

Отчет о выполнении лабораторной работы

Лабораторная работа №13

Филиппьева Ксения Дмитриевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	22
5	Ответы на вопросы	23

Список иллюстраций

3.1	создание файла	7
3.2	ввод кода	8
3.3	создание файла	8
3.4	текст в файле	9
3.5	вывод результата	9
3.6	вывод результата	10
3.7	вывод результата	10
3.8	исправленный текст	11
3.9	вывод результата	12
3.10	вывод результата	12
3.11	вывод результата	13
3.12	создание файла	13
3.13	код программы	14
3.14	создание файла	14
3.15	код программы	15
3.16	вывод результата	16
3.17	создание файла	16
3.18	вставка кода	17
3.19	вывод результата	18
3.20	вывод результата	19
3.21	создание файла	19
3.22	код программы	20
3.23	вывод результата	20
3.24	вывод результата	21

Список таблиц

1 Цель работы

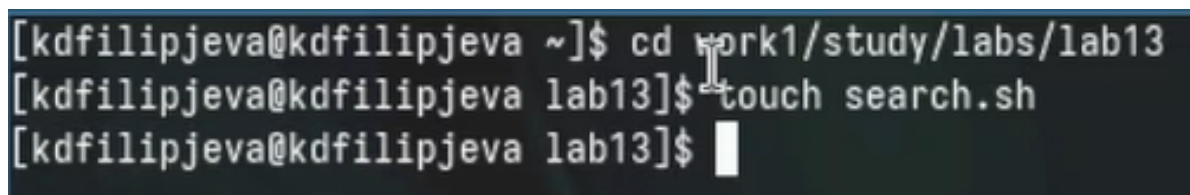
Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами: `-i`inputfile — прочитать данные из указанного файла; `-o`outputfile — 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`.
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 Выполнение лабораторной работы

Создадим файл для первого задания (рис. 3.1).

A terminal window with a dark background and light gray text. The prompt is [kdfilipjeva@kdfilipjeva ~]\$. The first command is cd work1/study/labs/lab13. The second prompt is [kdfilipjeva@kdfilipjeva lab13]\$. The second command is touch search.sh. The third prompt is [kdfilipjeva@kdfilipjeva lab13]\$.

```
[kdfilipjeva@kdfilipjeva ~]$ cd work1/study/labs/lab13
[kdfilipjeva@kdfilipjeva lab13]$ touch search.sh
[kdfilipjeva@kdfilipjeva lab13]$
```

Рис. 3.1: создание файла

Введем в него код для первого задания (рис. 3.2).

```
foot
GNU nano 7.2 search.sh
#!/bin/bash

# Инициализация переменных
input_file=""
output_file=""
pattern=""
case_sensitive=0
show_line_numbers=0

# Функция для обработки параметров командной строки
parse_arguments() {
    while getopts "i:o:p:Cn" opt; do
        case $opt in
            i) input_file="$OPTARG" ;;
            o) output_file="$OPTARG" ;;
            p) pattern="$OPTARG" ;;
            C) case_sensitive=1 ;;
            n) show_line_numbers=1 ;;
            *) echo "Использование: $0 [-i inputfile] [-o outp>
esac
```

