополнение до  $\text{expr}$

- 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. Такие символы, как *' < > \* ? | " &*, являются метасимволами и имеют для командного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме *\$, ', , "*. Например, – *echo \** выведет на экран символ, – *echo ab'|'cd* выведет на экран строку *ab|\*cd*.

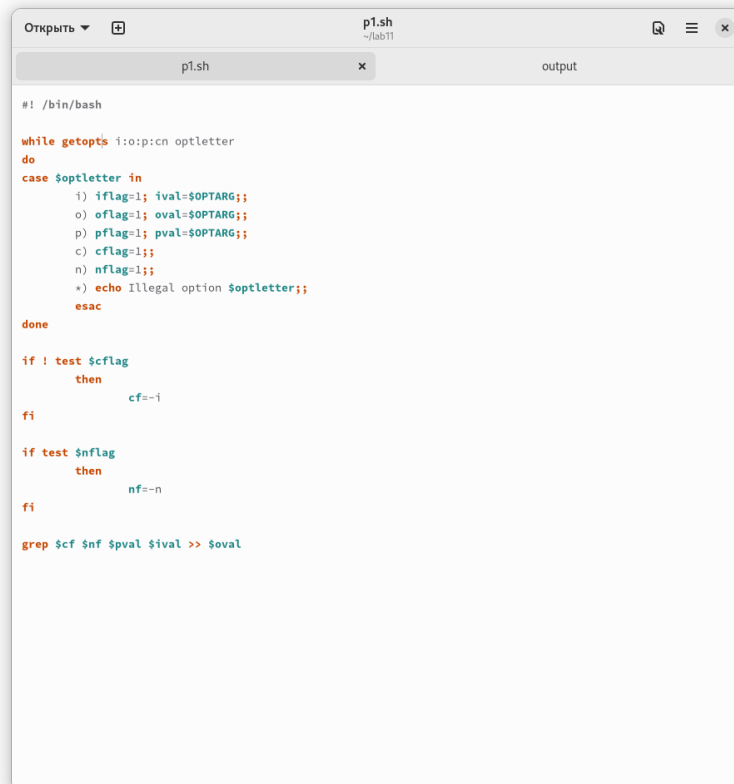
## 4 Выполнение лабораторной работы

Приступаю к первому заданию и создаю все нужные файлы (рис. 4.1). Далее пишу код для первого задания (рис. 4.2), который ищет строчки по шаблону и нужные потом выводит в другой файл. Строчки нумеруются (рис. 4.3).



```
iavolgin@fedora:~/lab11
[iavolgin@fedora ~]$ mkdir lab11
[iavolgin@fedora ~]$ cd lab11
[iavolgin@fedora lab11]$ touch input
[iavolgin@fedora lab11]$ touch output
[iavolgin@fedora lab11]$ touch p1.sh
[iavolgin@fedora lab11]$ chmod u+x p1.sh
[iavolgin@fedora lab11]$ p1.sh
bash: p1.sh: команда не найдена...
[iavolgin@fedora lab11]$ p1.sh -p улит -i input.txt -o output.txt -c -n
bash: p1.sh: команда не найдена...
[iavolgin@fedora lab11]$ bash p1.sh -p улит -i input.txt -o output.txt -c -n
p1.sh: строка 1: !#: команда не найдена
p1.sh: строка 3: get0ops: команда не найдена
p1.sh: строка 25: $oval: неоднозначное перенаправление
[iavolgin@fedora lab11]$ bash p1.sh -p улит -i input.txt -o output.txt -c -n
p1.sh: строка 3: getops: команда не найдена
p1.sh: строка 25: $oval: неоднозначное перенаправление
[iavolgin@fedora lab11]$ bash p1.sh -p улит -i input.txt -o output.txt -c -n
p1.sh: строка 3: getops: команда не найдена
p1.sh: строка 25: $oval: неоднозначное перенаправление
[iavolgin@fedora lab11]$ bash p1.sh -p улит -i input.txt -o output.txt -c -n
grep: input.txt: Нет такого файла или каталога
[iavolgin@fedora lab11]$ bash p1.sh -p улит -i input -o output -c -n
[iavolgin@fedora lab11]$
```

