**Филиал федерального государственного бюджетного образовательного учреждения**

**высшего образования**

**«Национальный исследовательский университет «МЭИ»**

**в г. Смоленске**

кафедра ИТЭУ

Отчеты по лабораторным работам

по дисциплине:

«Разработка и стандартизация программных средств и информационных технологий»

Группа: БЭИС-19

Студент: Большунов А. В.

Преподаватель: Лобанева Е.И.

Смоленск 2021

СОДЕРЖАНИЕ

[1 Лабораторная работа № 1. Основы языка и знакомство с основными элементами управления C# 3](#_Toc65966109)

[2 Лабораторная работа № 2. Основы работы с системой контроля версий git. 9](#_Toc65966110)

[Список использованных источников 27](#_Toc65966111)

# Лабораторная работа № 1. Основы языка и знакомство с основными элементами управления C#

Задание 1. Даны катет и гипотенуза прямоугольного треугольника. Найдите площадь треугольника. Выведите результат на экран.

Математическое описание решения. Площадь треугольника можно вычислить как половину произведения катетов

Схема алгоритма. В соответствии с заданием разработан алгоритм, схема которого отражена блок-схемой на рисунке 1.1

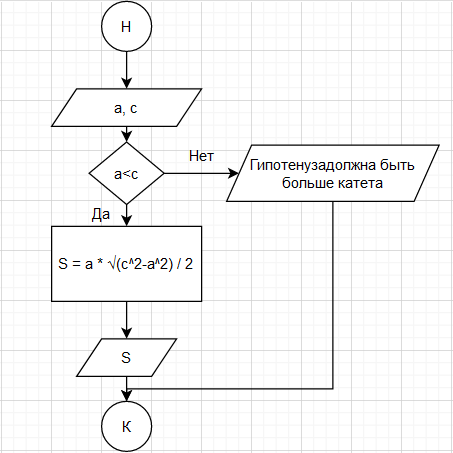


Рисунок 1.1 – Блок-схема алгоритма к лабораторной работе № 1

Разработка программы. Разработка программы проводилась на языке высокого уровня C# [1–2]. Список идентификаторов для программы, разрабатываемой в соответствии с алгоритмом, отраженном на рисунке 1.1, представлен в таблице 1.1.

Таблица 1.1 – Список идентификаторов для задания лаб. раб. № 1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование переменной и ее смысл | Обозначение в алгоритме | Обозначение в программе |
| 1 | Катет | a | a |
| 2 | Гипотенуза | c | c |
| 3 | Площадь | S | S |

Текст программы представлен на листинге 1.1

Тестирование программы. Тестирование является процессом прогона программы на различных входных данных с целью обнаружения в ней ошибок. В стандарте ISO/IEC TR 19759:2005 дано такое определение: тестирование прогр**а**ммного обесп**е**ч**е**ния — это процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.

В зависимости от доступа разработчика тестов к исходному коду тестируемой программы различают «тестирование (по стратегии) белого ящика» и «тестирование (по стратегии) чёрного ящика».

При тестировании белого ящика (также говорят — прозрачного ящика), разработчик теста имеет доступ к исходному коду программ и может писать код, который связан с библиотеками тестируемого программного обеспечения. Это типично для компонентного тестирования, при котором тестируются только отдельные части системы. Оно обеспечивает то, что компоненты конструкции работоспособны и устойчивы, до определённой степени. При тестировании белого ящика используются метрики покрытия кода или мутационное тестирование.

При тестировании чёрного ящика тестировщик имеет доступ к программе только через те же интерфейсы, что и заказчик или пользователь, либо, через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования, а к самому коду тестировщик доступа не имеет.

Листинг 1.1– Текст программы на языке C# для задания к лаб. раб. №1

using System;

using System.Collections.Generic;

using System.Text;

namespace LB1

{

class Program

{

static void Main(string[] args)

{

Console.Title = "Площадь треугольника по катету и гипотенузе";

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

string Ans;

do

{

Console.Clear();

try

{

Console.Write("Введите катет: ");

double a = Double.Parse(Console.ReadLine());

Console.Write("Введите гипотенузу: ");

double c = Double.Parse(Console.ReadLine());

if (a >= c)

Console.Write("Error: Гипотенуза должна быть больше катета! ");

else

{

double S = a \* Math.Pow(Math.Pow(c, 2) - Math.Pow(a, 2), 0.5) / 2;

Console.Write("Площадь треугольника = " + S + "\n");

}

}

catch (Exception)

{

Console.Write("Введенные значеня некорректны! \n");

}

Console.Write("Повторим? ");

Ans = Console.ReadLine();

}

while (Ans == "Y" || Ans == "y");

}

}

}

В данном случае тестирование проводилось по методу «белого ящика», так как структура программы известна и можно было обеспечивать прохождение алгоритма по всем ветвям вычислений. Результаты тестирования отражены в таблице 1.2. Вид окна с результатами работы программы на некотором наборе исходных данных показан на рисунке 1.2.

Проведенное тестирование показало корректную работу программы на предложенных тестовых наборах входных данных, а выявленные ошибки были исправлены путем введения в программу дополнительных проверок.

Таблица 1.2 – Результаты тестирования программы задания № 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | Тестовые воздействия | | Результат программы | |
| Ожидаемый  результат | Результат, выданный программой |
| 1 | Ввод числовых  данных | Вводимые данные являются малыми числами из диапазона 10-13  до 10-37 | Корректное вычисление функции | Функция вычисляется корректно |
| Вводимые данные являются большими числами из диапазона 10-13  до 10-37 | Корректное вычисление функции | Функция вычисляется корректно |
| 2 | Некорректный ввод данных | Вводимые данные являются буквами | Выдача сообщения об ошибке | Выдача сообщения об ошибке |
| Число *а* не введено | Программа продолжает запрашивать ввод данных | Программа продолжает запрашивать ввод данных |
| Вместо разделительной точки введена запятая | Корректное вычисление функции | Выдача сообщения об ошибке |

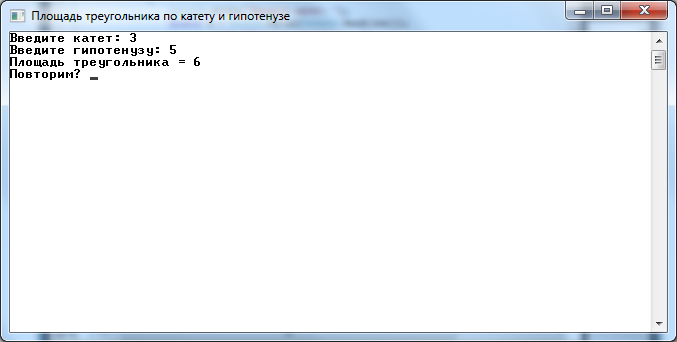


Рисунок 1.2 – Результаты работы программы

Задание 2. Выведите на экран N значений функции G(x)=sin2(π\*x)\*cos (π\*x) на интервале a<x<=b.

Математическое описание решения. Для данной задачи математическое описание присутствует в самом задании.

Схема алгоритма. В соответствии с заданием разработан алгоритм, схема которого отражена блок-схемой на рисунке 1.3

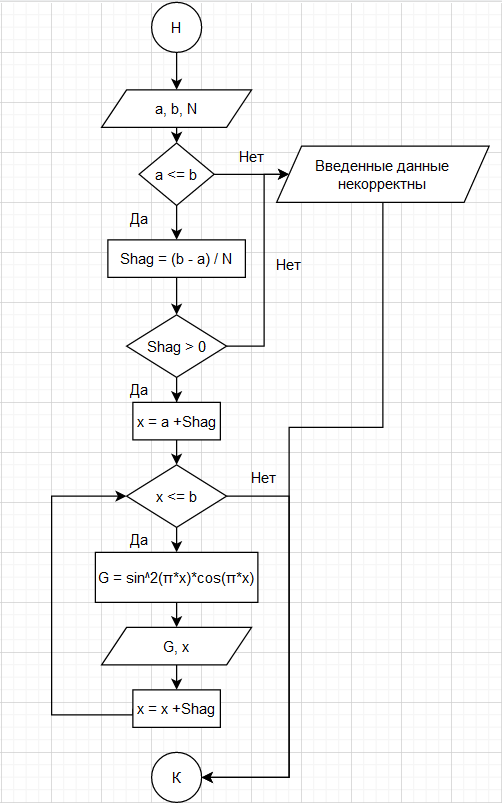


Рисунок 1.3 – Блок-схема алгоритма к лабораторной работе № 1

Разработка программы. Разработка программы проводилась на языке высокого уровня C# [1–2]. Список идентификаторов для программы, разрабатываемой в соответствии с алгоритмом, отраженном на рисунке 1.3, представлен в таблице 1.3.

Таблица 1.3 – Список идентификаторов для задания лаб. раб. № 1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование переменной и ее смысл | Обозначение в алгоритме | Обозначение в программе |
| 1 | Начало интервала | a | a |
| 2 | Переменная | x | x |
| 3 | Конец интервала | b | b |
| 4 | Кол-во значений функции | N | N |
| 5 | Интервал промежутка | Shag | Shag |
| 6 | Значение функции | G | G |

Текст программы представлен на листинге 1.2

Тестирование программы. Тестирование является процессом прогона программы на различных входных данных с целью обнаружения в ней ошибок. В стандарте ISO/IEC TR 19759:2005 дано такое определение: тестирование прогр**а**ммного обесп**е**ч**е**ния — это процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.

В зависимости от доступа разработчика тестов к исходному коду тестируемой программы различают «тестирование (по стратегии) белого ящика» и «тестирование (по стратегии) чёрного ящика».

Листинг 1.2– Текст программы на языке C# для задания к лаб. раб. №1

namespace LB2

{

class Program

{

static void Main(string[] args)

{

Console.Title = "Площадь треугольника по катету и гипотенузе";

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

string Ans;

do

{

Console.Clear();

try

{

Console.Write("Введите интервал: a<x<=b \n");

Console.Write("a: ");

double a = Double.Parse(Console.ReadLine());

Console.Write("b: ");

double b = Double.Parse(Console.ReadLine());

Console.Write("Введите кол-во значений N: ");

int N = Int32.Parse(Console.ReadLine());

double Shag = (b - a) / N;

if (a <= b && Shag > 0)

{

double x = a + Shag;

double G;

while (x <= b)

{

G = Math.Pow(Math.Sin(Math.PI \* x), 2) \* Math.Cos(x);

Console.Write("x= " + x);

Console.Write("G= " + G + "\n");

x = x + Shag;

}

}

else

{

Console.Write("Error: Введенные данные некорректны");

}

}

catch (Exception)

{

Console.Write("Введенные значеня некорректны! \n");

}

Console.Write("\nПовторим? ");

Ans = Console.ReadLine();

}

while (Ans == "Y" || Ans == "y");

}

}

}

При тестировании белого ящика (также говорят — прозрачного ящика), разработчик теста имеет доступ к исходному коду программ и может писать код, который связан с библиотеками тестируемого программного обеспечения. Это типично для компонентного тестирования, при котором тестируются только отдельные части системы. Оно обеспечивает то, что компоненты конструкции работоспособны и устойчивы, до определённой степени. При тестировании белого ящика используются метрики покрытия кода или мутационное тестирование.

При тестировании чёрного ящика тестировщик имеет доступ к программе только через те же интерфейсы, что и заказчик или пользователь, либо, через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования, а к самому коду тестировщик доступа не имеет.

В данном случае тестирование проводилось по методу «белого ящика», так как структура программы известна и можно было обеспечивать прохождение алгоритма по всем ветвям вычислений. Результаты тестирования отражены в таблице 1.4. Вид окна с результатами работы программы на некотором наборе исходных данных показан на рисунке 1.4.

Проведенное тестирование показало корректную работу программы на предложенных тестовых наборах входных данных, а выявленные ошибки были исправлены путем введения в программу дополнительных проверок.

Таблица 1.4 – Результаты тестирования программы задания № 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | Тестовые воздействия | | Результат программы | |
| Ожидаемый  результат | Результат, выданный программой |
| 1 | Ввод числовых  данных | Вводимые данные являются малыми числами из диапазона 10-13  до 10-37 | Корректное вычисление функции | Функция вычисляется корректно |
| Вводимые данные являются большими числами из диапазона 10-13  до 10-37 | Корректное вычисление функции | Функция вычисляется корректно |
| 2 | Некорректный ввод данных | Вводимые данные являются буквами | Выдача сообщения об ошибке | Выдача сообщения об ошибке |
| Число *а* не введено | Программа продолжает запрашивать ввод данных | Программа продолжает запрашивать ввод данных |
| Вместо разделительной точки введена запятая | Корректное вычисление функции | Выдача сообщения об ошибке |

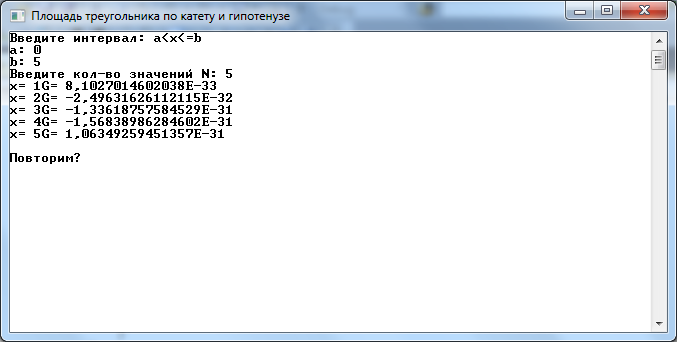


Рисунок 1.4 – Результаты работы программы

Выводы. В результате выполнения лабораторной работы №1 был разработаны алгоритмы решения задачи и реализующие их программы на языке C#. Проведенное тестирование показало корректную работу программы для предложенных тестовых воздействий.

# Лабораторная работа № 2. Основы работы с системой контроля версий git.

Задание 1. Регистрация на платформе github.com / gitlab.com.

Для первоначальной работы необходимо зарегистрироваться в системе контроля версий на сайте github.com. В качестве основных полей следует указать никнейм, почту и пароль для входа в личный кабинет, а также ввести капчу.