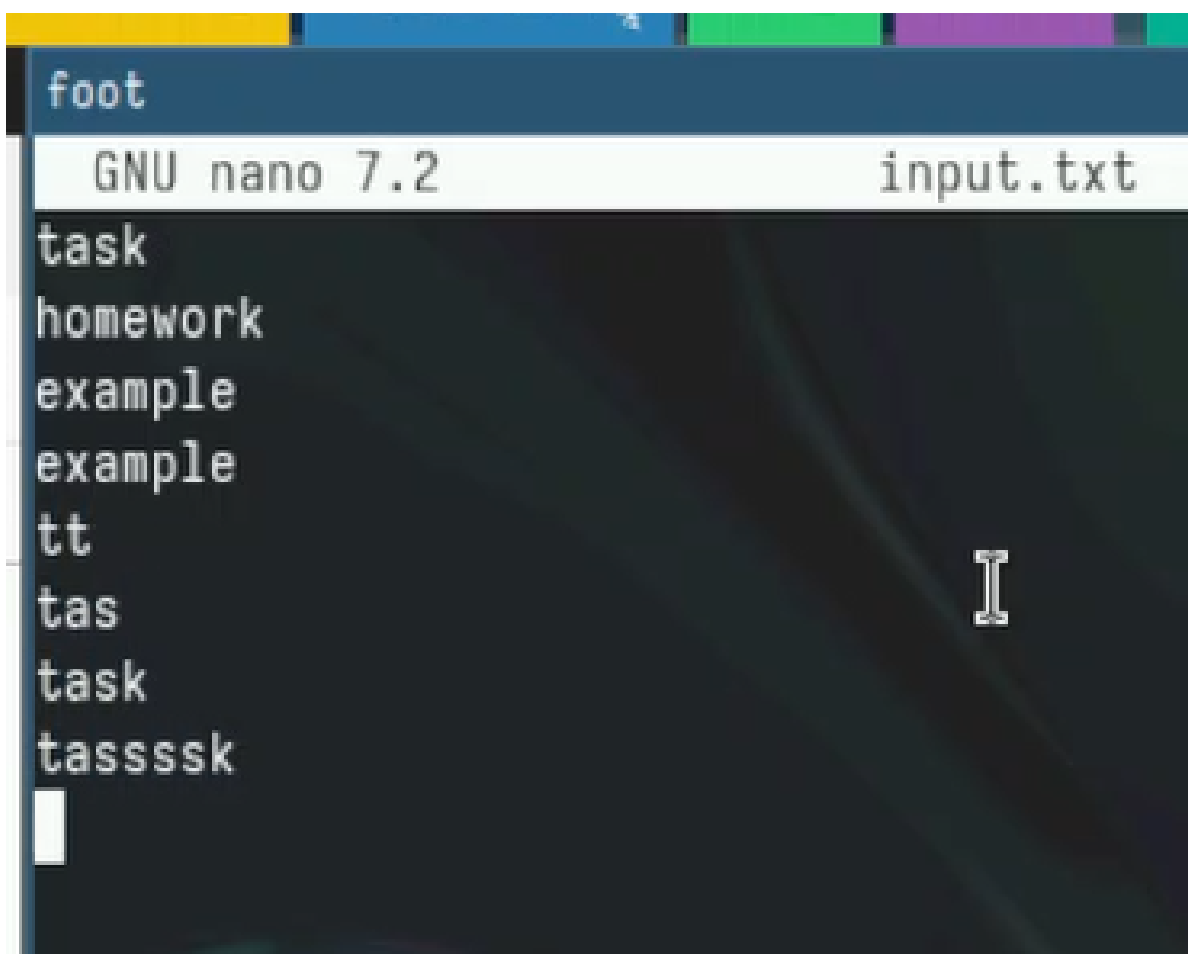
Рис. 3.2: ввод кода

Создадим файл в который будем вводить текст, с которым будет работать программа (рис. 3.3).

```
[kdfilipjeva@kdfilipjeva lab13]$ touch input.txt
[kdfilipjeva@kdfilipjeva lab13]$
```

Рис. 3.3: создание файла

Текст для работы программы (рис. 3.4).

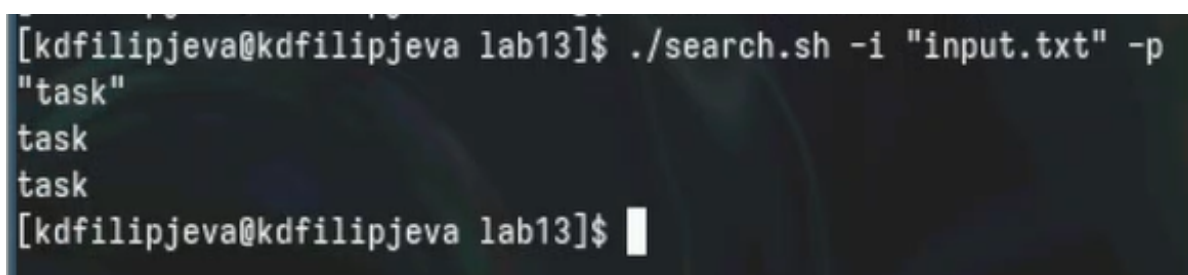


```
foot
GNU nano 7.2 input.txt
task
homework
example
example
tt
tas
task
tassssk

```

Рис. 3.4: текст в файле

Вывод найденного текста по установленному шаблону в командную строку (рис. 3.5).



```
[kdfilipjeva@kdfilipjeva lab13]$ ./search.sh -i "input.txt" -p "task"
task
task
[kdfilipjeva@kdfilipjeva lab13]$
```

Рис. 3.5: вывод результата

Выведем найденный текст в отдельный файл (рис. 3.6).

```
[kdfilipjeva@kdfilipjeva lab13]$ ./search.sh -i "input.txt" -p  
"task" -o "output.txt"
```

Рис. 3.6: вывод результата

Выведенный текст в отдельном файле (рис. 3.7).