Рис. 4.1: Создаю нужные файлы



```
#!/bin/bash

while getopts i:op: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
```

Рис. 4.2: Код программы

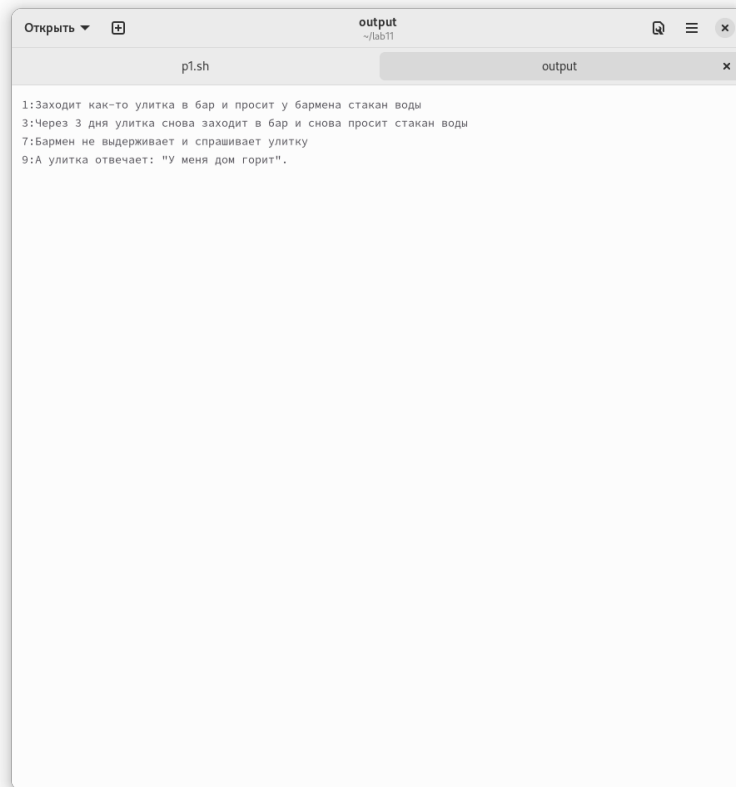
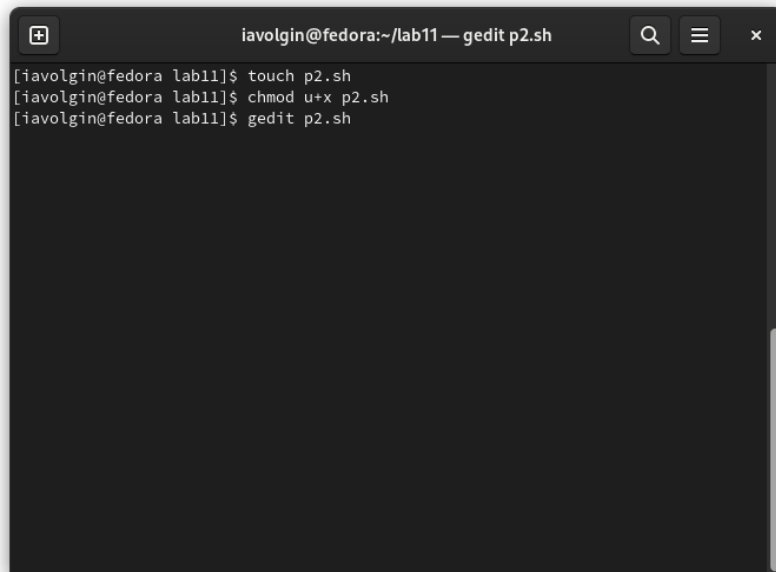