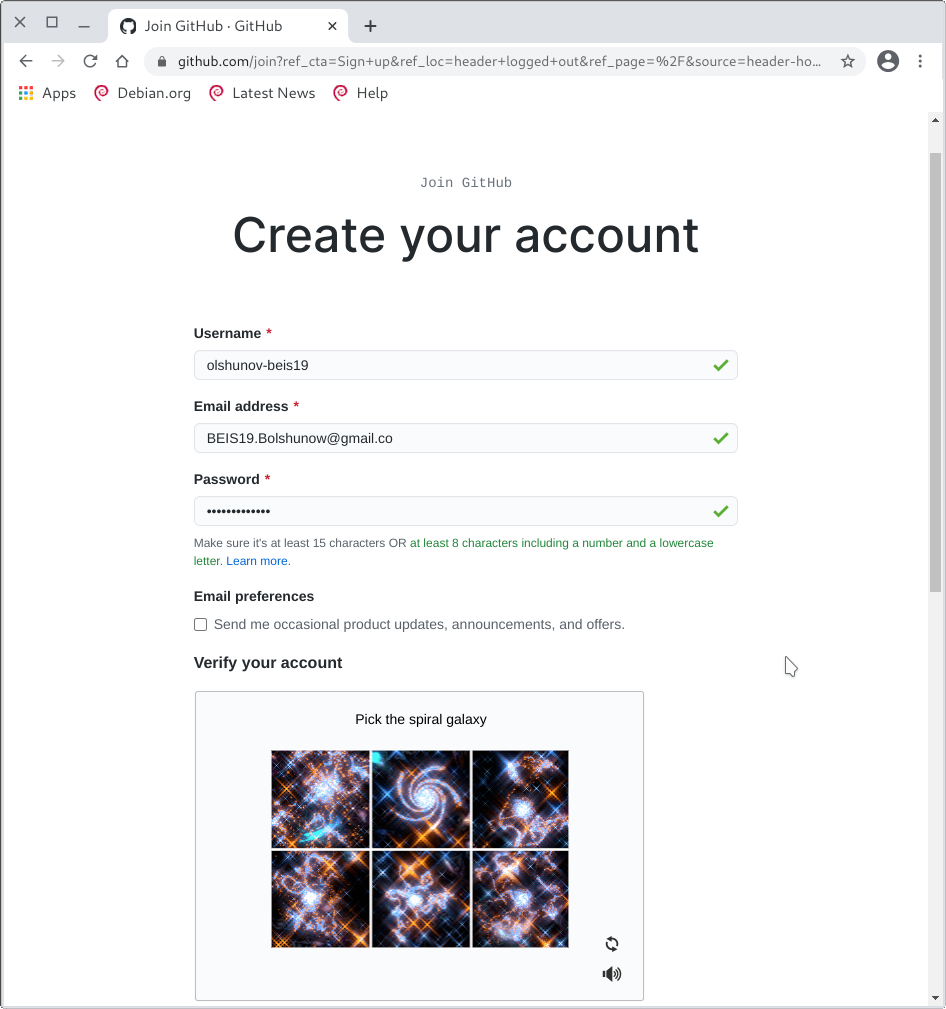


Рисунок 2.1 – Регистрация на платформе github.com

Задание 2. Создание репозитория на GitHub.

В качестве примера создан git-репозиторий на ресурсе github.com. Для создания пустого репозитория необходимо кликнуть по ссылке «New». Далее откроется страница заполнения параметров репозитория (рисунок 2.2).

Заполнение полей также указано на рисунке 2.2.

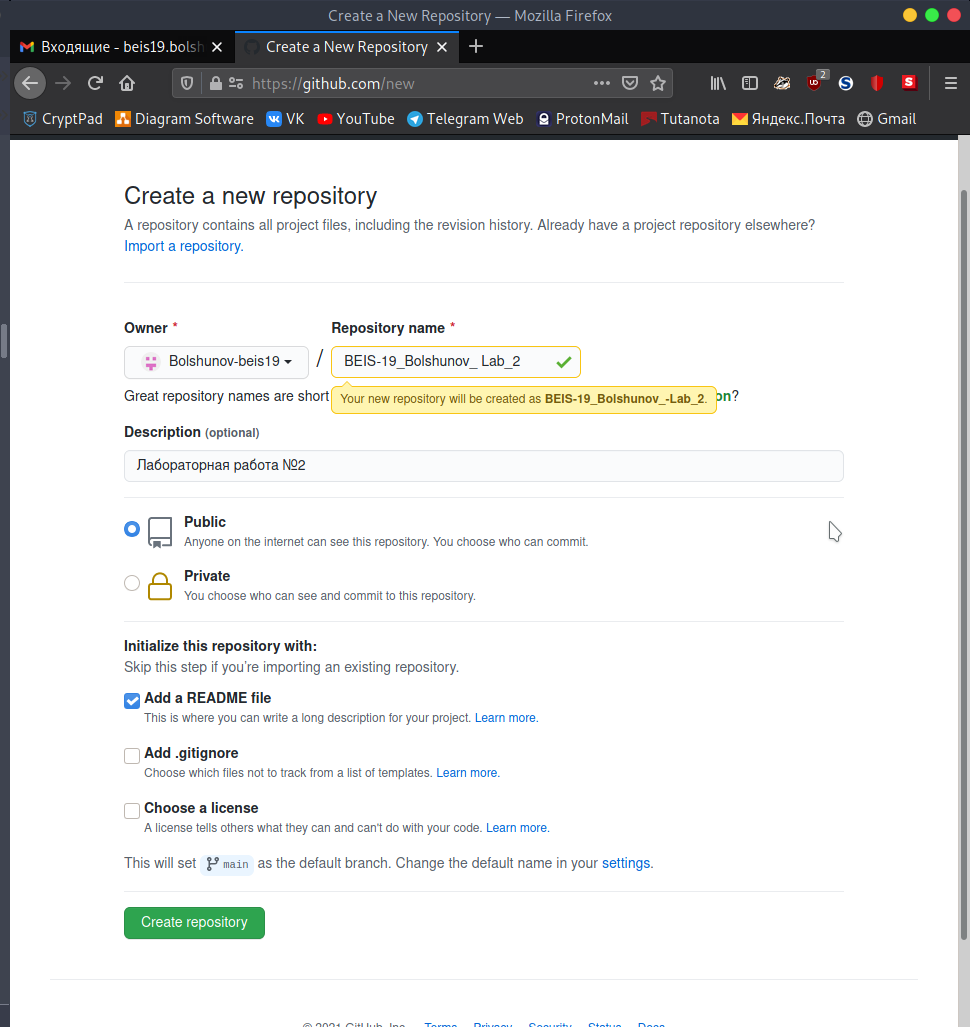


Рисунок . – Создание нового репозитория на сайте github.com

Пока в репозитории находится только файл README.md.

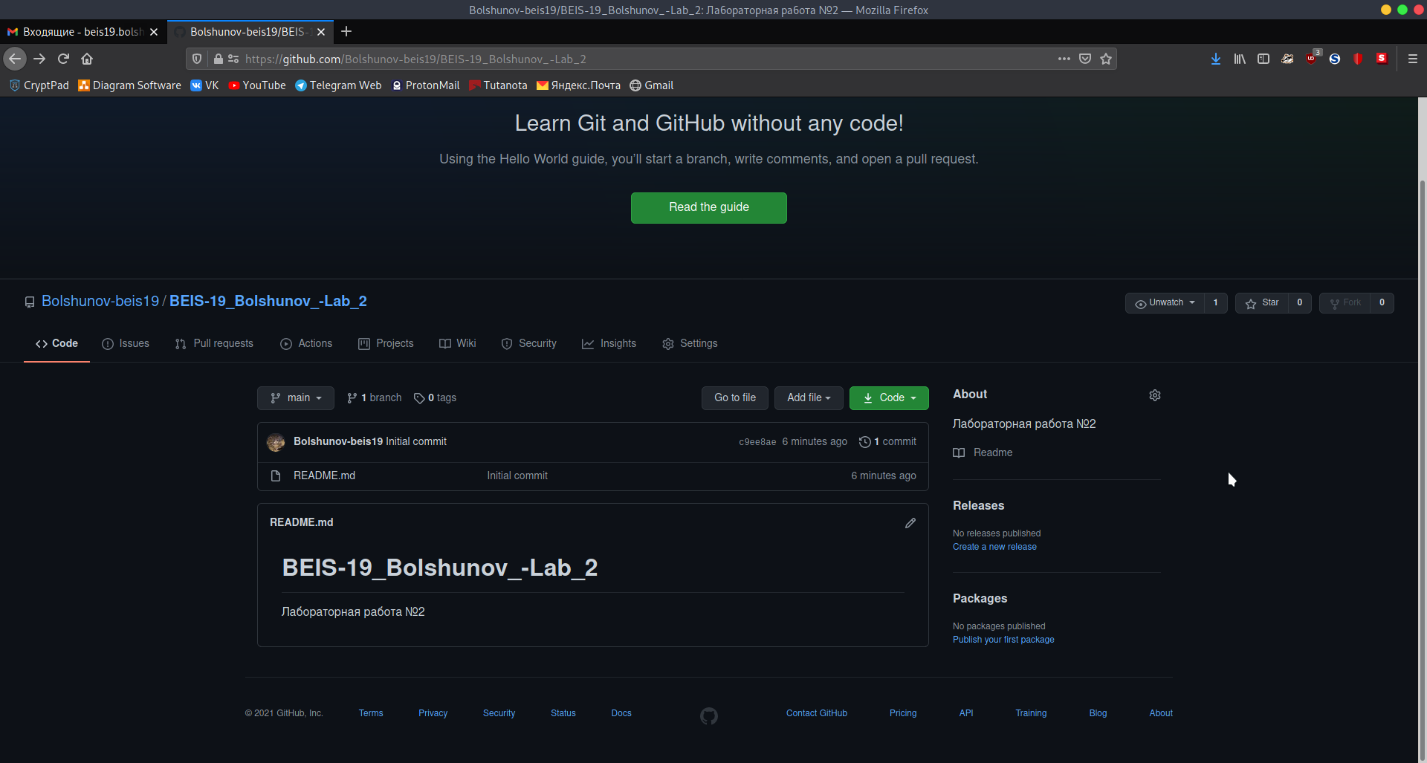


Рисунок . – Окно структуры свежесозданного репозитория

В разделе «branch» ветка main переименована в ветку master. Процесс представлен на рисунке 2.4.

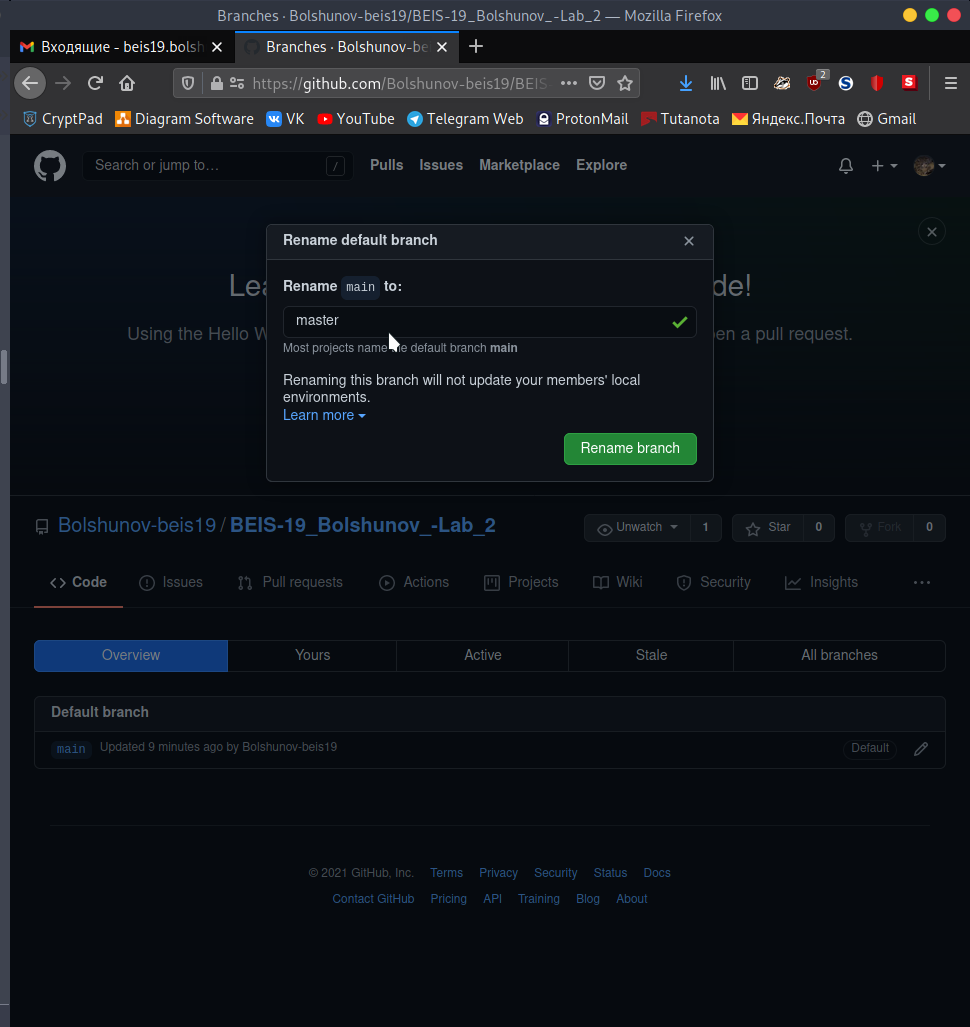


Рисунок . – Переименование ветки main в master

Задание 3. Использование встроенного редактора кода.

Требуется отредактировать файл README.md. Для этого нужно выбрать файл и нажать на кнопку редактирования, затем откроется встроенный редактор кода.

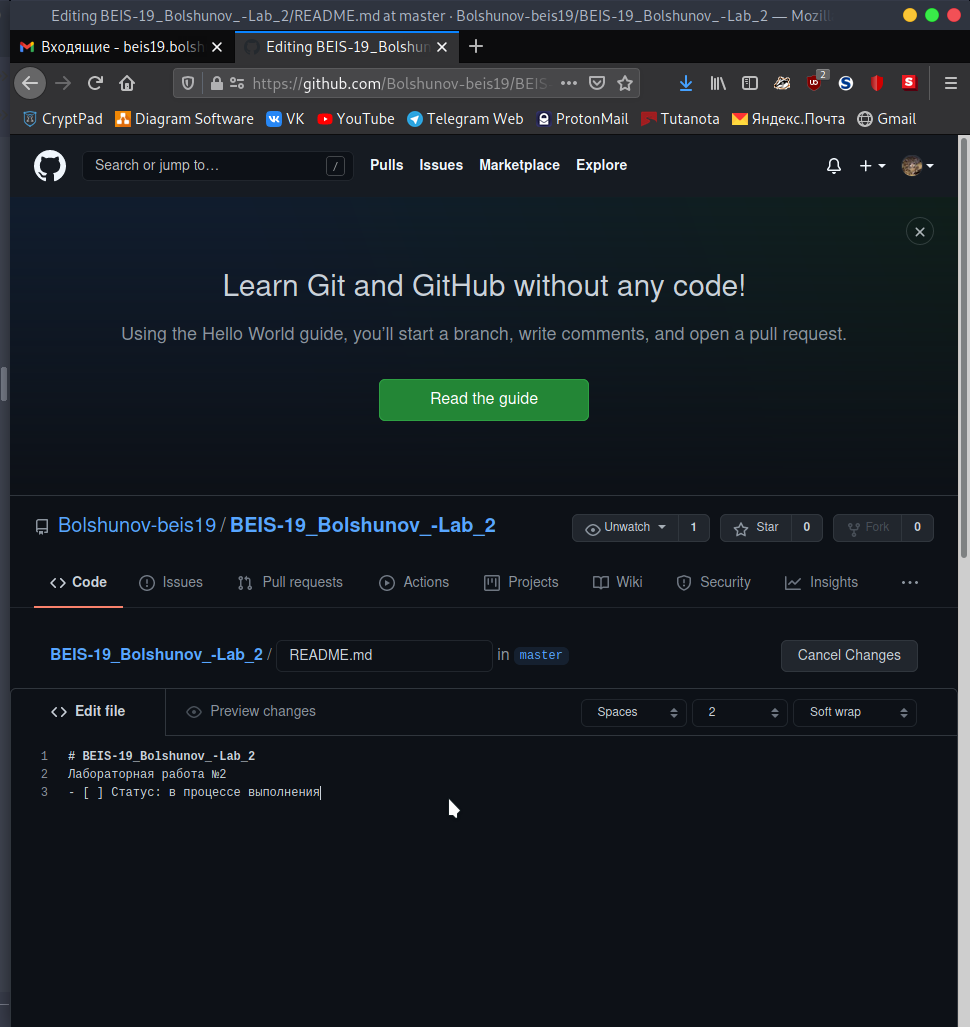


Рисунок . – Редактирование README.md во встроенном редакторе

Далее добавлено описание внесенных изменений. После нажатия «Commit changes» изменения были сохранены в репозитории.