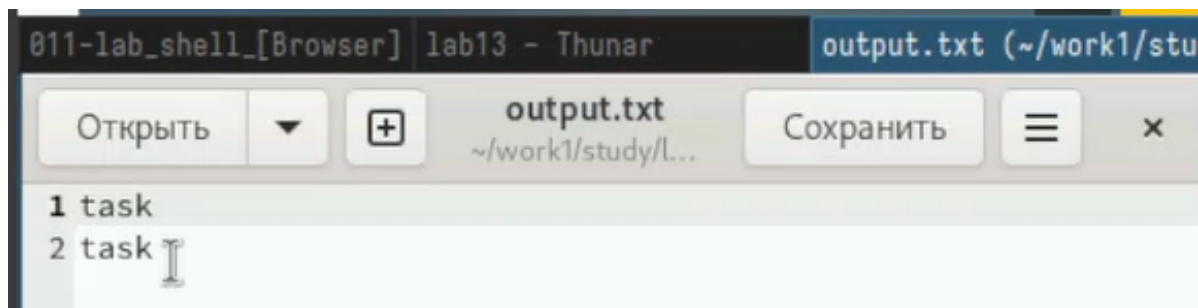


Рис. 3.7: вывод результата

Отредактируем текст для работы программы (рис. 3.8).

```
foot
GNU nano 7.2
task
homework
example
example
tt
tas
task
tassssk

Task
```

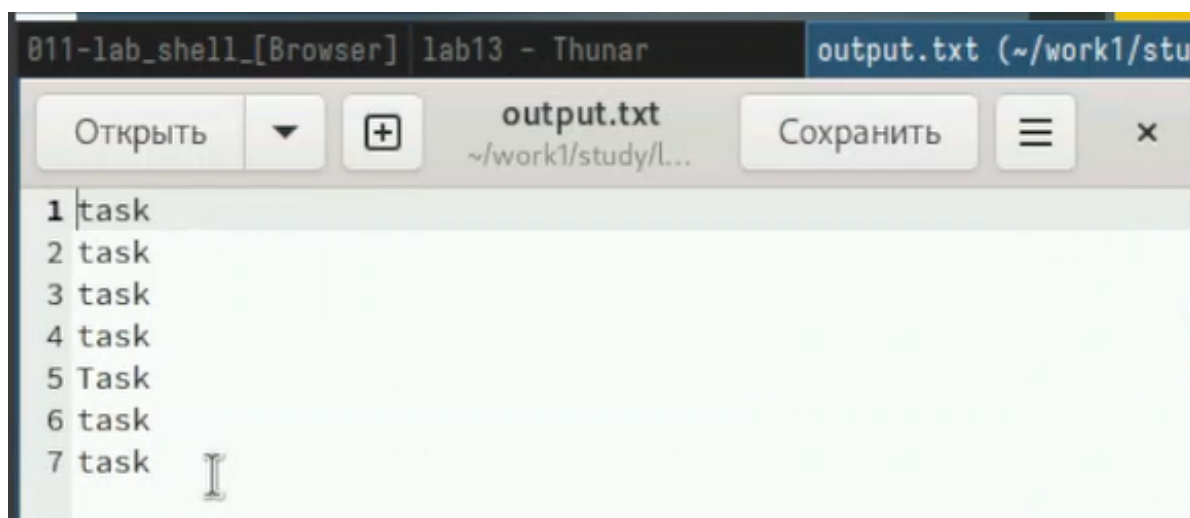
Рис. 3.8: исправленный текст

Выведем текст в файл с учетом регистра (рис. 3.9).

```
[kdfilipjeva@kdfilipjeva lab13]$ ./search.sh -i "input.txt" -p  
"task" -C -o "output.txt"
```

Рис. 3.9: вывод результата

Вывод текста с учетом регистра(видно, что вывело только с маленькой буквы, а заглавную не тронуло) (рис. 3.10).



011-lab_shell_[Browser] lab13 - Thunar output.txt (~/work1/stu

Открыть output.txt Сохранить

1 task
2 task
3 task
4 task
5 Task
6 task
7 task

Рис. 3.10: вывод результата

Выведем текст с учетом регистра и нумерацией строк, из которых было взято слово (рис. ??).

![вывод результата(image/1311.png){#fig:11 width=100%}]

Вывод текста с нумерацией (рис. 3.11).



Рис. 3.11: вывод результата

Создадим файл для второго задания и выдадим права на выполнение (рис. 3.12).