Рис. 4.3: Выполнение программы

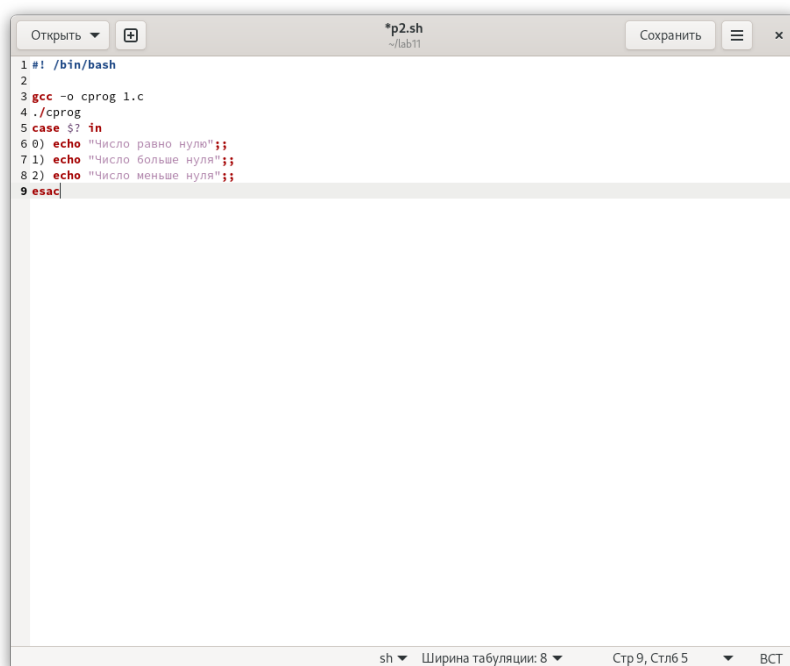
Далее второе задание. Так же создаю нужные файлы и даю права доступа (рис. 4.4). Пишу на си программу (рис. 4.5) (рис. 4.6), которая выводит, является ли заданное число больше, меньше или равно нулю (рис. 4.7).



A terminal window titled "iavolgin@fedora:~/lab11 — gedit p2.sh". The terminal shows the following commands and their outputs:

```
[iavolgin@fedora lab11]$ touch p2.sh
[iavolgin@fedora lab11]$ chmod u+x p2.sh
[iavolgin@fedora lab11]$ gedit p2.sh
```

Рис. 4.4: Создаю нужные файлы

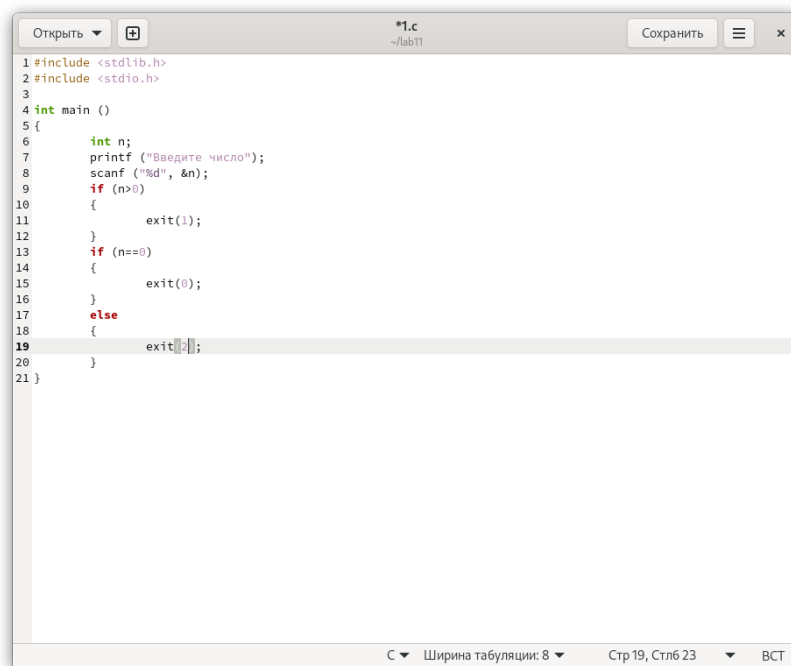


An editor window titled "\*p2.sh" with the file path "~/lab11". The code inside is as follows:

```
1 #!/bin/bash
2
3 gcc -o cprog 1.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
```