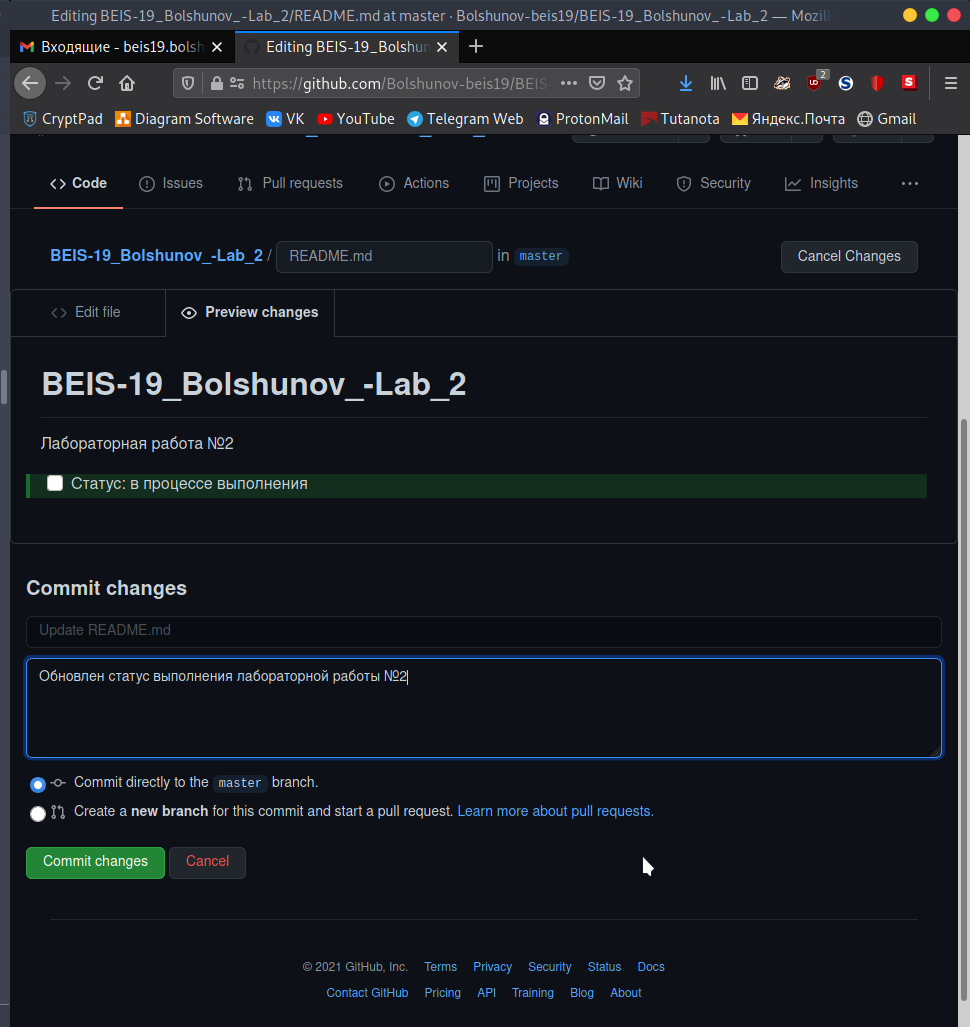


Рисунок . – Сохранение изменений

Задание 4. Создание нового проекта в локальном репозитории и последующая загрузка на удаленный сервер при помощи терминала

Ранее созданный репозиторий скопирован в локальную папку с помощью команды «git clone [address]». Команды выполнялись в Linux дистрибутиве Parrot OS, поэтому вводимые команды и формат вывода могут отличаться от аналогичных в Windows.

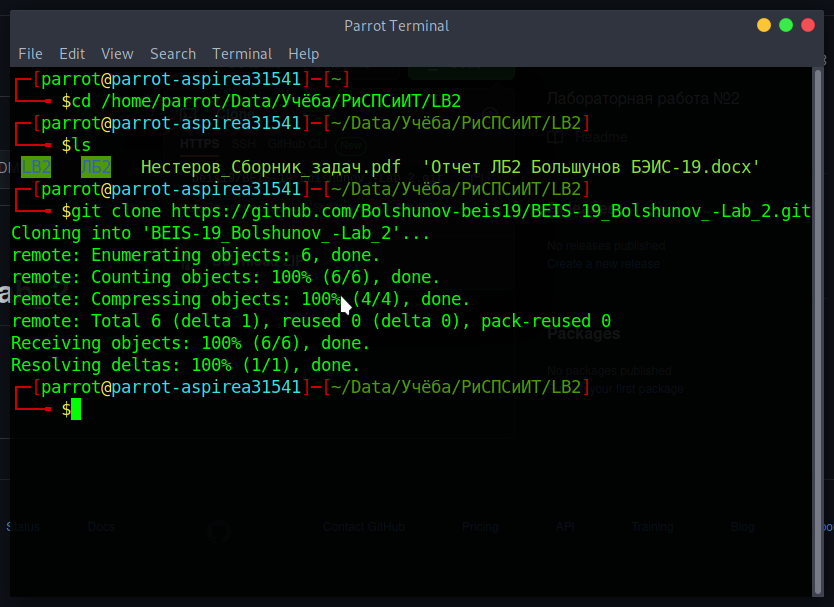


Рисунок . – Копирование репозитория

В Visual Studio создан простой проект Windows Forms согласно заданию.

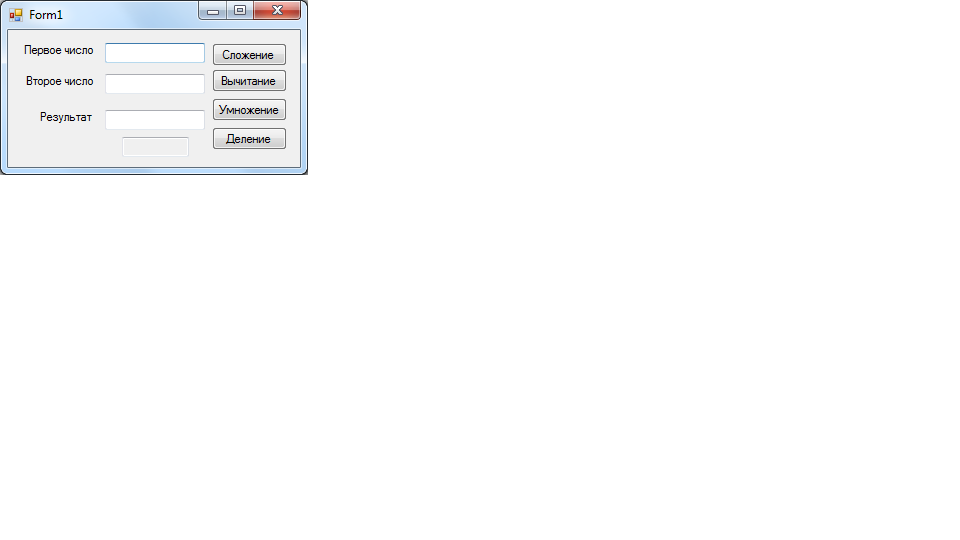


Рисунок . – Окно приложения для примера

Для загрузки файлов используется командная строка. Прежде всего необходимо посмотреть статус изменений, для этого используется команда «git status» (рисунок 2.9)

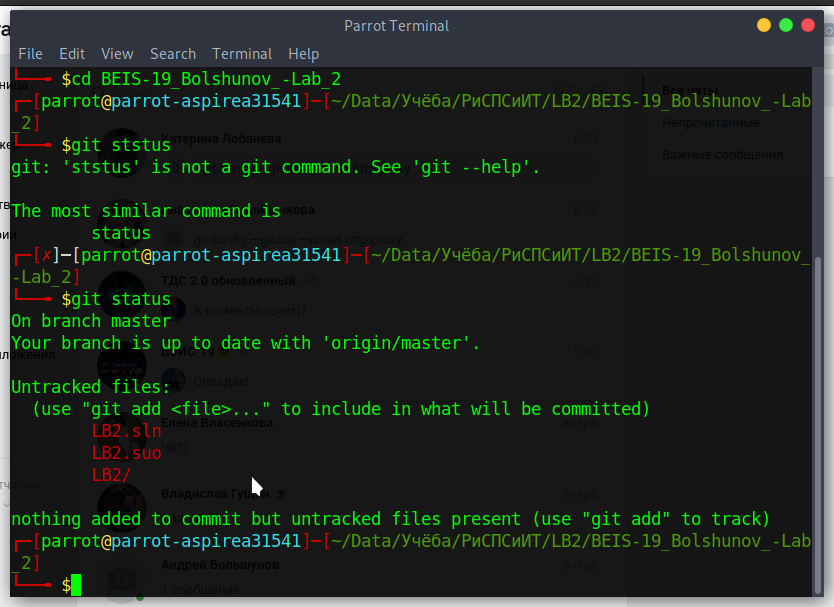


Рисунок . – Статус изменений

Как можно заметить, файлы проекта выделены красным, так как они новые и не были загружены в репозиторий. Для загрузки всех файлов в репозиторий используется команда «git add \*.\*». Однако, при этом не добавилась папка, поэтому для её добавления используем «git add LB2/».

Снова проверяем статус «git status» и видим, что файлы добавлены в список отслеживаемых.

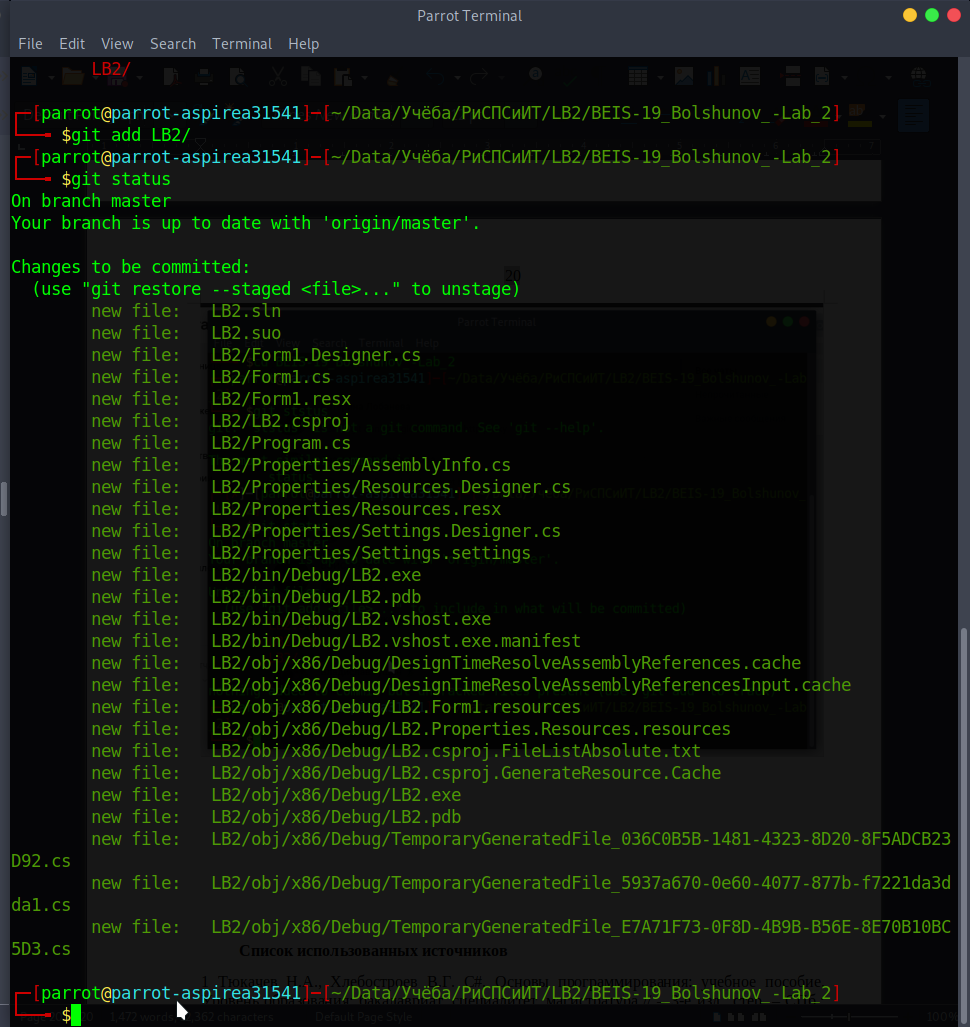


Рисунок . – Добавление файлов проекта в «отслеживаемые» и просмотр повторного статуса репозитория

Далее выполнен коммит, то есть зафиксирован текущий список изменений и связан с определённым названием. Ключ «-m» в команде ниже используется для указания комментария к текущему коммиту. Таким образом, команда для выполнения кода будет следующей: git commit -m «Lab-2».