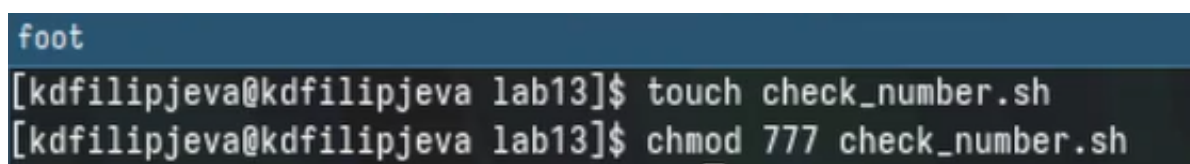
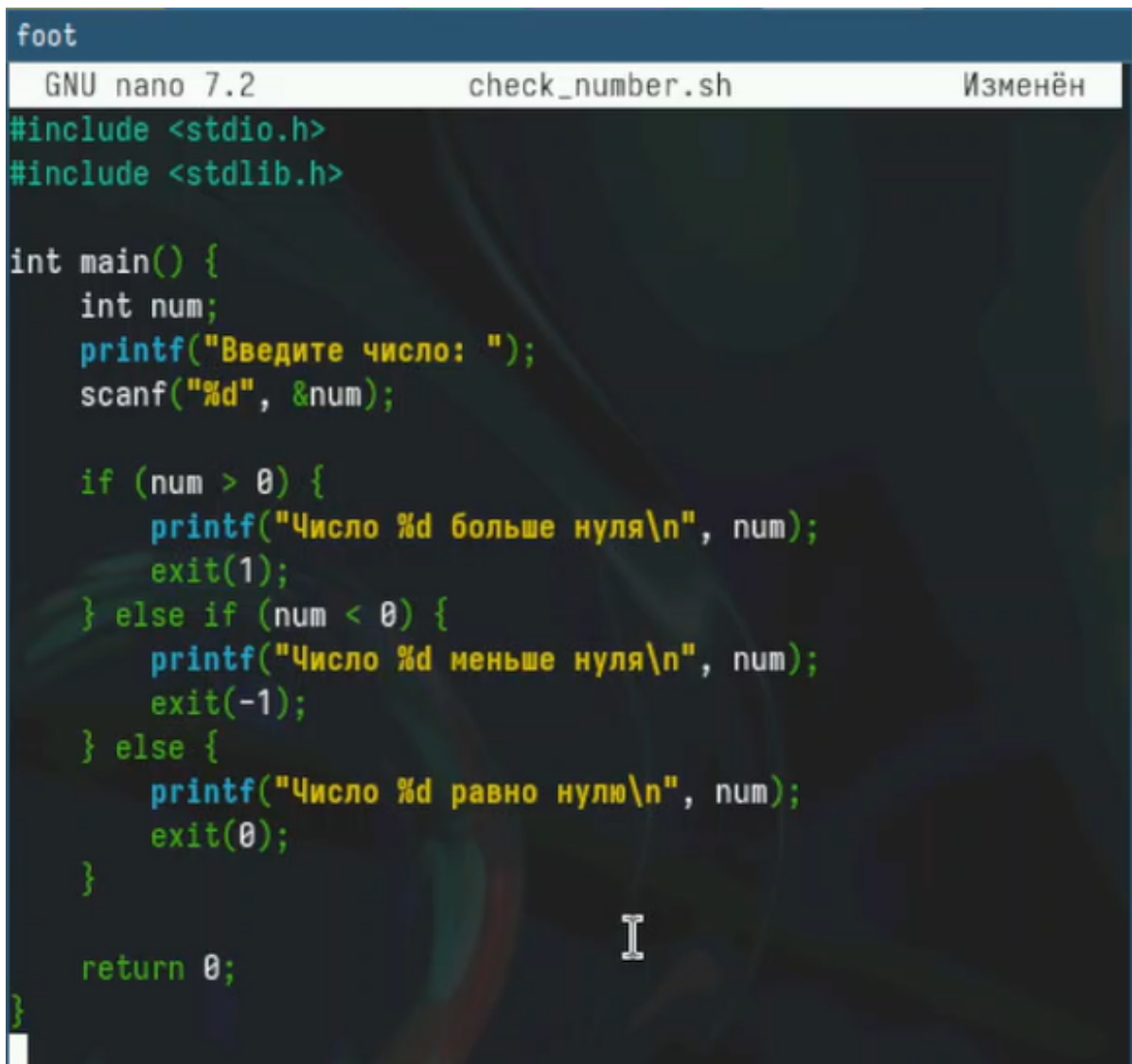