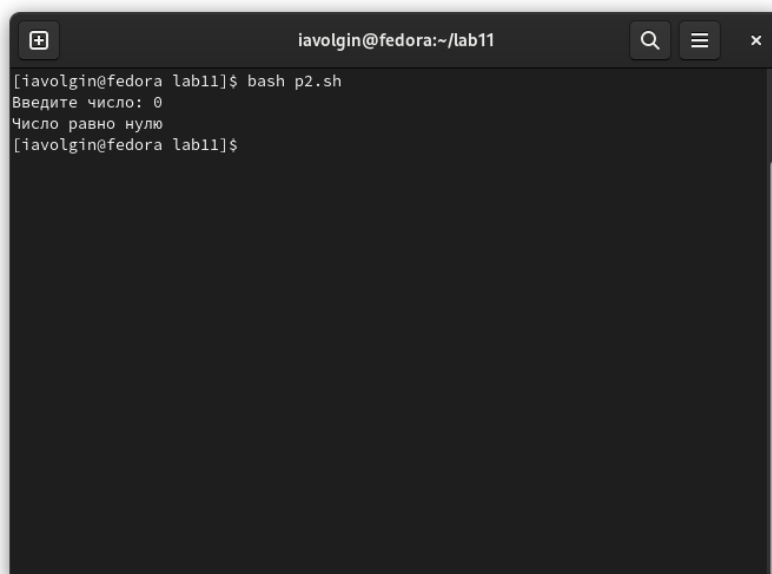
The status bar at the bottom indicates "sh", "Ширина табуляции: 8", "Стр 9, Стлб 5", and "ВСТ".

Рис. 4.5: Код программы



```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main ()
5 {
6     int n;
7     printf ("Введите число");
8     scanf ("%d", &n);
9     if (n>0)
10    {
11        exit(1);
12    }
13    if (n==0)
14    {
15        exit(0);
16    }
17    else
18    {
19        exit(1);
20    }
21 }
```

Рис. 4.6: Код программы

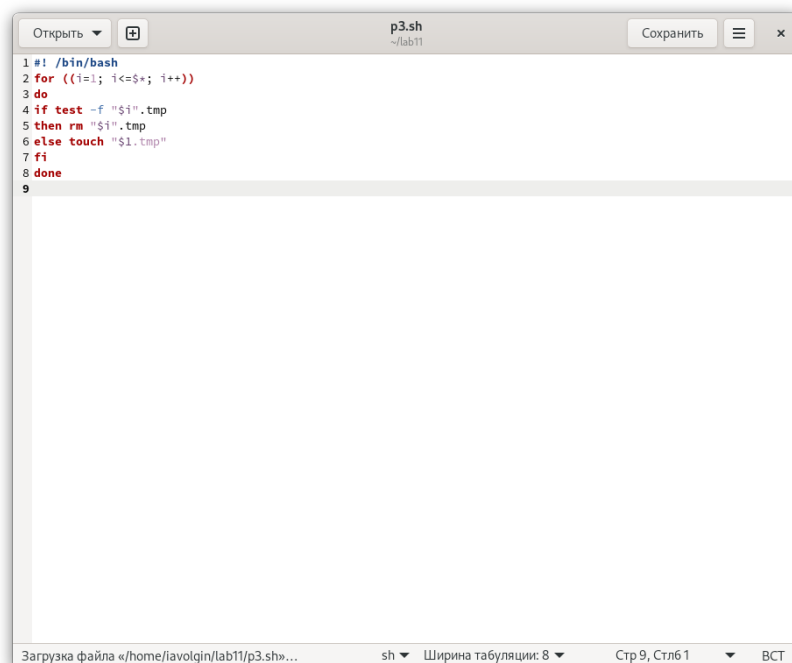


```
iavolgin@fedora:~/lab11
[iavolgin@fedora lab11]$ bash p2.sh
Введите число: 0
Число равно нулю
[iavolgin@fedora lab11]$
```