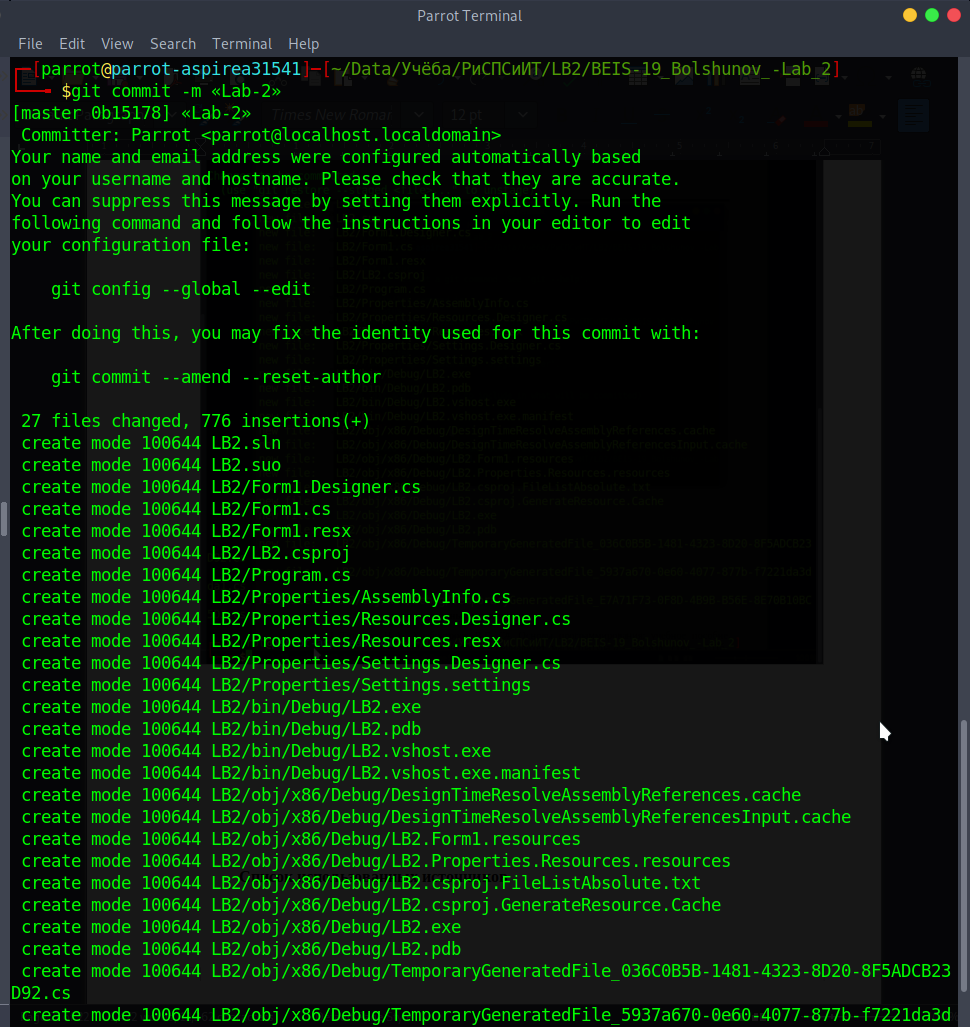


Рисунок . – Результат выполнения команды коммита

Как мы можем увидеть из рисунка 2.11, после выполнения команды в окне терминала отображаются все изменения, которые были выполнены с последнего коммита – отображается количество измененный функций, количество вставок и удалений.

Этап загрузки файлов в удалённый репозиторий осуществлён при помощи команды: «git push origin master»

Как можно увидеть на рисунке 2.12, в результате выполнения команды была выполнена загрузка файлов на удалённый репозиторий.

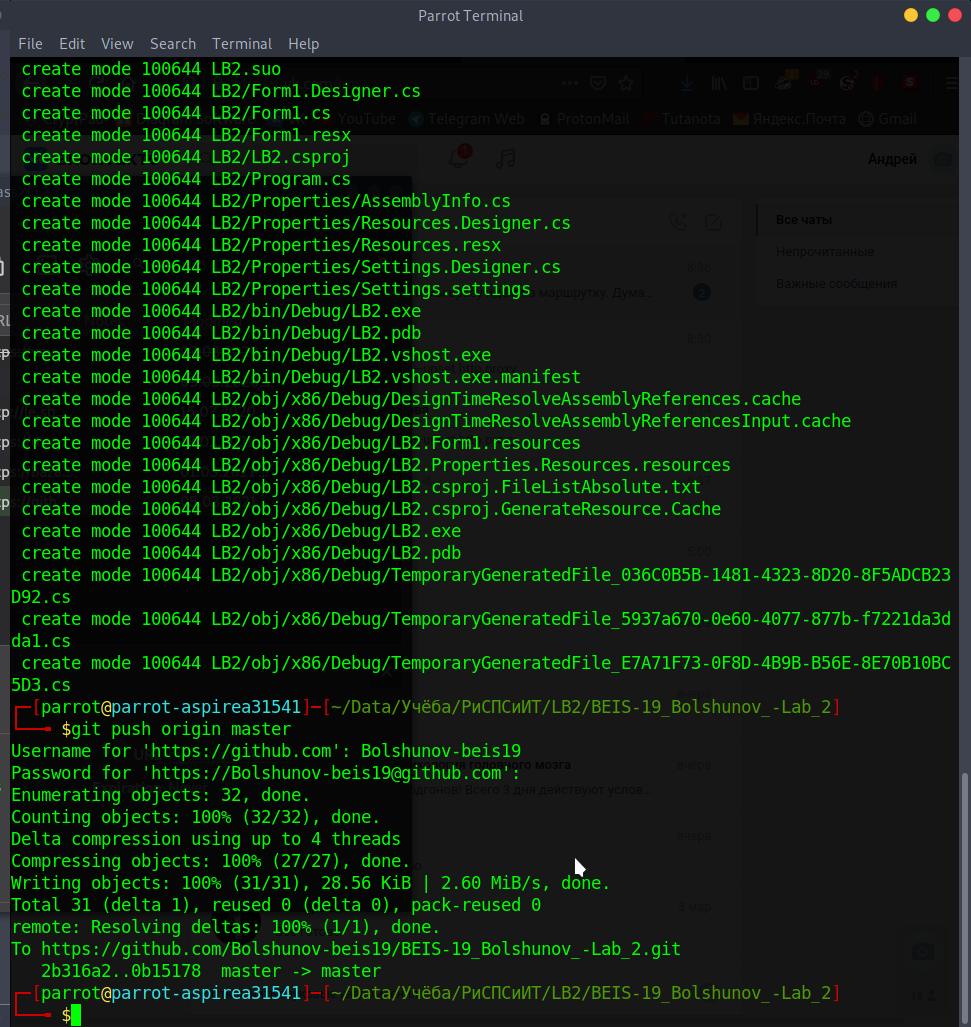


Рисунок . – Результат выполнения команды коммита

В результате после успешного выполнения коммита, на сайте github.com в репозитории доступны файлы из локальной папке, что показано на рисунке 2.13.

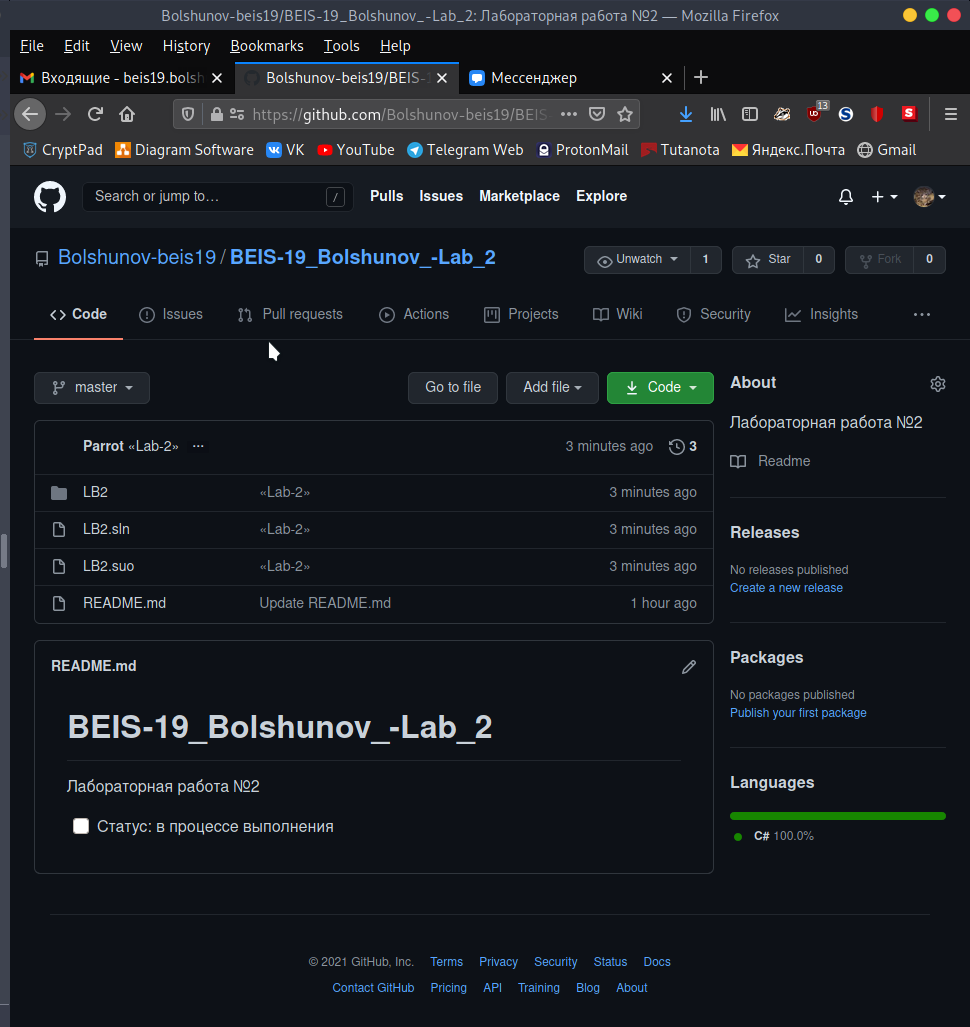


Рисунок . – Удалённый репозиторий на сайте github.com

Задание 5. Внесение изменений в проект и загрузка обновлений в удалённый репозиторий.

Выполнено изменение реализованного программного обеспечения ­ изменен тип данных на вещественный.



Рисунок . – Смена типа данных на вещественный

Изменения были затронуты в файле Form1.cs в папке LB2. Для загрузки изменений на удалённый репозиторий необходимо добавить изменённые файлы командой: «git add LB2/».

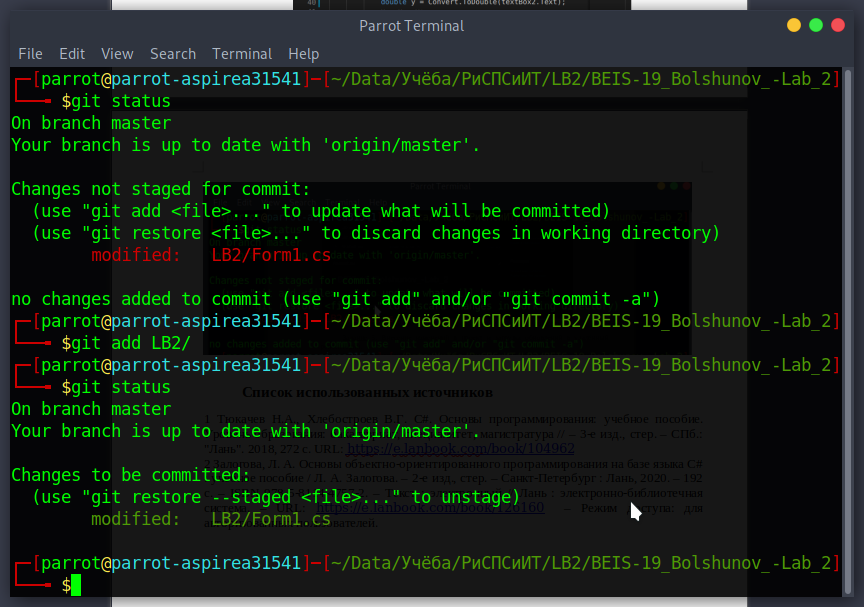


Рисунок . – Загрузка изменений

Далее выполнена команда создания нового коммита с комментарием: «git commit -m «изменён тип переменных»».

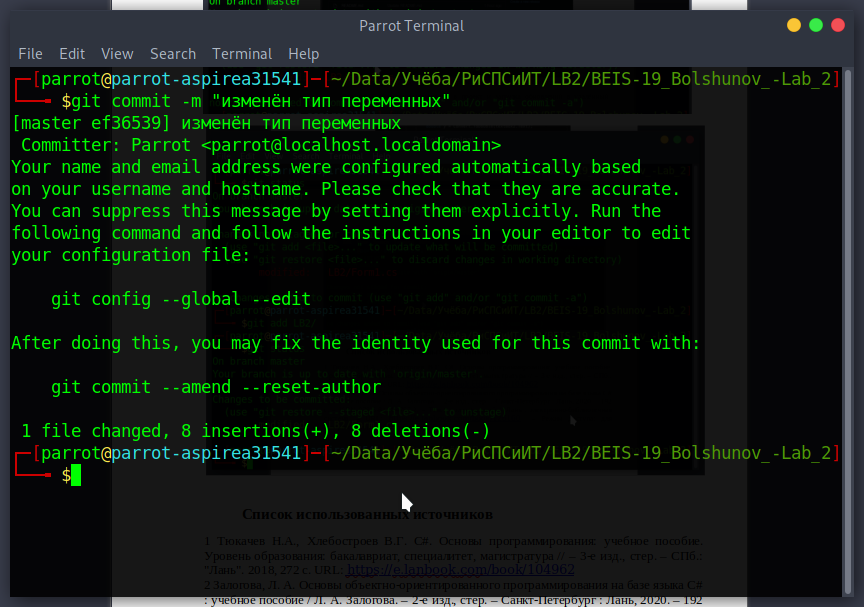


Рисунок . – Добавление коммита для внесенных изменений

После выполнения коммита необходимо зафиксировать изменения на удалённом репозитории – для этого выполним команду «git push origin master».