Рис. 3.12: создание файла

Вставим код программы, который ответственен за определение числа (рис. 3.13).



```
foot
GNU nano 7.2      check_number.sh      Изменён
#include <stdio.h>
#include <stdlib.h>

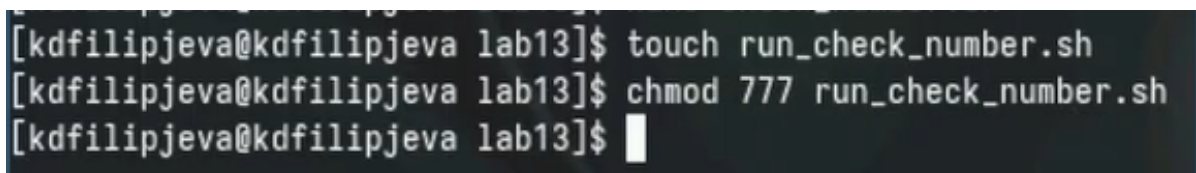
int main() {
    int num;
    printf("Введите число: ");
    scanf("%d", &num);

    if (num > 0) {
        printf("Число %d больше нуля\n", num);
        exit(1);
    } else if (num < 0) {
        printf("Число %d меньше нуля\n", num);
        exit(-1);
    } else {
        printf("Число %d равно нулю\n", num);
        exit(0);
    }

    return 0;
}
```

Рис. 3.13: код программы

Создадим файл для второго задания, который будет “общаться с пользователем” (рис. 3.14).



```
[kdfilipjeva@kdfilipjeva lab13]$ touch run_check_number.sh
[kdfilipjeva@kdfilipjeva lab13]$ chmod 777 run_check_number.sh
[kdfilipjeva@kdfilipjeva lab13]$
```

Рис. 3.14: создание файла

Вставим в него код программы (рис. 3.15).

```
foot
GNU nano 7.2      run_check_number.sh      Изменён
#!/bin/bash

# Вызов программы на С
./check_number

# Анализ кода завершения
case $? in
    0)
        echo "Число равно нулю"
        ;;
    1)
        echo "Число больше нуля"
        ;;
    -1)
        echo "Число меньше нуля"
        ;;
    *)
        echo "Ошибка при выполнении программы"
        ;;
esac
```

Рис. 3.15: код программы

Скомпилируем наш код на языке Си и проверим работоспособность (рис. 3.16).

```
[kdfilipjeva@kdfilipjeva lab13]$ gcc -o check_number check_n  
umber.c  
[kdfilipjeva@kdfilipjeva lab13]$ ./run_check_number.sh  
Введите число: 10  
Число 10 больше нуля  
Число больше нуля  
[kdfilipjeva@kdfilipjeva lab13]$ ./run_check_number.sh  
Введите число: 0  
Число 0 равно нулю  
Число равно нулю  
[kdfilipjeva@kdfilipjeva lab13]$ ./run_check_number.sh  
Введите число: -10  
Число -10 меньше нуля  
Ошибка при выполнении программы  
[kdfilipjeva@kdfilipjeva lab13]$
```

Рис. 3.16: вывод результата

Создадим файл для третьего задания и выдадим ему права на выполнение (рис. 3.17).

```
foot  
[kdfilipjeva@kdfilipjeva lab13]$ touch file_manager.sh  
[kdfilipjeva@kdfilipjeva lab13]$ chmod 777 file_manager.sh  
[kdfilipjeva@kdfilipjeva lab13]$ nano file_manager.sh
```

Рис. 3.17: создание файла

Вставим в него необходимый код для выполнения задания (рис. 3.18).


```
foot
GNU nano 7.2      file_manager.sh
#!/bin/bash

# Функция для создания файлов
create_files() {
    local num_files=$1
    for i in $(seq 1 $num_files); do
        touch "$i.tmp"
    done
    echo "Создано $num_files файлов"
}

# Функция для удаления файлов
delete_files() {
    local num_files=$1
    for i in $(seq 1 $num_files); do
        rm -f "$i.tmp"
    done
    echo "Удалено $num_files файлов"
```

Рис. 3.18: вставка кода

Работоспособность кода (рис. 3.19).