Рис. 4.7: Выполнение программы

Третье задание. Также создаю нужные файлы, не стал скрины делать, потому

что все идентично двум предыдущим заданиям. Пишу код, который позволяет создать указанное число файлов (например, если указать число 3, то создадутся файлы 1.tmp, 2.tmp, 3.tmp, и так любое количество) (рис. 4.8). Если такие файлы уже существуют, то они просто удаляются (рис. 4.9).

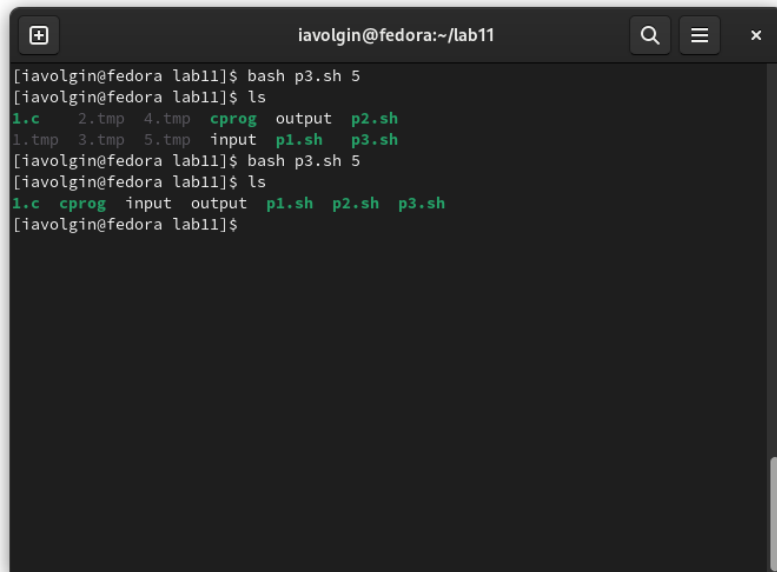
A screenshot of a text editor window titled 'p3.sh' with a path of '~/lab11'. The window contains a shell script with the following code:

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

The script is a bash script that uses a for loop to iterate from 1 to 3. For each iteration, it checks if a file named '\$i.tmp' exists using the 'test' command. If it exists, it removes the file with 'rm'. If it does not exist, it creates the file with 'touch'. The script ends with a 'done' statement for the loop. The editor interface includes a top bar with 'Открыть' (Open) and 'Сохранить' (Save) buttons, and a bottom status bar showing 'sh', 'Ширина табуляций: 8', 'Стр 9, Стлб 1', and 'ВСТ'.