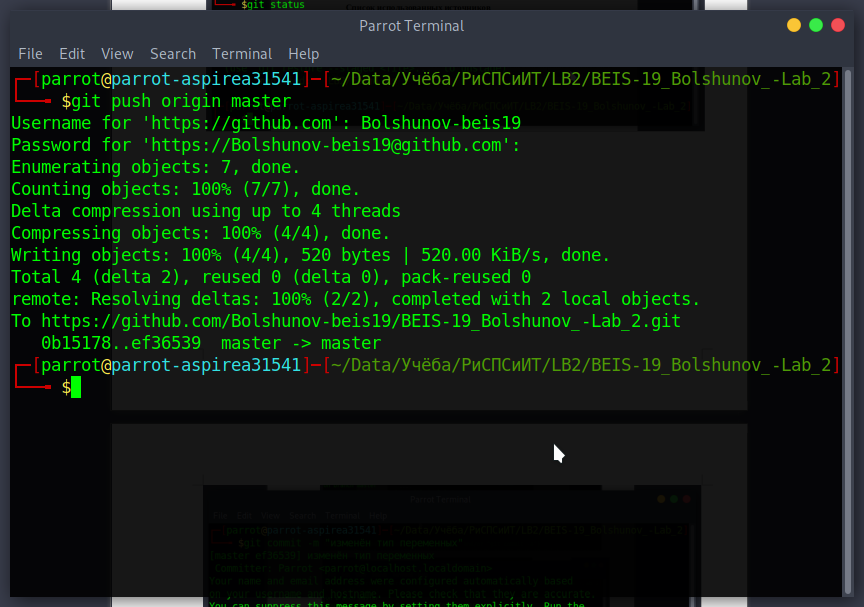
****

Рисунок . – Добавление коммита для внесенных изменений

После этого появились изменения на сайте репозитория –последний коммит был внесен только в файл в папке «LB2», о чём свидетельствует соответствующий комментарий:

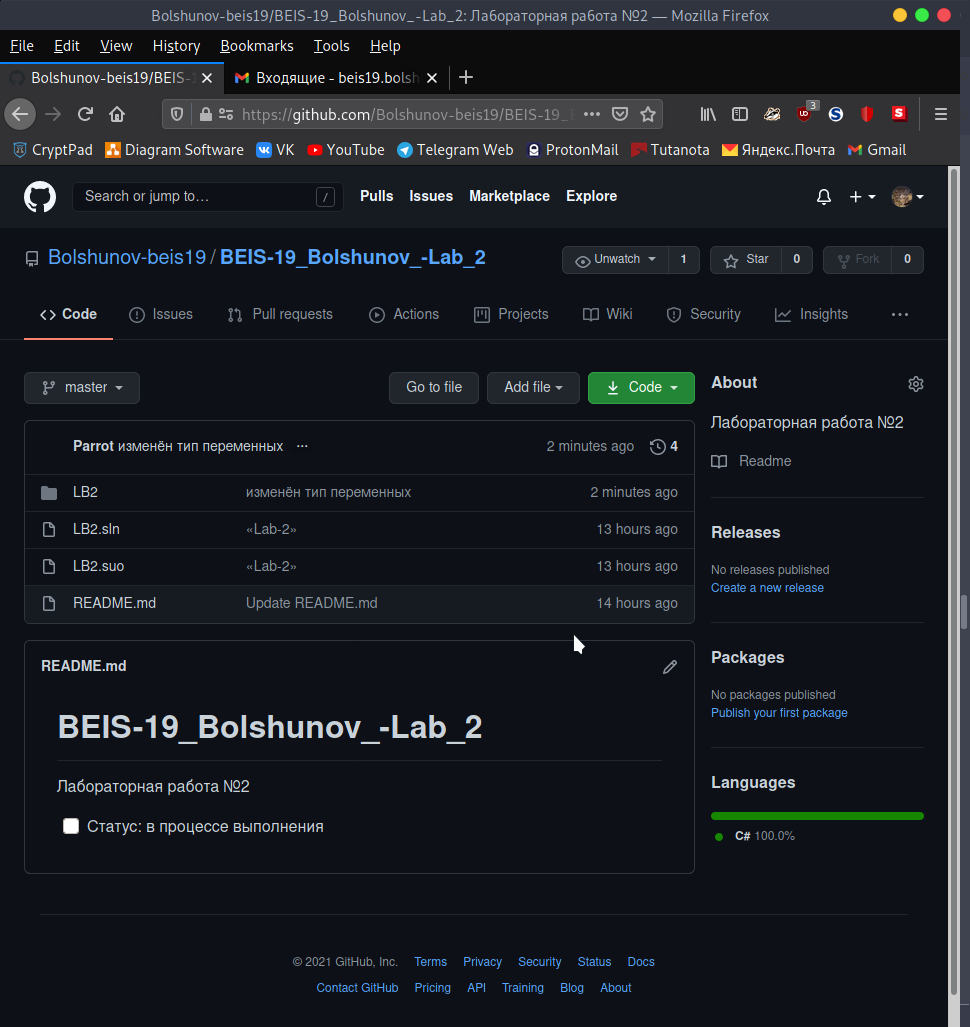


Рисунок . – Просмотр изменений на сайте репозитория

Выполнен просмотр изменений коммита «изменён тип переменных». В открытой вкладке отображаются все коммиты с временем изменения и уникальным идентификатором (рисунок 2.19).

После нажатия на конкретный коммит открывается окно просмотра внесенных изменений. В строках записи переменных были внесены изменения в конкретный тип (рисунок 2.20).

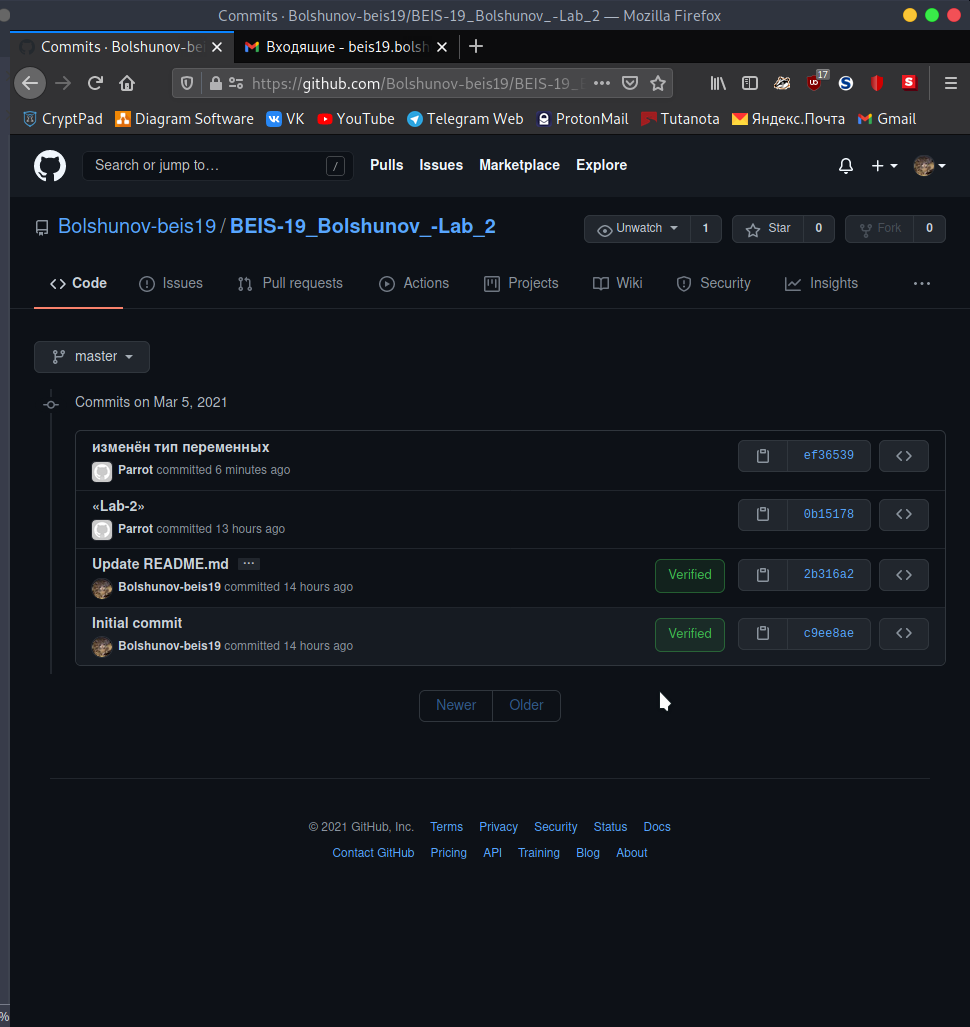


Рисунок . – Окно просмотра коммитов репозитория

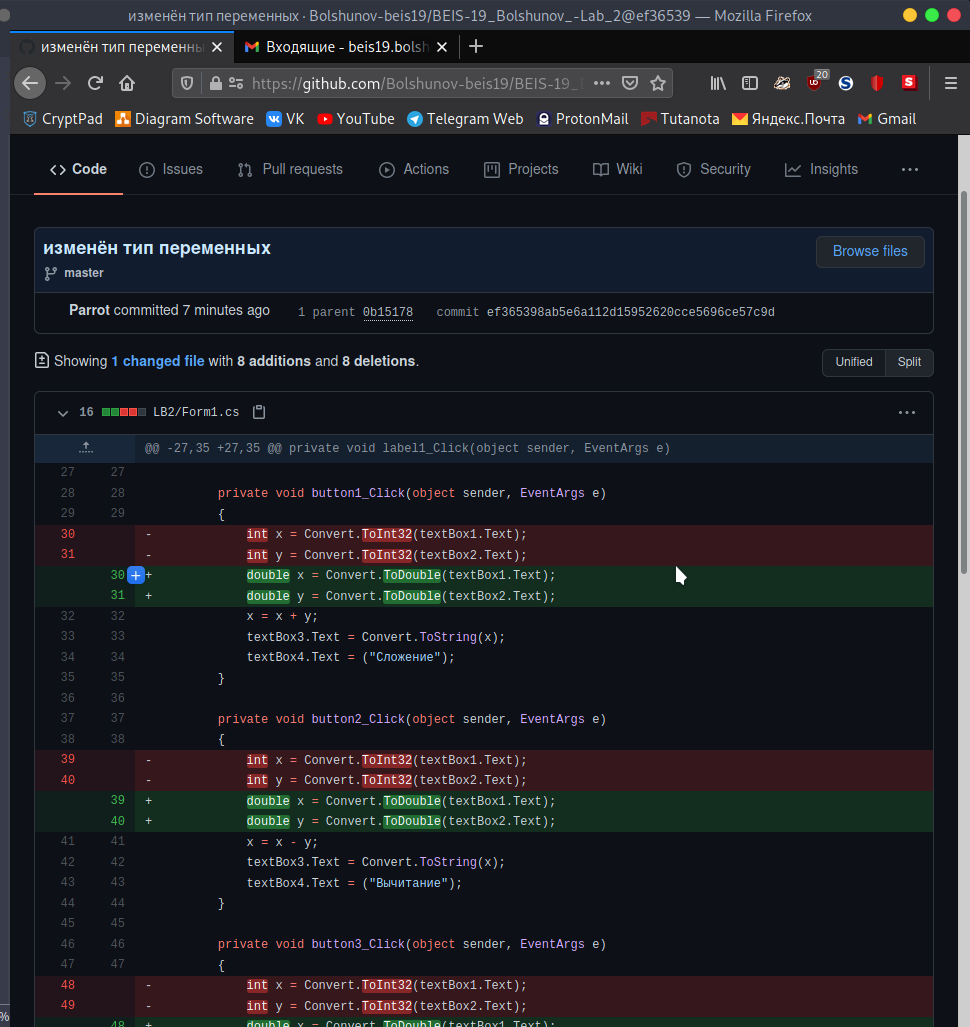


Рисунок . – Окно просмотра действий коммита

Задание 6. Загрузка файлов в репозиторий github посредством веб-интерфейса

Для загрузки проекта необходимо открыть репозиторий на сайте github.com и нажать на кнопку «Add file», а затем в выпадающем меню «Upload files». В следующем окне необходимо выбрать или переместить файлы проекта в соответствующее окно.

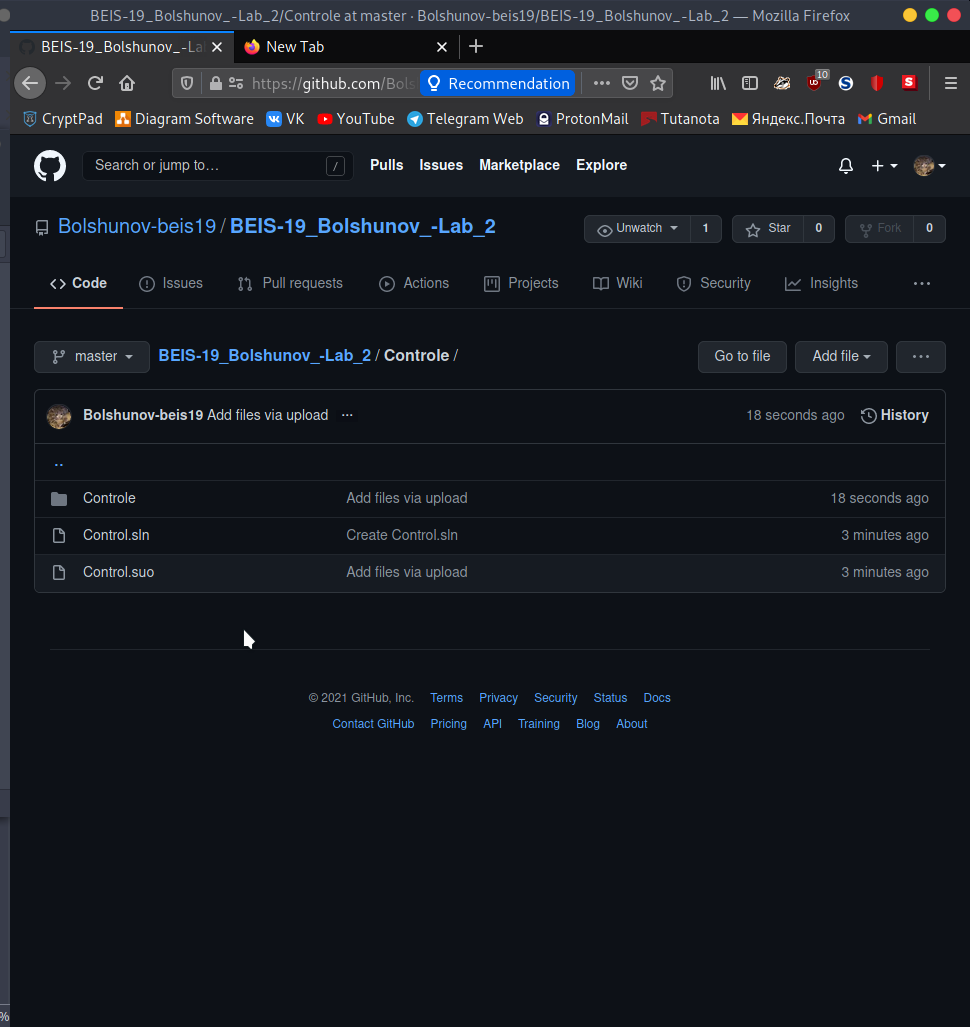


Рисунок . – Добавление файлов контрольной работы

Задание 7. Загрузка файлов в репозиторий github при помощи клиента Git.

В программе git Gui создан новый репозиторий «New\_Repo».

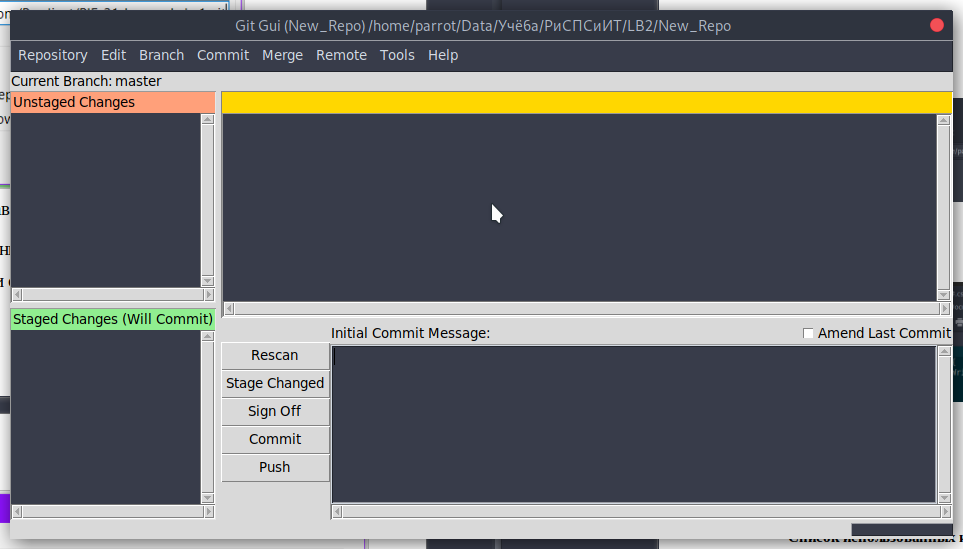


Рисунок . – Новый репозиторий в git Gui

Далее через файловый менеджер в репозиторий добавлен файл «Hello\_World».