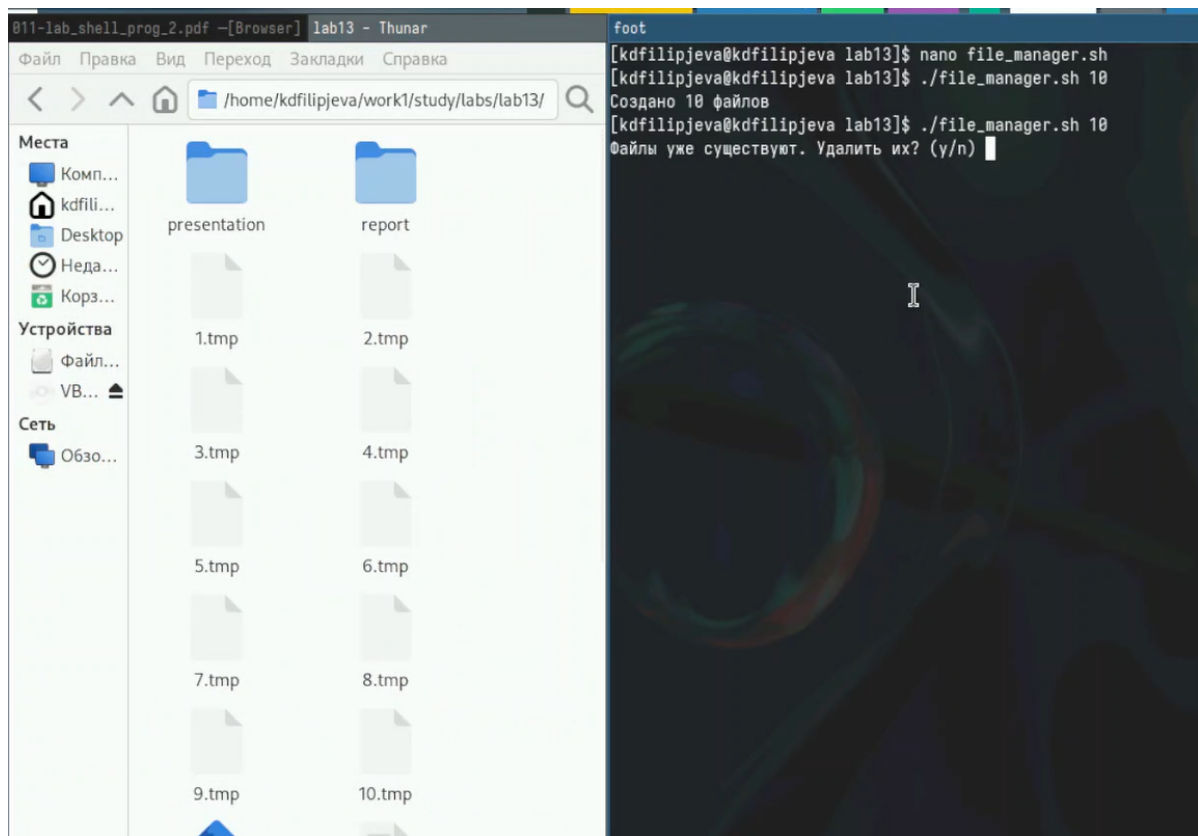


Рис. 3.19: вывод результата

Работоспособность кода в обратную сторону (рис. 3.20).