Рис. 4.8: Код программы

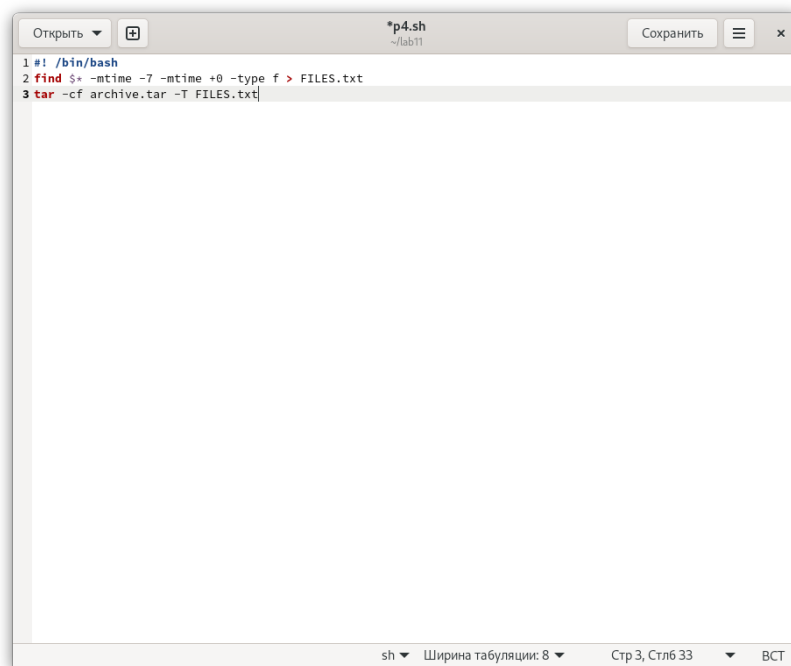




```
iavolgin@fedora:~/lab11
[iavolgin@fedora lab11]$ bash p3.sh 5
[iavolgin@fedora lab11]$ ls
1.c      2.tmp    4.tmp    cprog    output   p2.sh
1.tmp    3.tmp    5.tmp    input    p1.sh    p3.sh
[iavolgin@fedora lab11]$ bash p3.sh 5
[iavolgin@fedora lab11]$ ls
1.c      cprog    input    output   p1.sh    p2.sh    p3.sh
[iavolgin@fedora lab11]$
```

Рис. 4.9: Выполнение программы

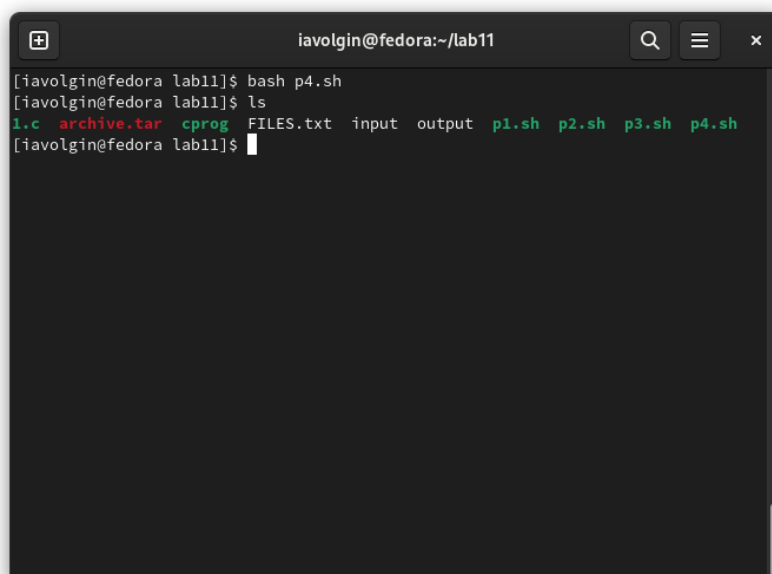
Четвертое задание. Так же не стал делать скринов создания всех нужных файлов. Программа должна выполнять архивирование всех файлов в указанной директории (рис. 4.10). Вот так это выглядит (рис. 4.11).



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

The screenshot shows a code editor window with a title bar containing "Открыть", a plus icon, "\*p4.sh", "~ /lab11", "Сохранить", a hamburger menu icon, and a close icon. The code is a shell script with three lines. The status bar at the bottom indicates "sh", "Ширина табуляции: 8", "Стр 3, Стлб 33", and "ВСТ".

Рис. 4.10: Код программы



```
iavolgin@fedora:~/lab11
[iavolgin@fedora lab11]$ bash p4.sh
[iavolgin@fedora lab11]$ ls
l.c  archive.tar  cprog  FILES.txt  input  output  p1.sh  p2.sh  p3.sh  p4.sh
[iavolgin@fedora lab11]$
```

The screenshot shows a terminal window with the title "iavolgin@fedora:~/lab11". It displays the execution of the script "p4.sh" and the resulting files in the directory: "l.c", "archive.tar", "cprog", "FILES.txt", "input", "output", "p1.sh", "p2.sh", "p3.sh", and "p4.sh".

Рис. 4.11: Выполнение программы

## 5 Выводы

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