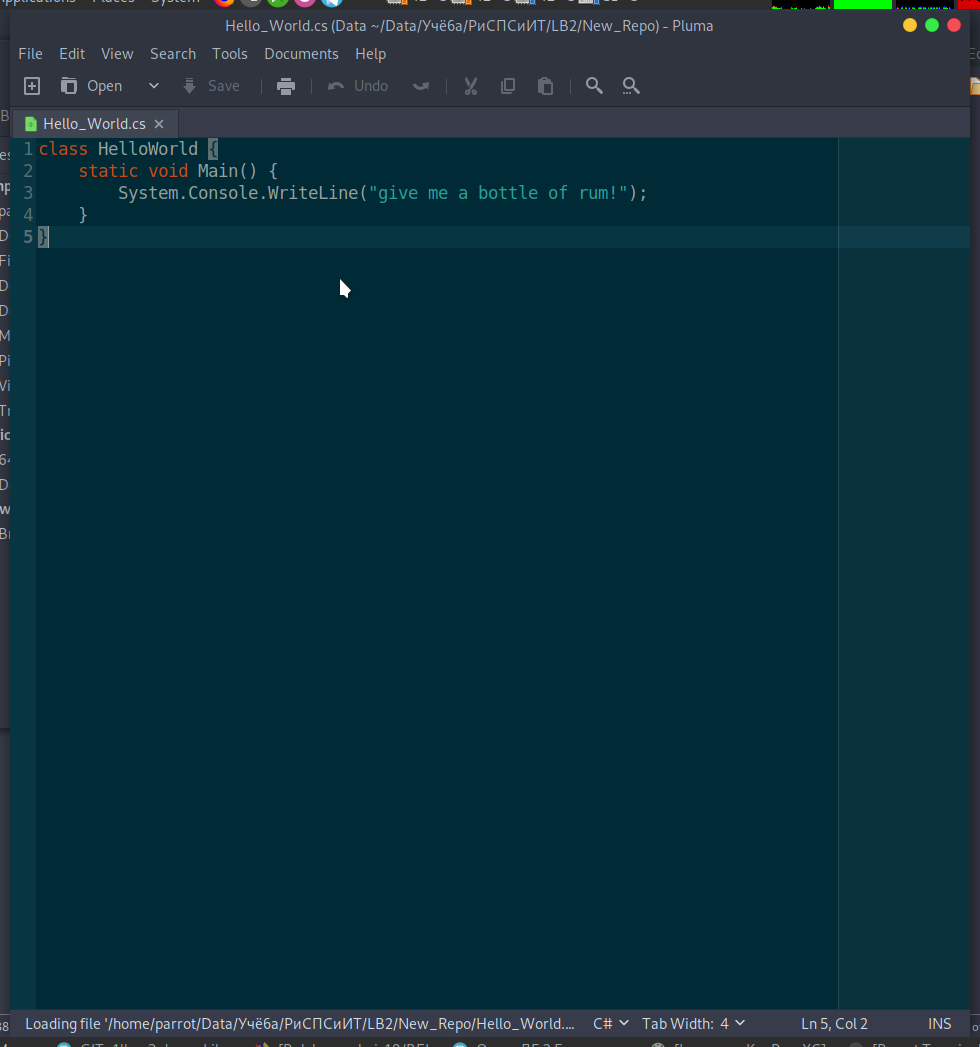
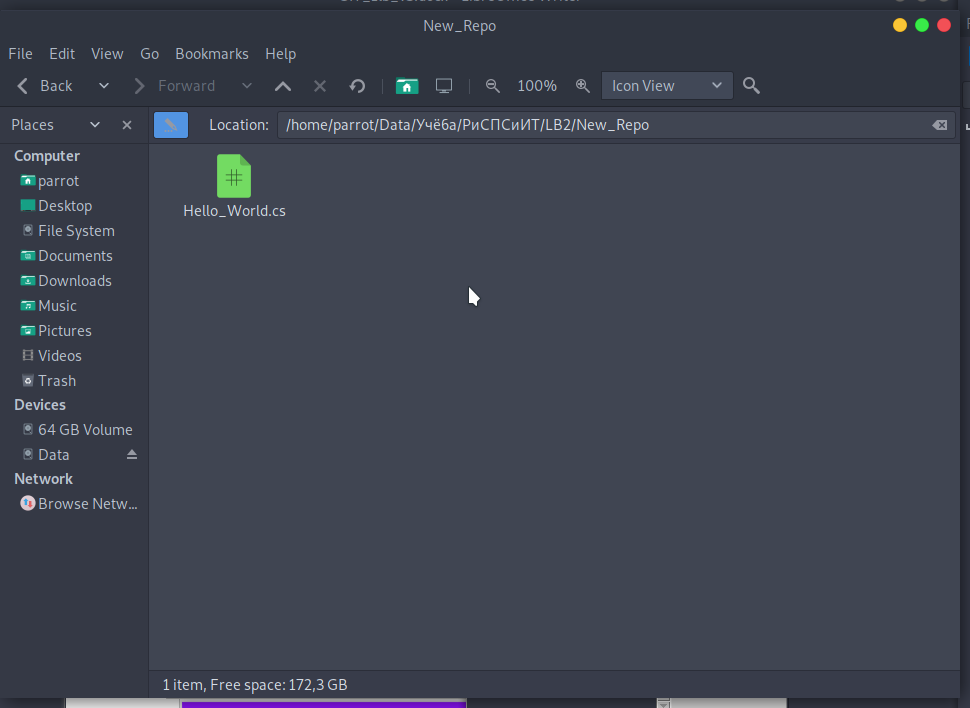


Рисунок . – Файл Hello\_World

После сохранения изменений на вкладке «Unstaged Changes» виден созданный файл и в окне справа его содержимое (рисунок 2.24).

При нажатии на кнопку фиксации изменений (Stage Changed), в окне предлагается отметить один не отслеживаемый файл. На данном этапе нужно нажать кнопку «Yes».

Также в окне «Initial Commit Message» написано наименование коммита для вносимых изменений «First commit!» (рисунок 2.25).

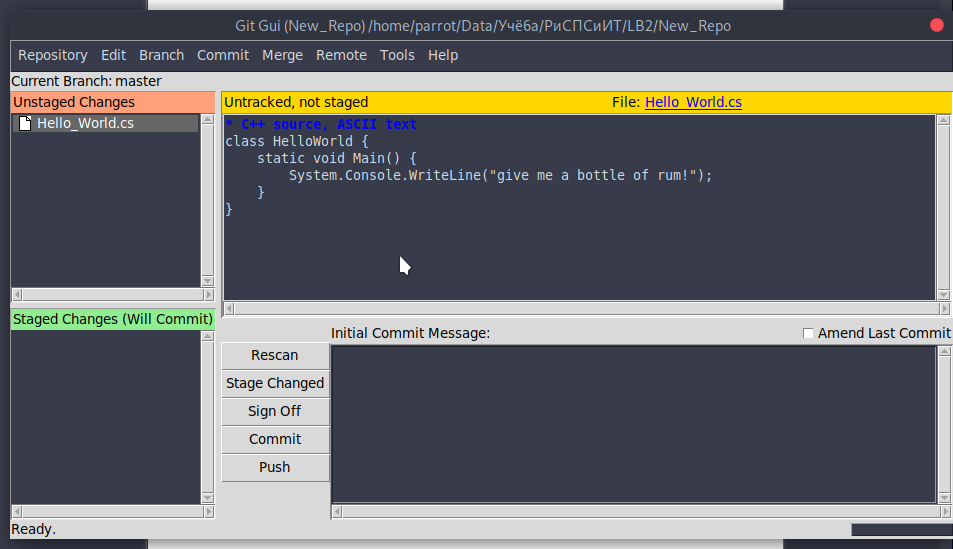


Рисунок . – Просмотр изменений в Git Gui

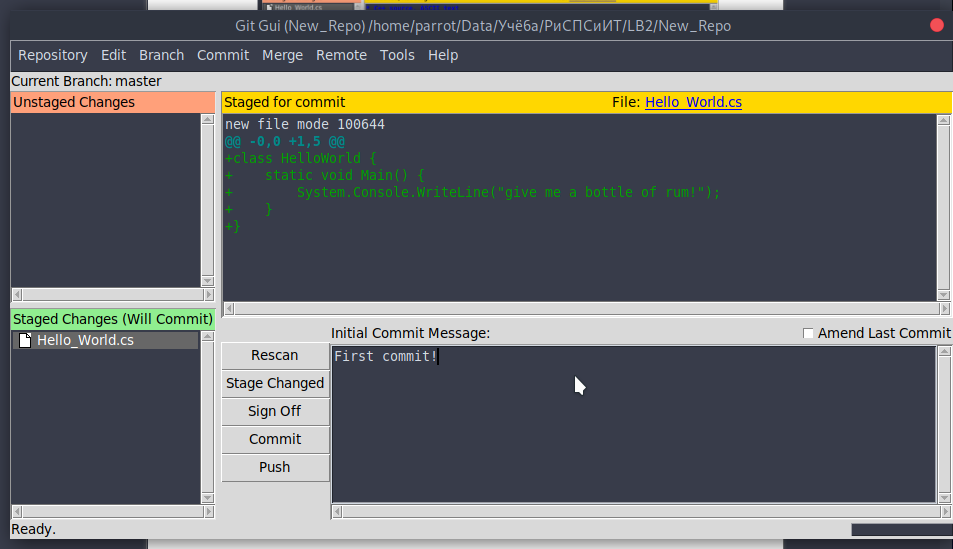


Рисунок . – Commit

После необходимо нажать на вкладку «Remote» и выбрать пункт «Add». В окне следует указать имя репозитория и его удалённую локацию (рисунок 2.26).

Все данные введены корректно, поэтому на следующем этапе отображён процесс синхронизации с удалённым репозиторием (рисунок 2.27).

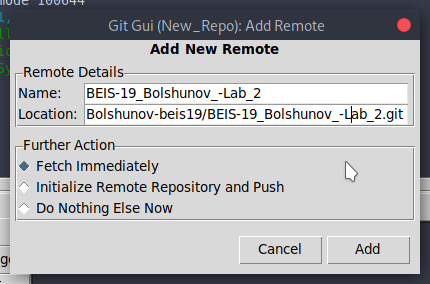


Рисунок . – Ввод данных репозитория

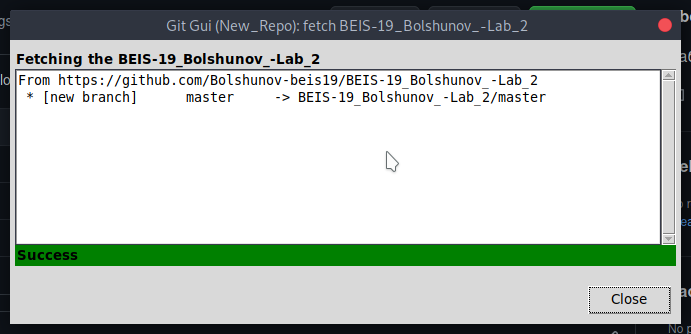


Рисунок . – Процесс синхронизации

Далее необходимо нажать кнопку «Push» в главном окне программы. В окне программы отражена ветка, в которую будет загружены изменения, а также удалённый репозиторий (рисунок 2.28). Все данные верны, после нажатия кнопки «Push» отображён прогресс загрузки изменений (рисунок 2.29).

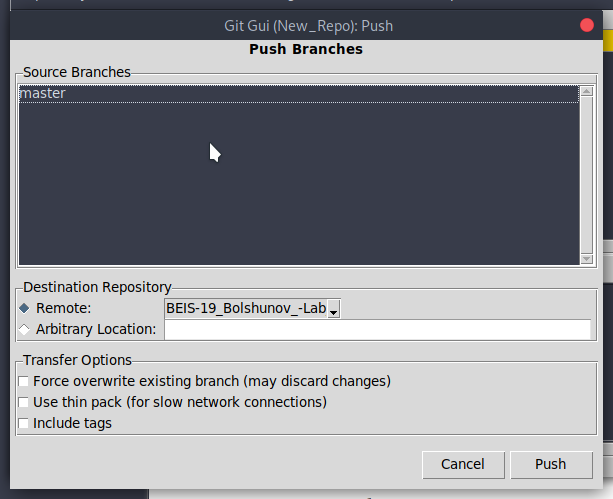


Рисунок . – Окно загрузки изменений на сервер

Рисунок . – Окно синхронизации с удалённым репозиторием

После этого на сайте репозитория появилось уведомление об изменениях и можно увидеть загруженный файл в ветке master.

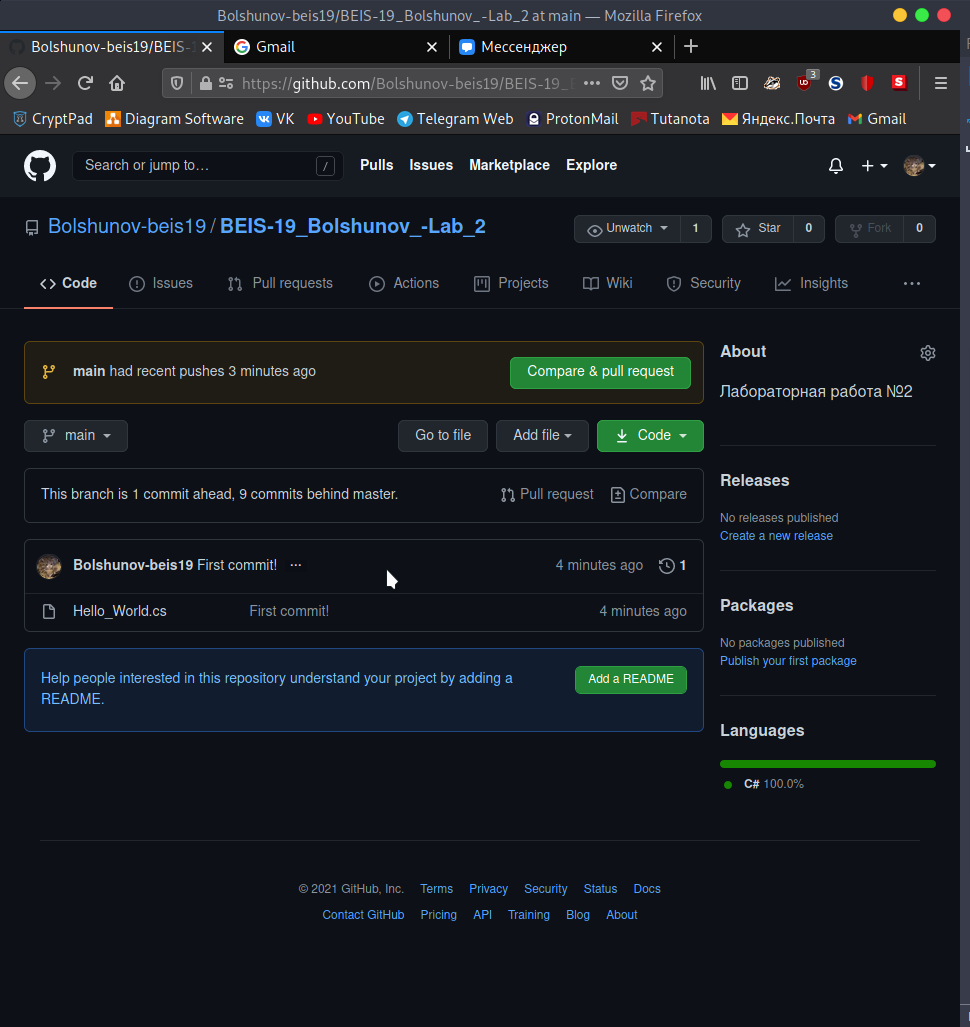


Рисунок . – Уведомление об изменениях на сайте github.com

Задание 8. Основы работы в GitHub Desktop / работа с Pull requests.