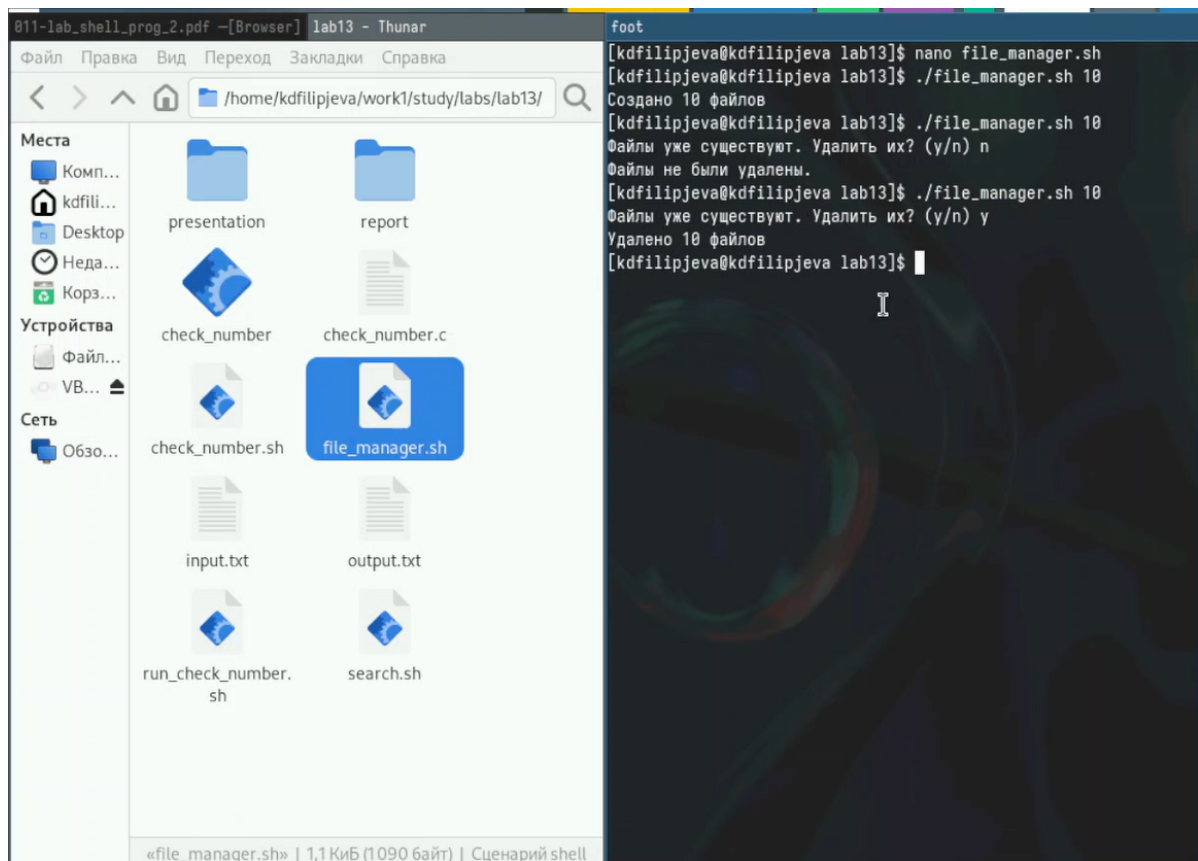


Рис. 3.20: вывод результата

Создадим файл для четвертого задания и выдадим ему права на выполнение (рис. 3.21).

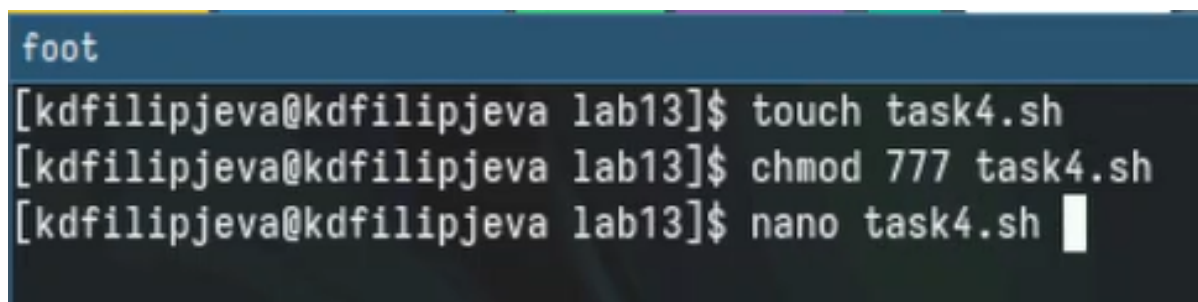


Рис. 3.21: создание файла

Вставим в него необходимый код программы (рис. 3.22).

```

foot
GNU nano 7.2 task4.sh

# Переход в директорию
pushd "$dir" > /dev/null

# Создание архива
tar -czf "$archive_name" .
echo "Файлы в директории '$dir' упакованы в архив '$archive_name'"

# Возврат в предыдущую директорию
popd > /dev/null
}

# Функция для упаковки только недавно измененных файлов
pack_recent_files() {

```

Рис. 3.22: код программы

Работоспособность кода (рис. 3.23).

```

[kdfilipjeva@kdfilipjeva lab13]$ nano task4.sh
[kdfilipjeva@kdfilipjeva lab13]$ ./task4.sh ~/work1/study/labs/lab13/report/ backup.tar.gz 7
tar: .: файл изменился во время чтения
Файлы в директории '/home/kdfilipjeva/work1/study/labs/lab13/report/' упакованы в архив 'backup.tar.gz'.
Файлы в директории '/home/kdfilipjeva/work1/study/labs/lab13/report/', измененные менее 7 дней назад, упакованы в архив 'backup.tar_recent_20240504223949.tar.gz'.
[kdfilipjeva@kdfilipjeva lab13]$

```

Рис. 3.23: вывод результата

Созданные 2 архива: всей папки и только файлов, которые были изменены менее чем неделю назад (рис. 3.24).

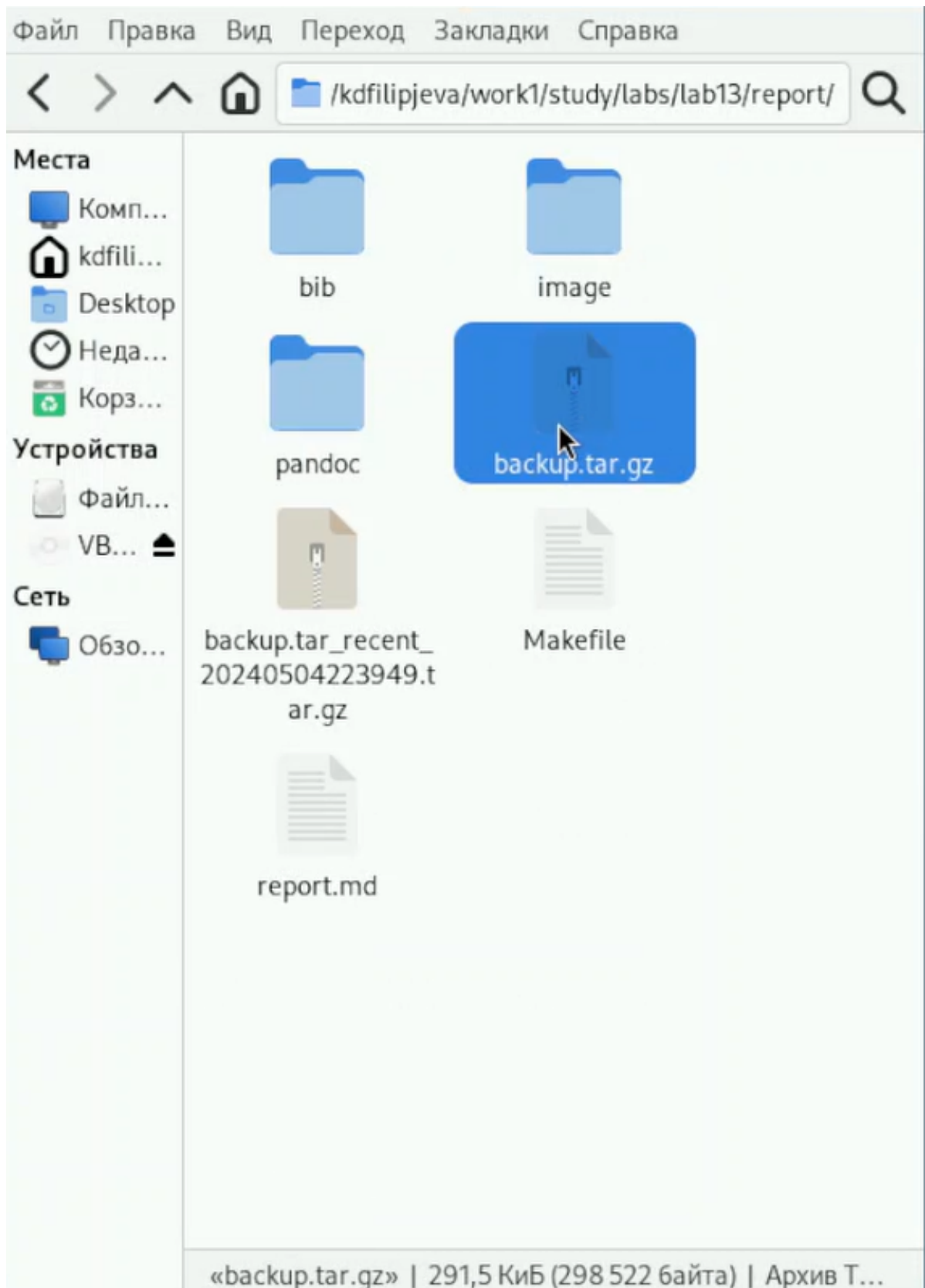


Рис. 3.24: вывод результата

4 Выводы

Мы получили новые и отработали уже имеющиеся навыки программирования в оболочке ОС Linux.

5 Ответы на вопросы

1. Команда `getopts` используется для разбора параметров командной строки в скриптах Bash. Она позволяет легко обрабатывать флаги и аргументы, переданные пользователем при запуске скрипта.
2. Метасимволы, такие как `*`, `?`, `[]`, используются в командной строке для создания шаблонов имен файлов. Они позволяют быстро и гибко генерировать списки файлов, соответствующих определенным критериям.
3. Основные операторы управления действиями в Bash:
 - `if-then-else-fi` - для условного выполнения команд
 - `case-esac` - для многовариантного выбора
 - `for-do-done` - для итерации по списку значений
 - `while-do-done` - для выполнения команд, пока условие истинно
 - `until-do-done` - для выполнения команд, пока условие ложно
4. Для прерывания цикла в Bash используются:
 - `break` - для выхода из текущего цикла
 - `continue` - для перехода к следующей итерации цикла
5. Команды `false` и `true` возвращают соответственно ложное (1) и истинное (0) значение, которое можно использовать в управляющих конструкциях.
6. Строка `if test -f mans/i.$s` проверяет, существует ли файл с именем, сформированным из переменных `$s` и `$i`. Если файл существует (`-f`), то выполняются дальнейшие действия.
7. Конструкция `while` выполняет команды, пока условие истинно, а `until` выполняет команды, пока условие ложно. Таким образом, `until` можно рассматривать как инвертированный `while`.