Установил GitHub Desktop на Windows 10. После входа в аккаунт github создал новый локальный обучающий репозиторий, который в дальнейшем будет отправлен в репозиторий аккаунта. Также указано, что репозиторий приватный – видимость будет только у владельца аккаунта.

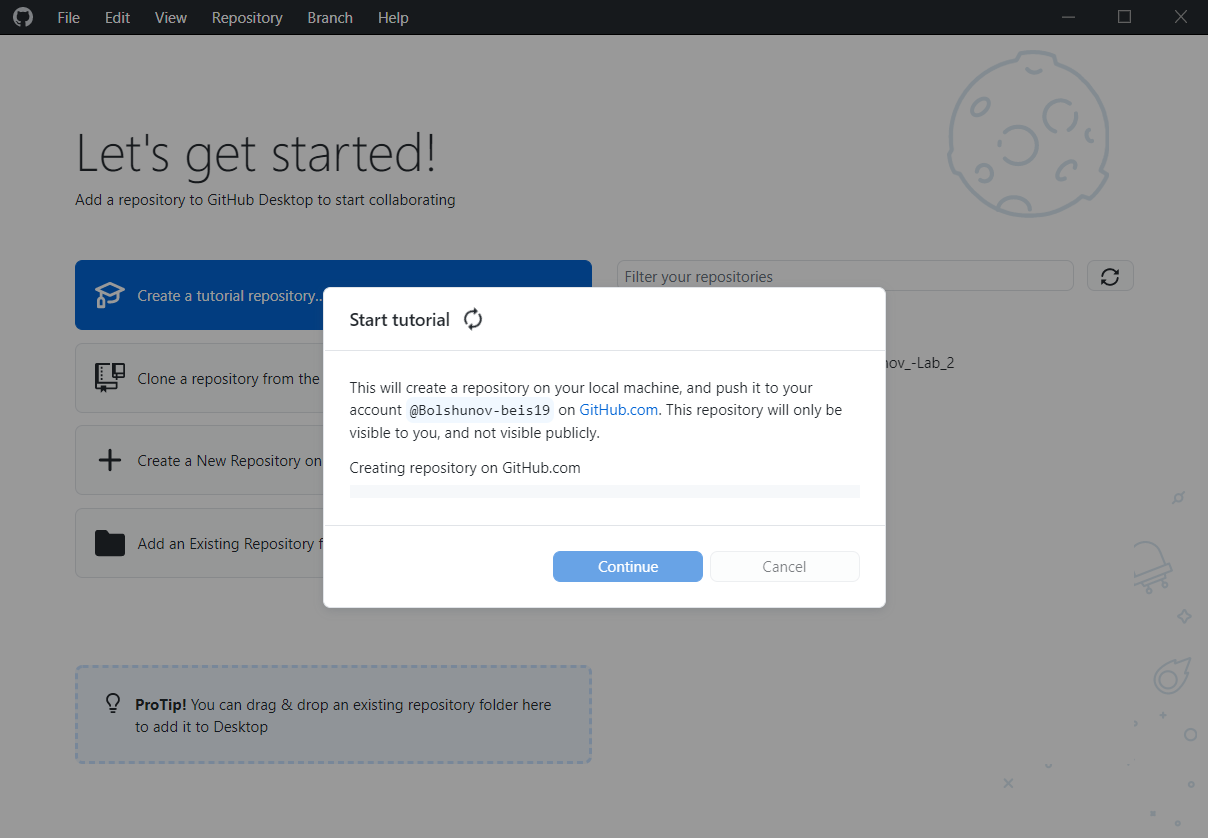


Рисунок . – Процесс создания временного репозитория

После успешного создания репозитория, справа высвечивается область быстрого старта, в которой предлагается создать новую ветку в репозитории desktop-tutorial:

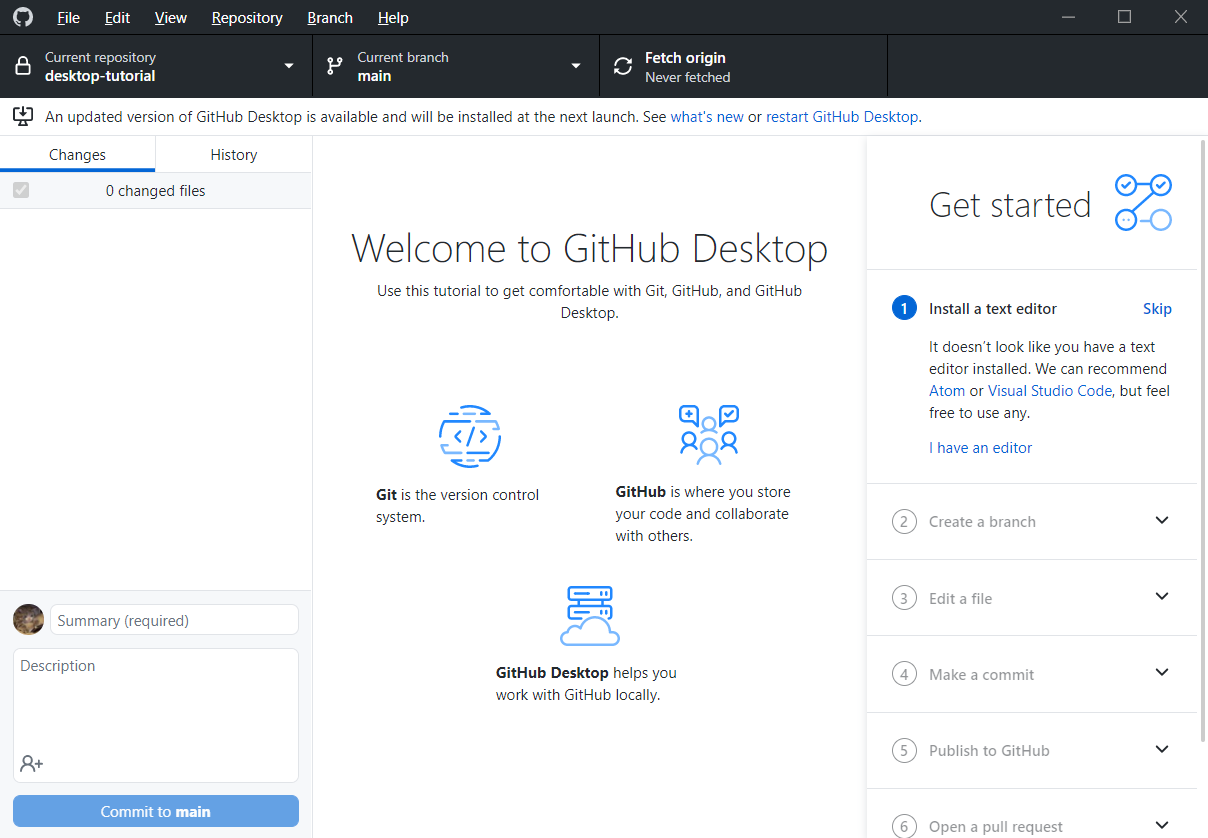


Рисунок . – Главное окно работы с GitHub Desktop

Создал новую ветку development (рисунок 2.33).

После создания ветки отредактировал файл README.md в качестве примера. После нажатия кнопки редактирования открылся стандартный текстовый редактор (рисунок 2.34).

После изменения файла сделал первый commit (рисунок 2.35).

На следующем этапе отправил изменения на удалённый сервер, дс помощью кнопки Pushing branch (рисунок 2.36).

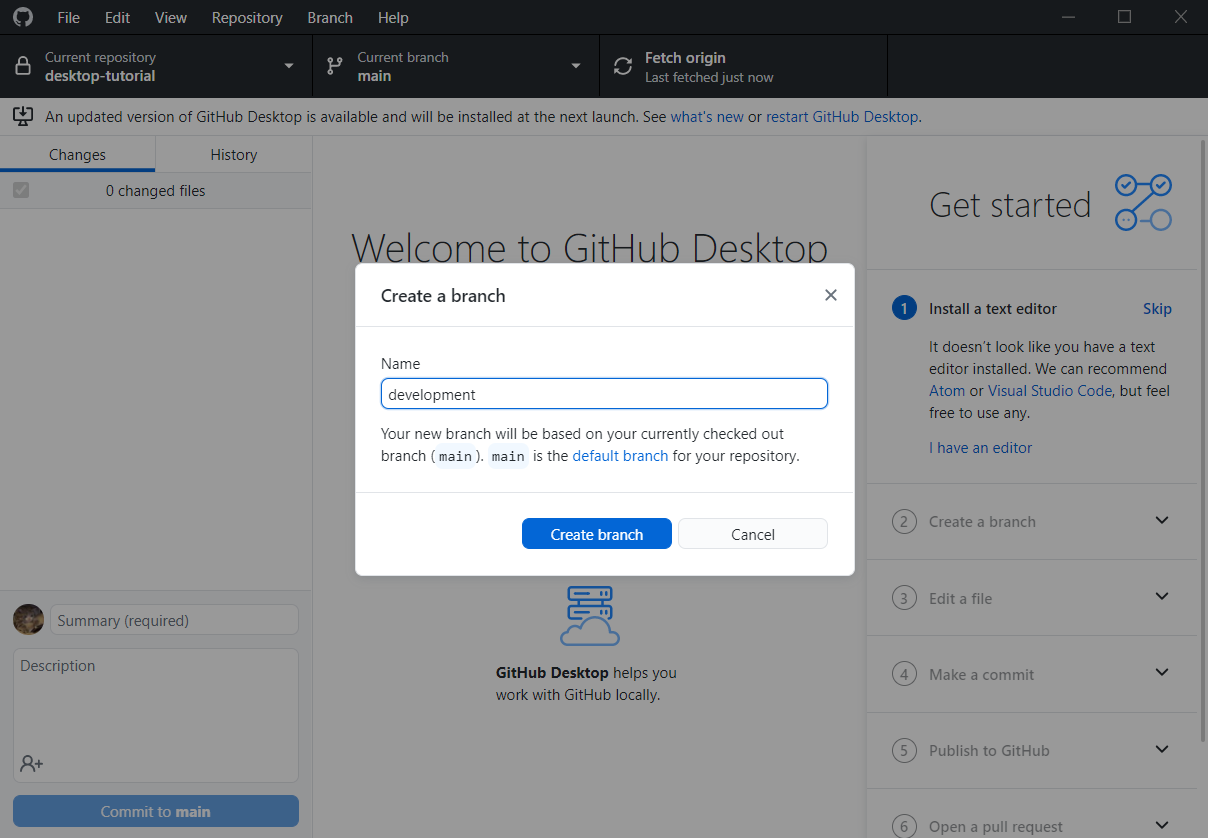


Рисунок . – Создание ветки development

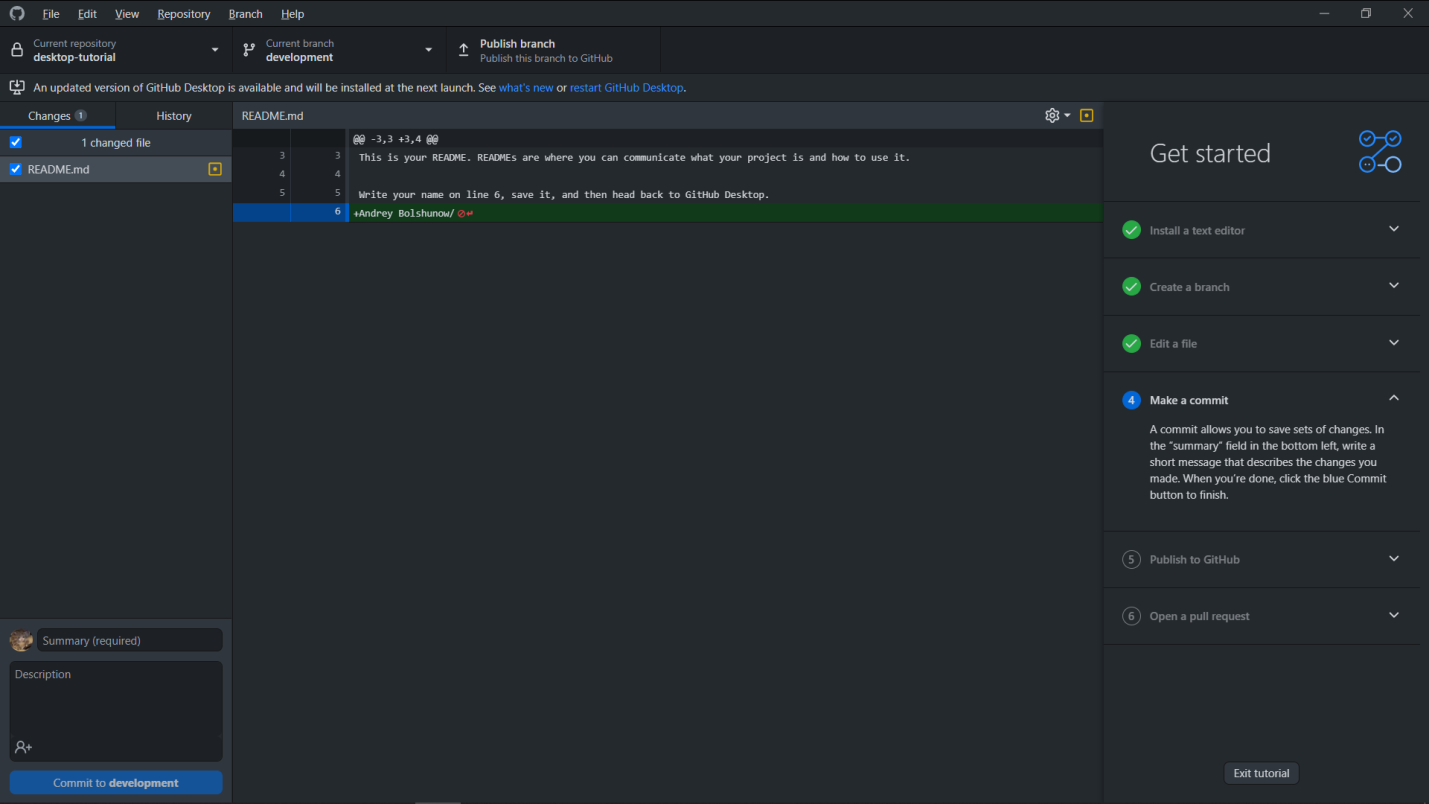


Рисунок . – Редактирование файла README.md

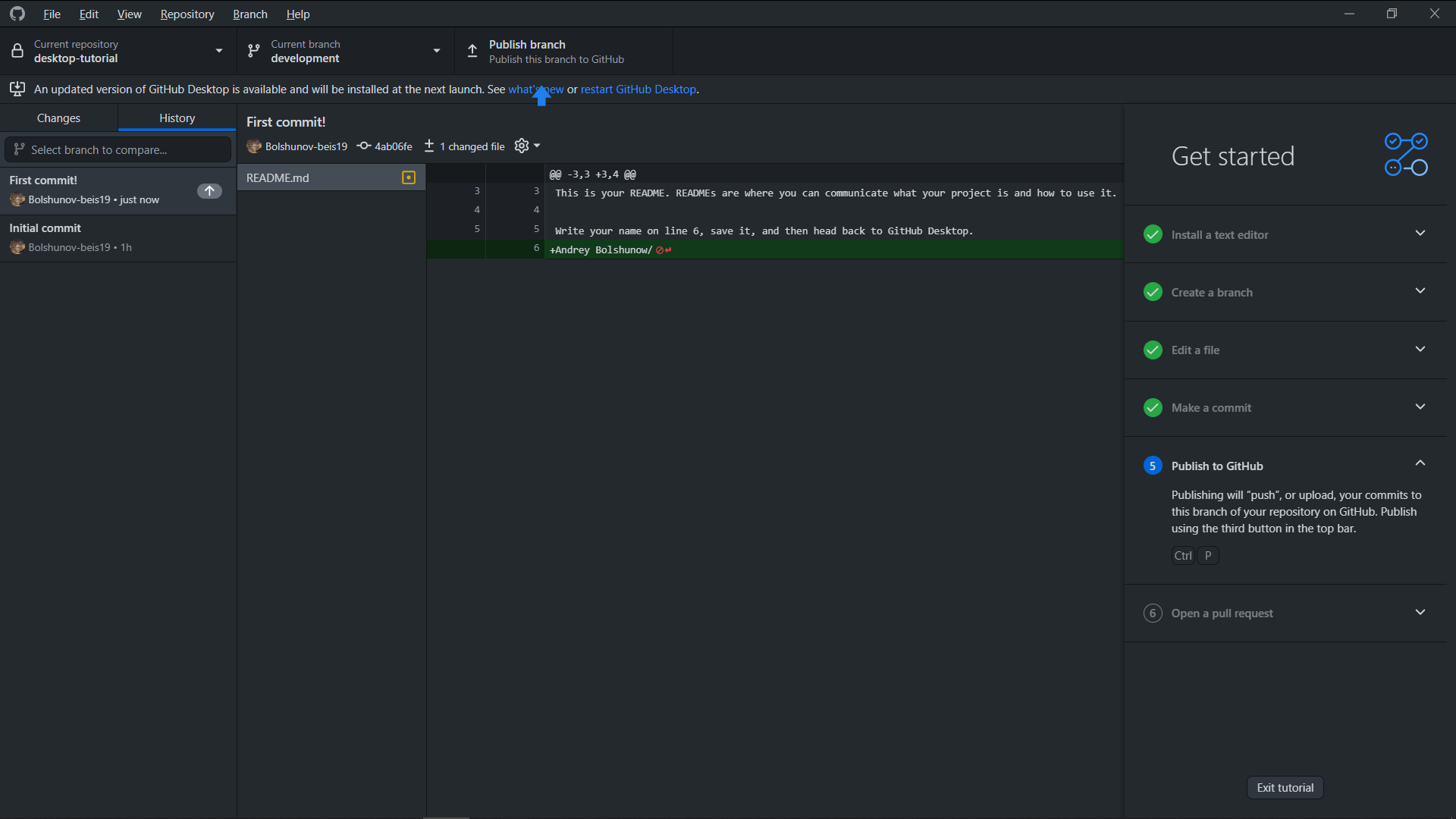


Рисунок . – Окно изменений и внесение первого коммита в ветку develop

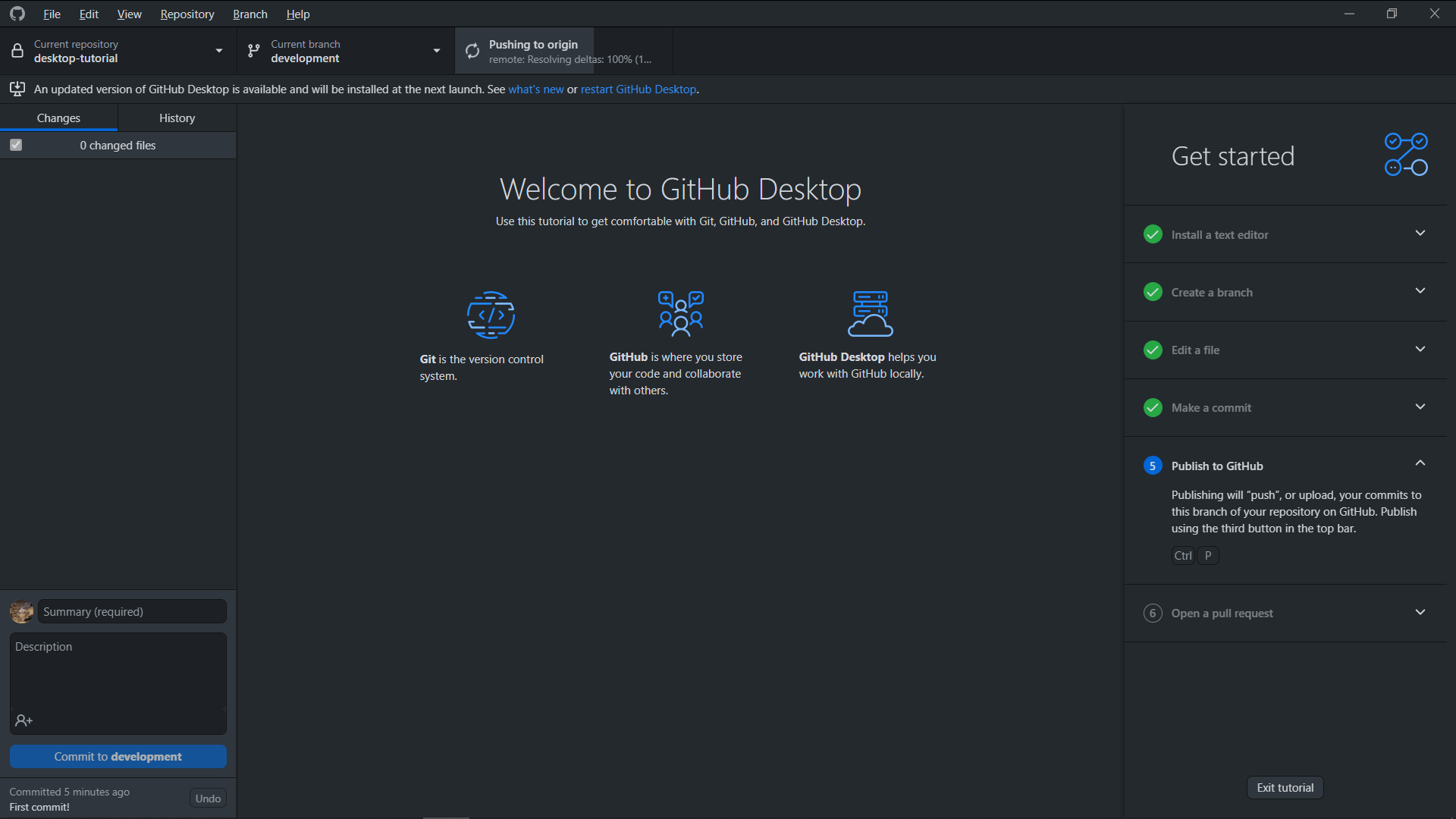


Рисунок . – Отправка изменений на удалённый сервер

Далее открыл pull request. Pull request представляет собой запрос к администратору репозитория на применение изменений из стороннего источника – ветки. Программа перенаправила на сайт на сайт github.com, где можно создать новый запрос.

В качестве примера создадим запрос и заполним необходимые поля – наименование, описание. Как видно на рисунке ниже, pull request представляет собой веб-оболочку команды merged, т.е. после открытия данного pull request, из ветки development содержимое попадёт в ветку master. После создания pull request он отображается на вкладке Pull requests.

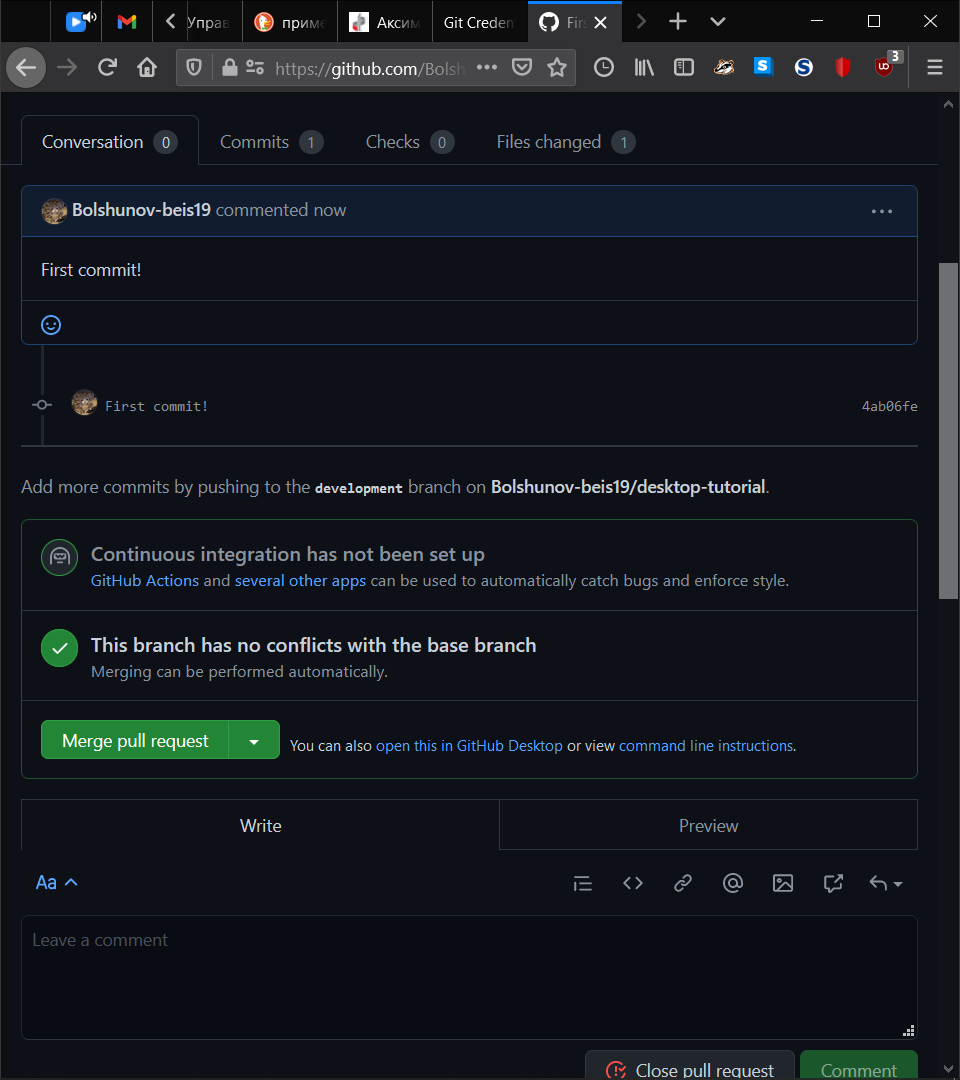


Рисунок . – Процедура принятия запроса

Далее осуществил слияние веток.

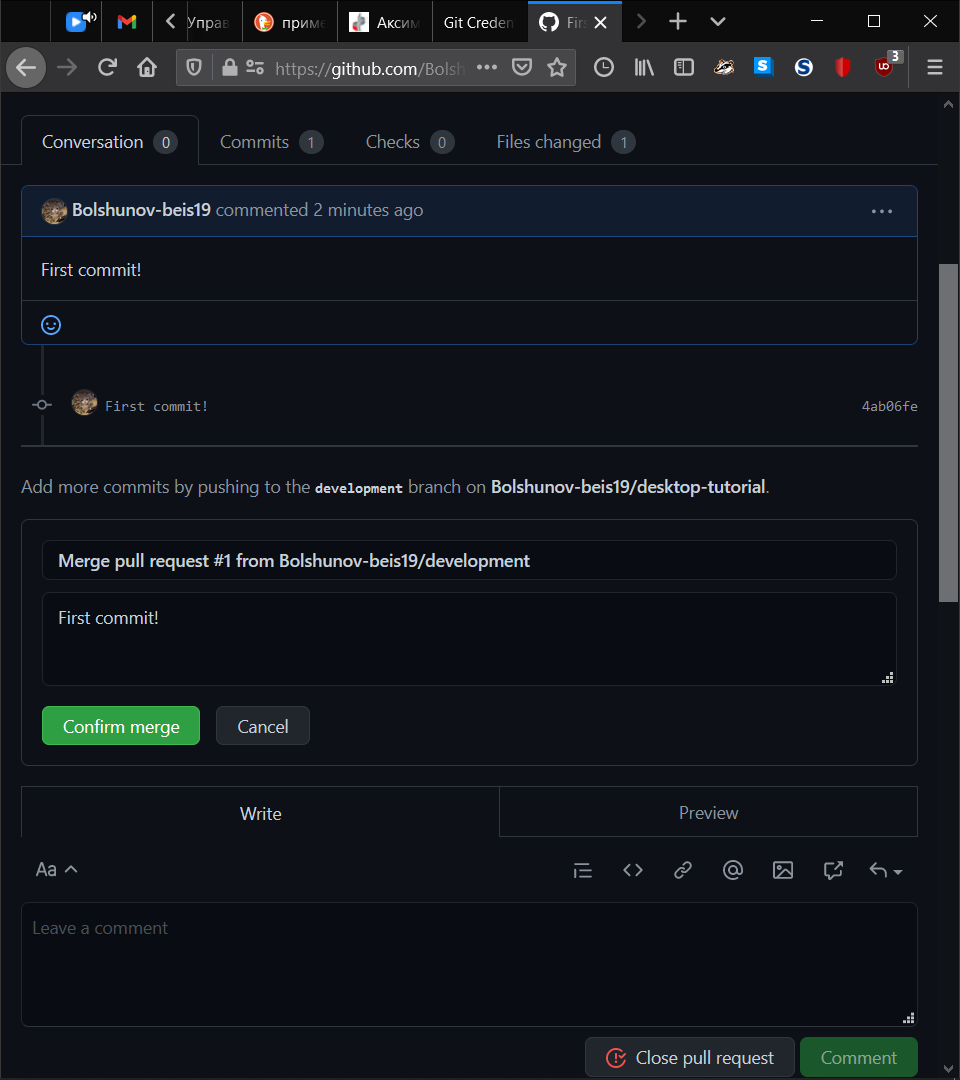


Рисунок . – Подтверждение слияния

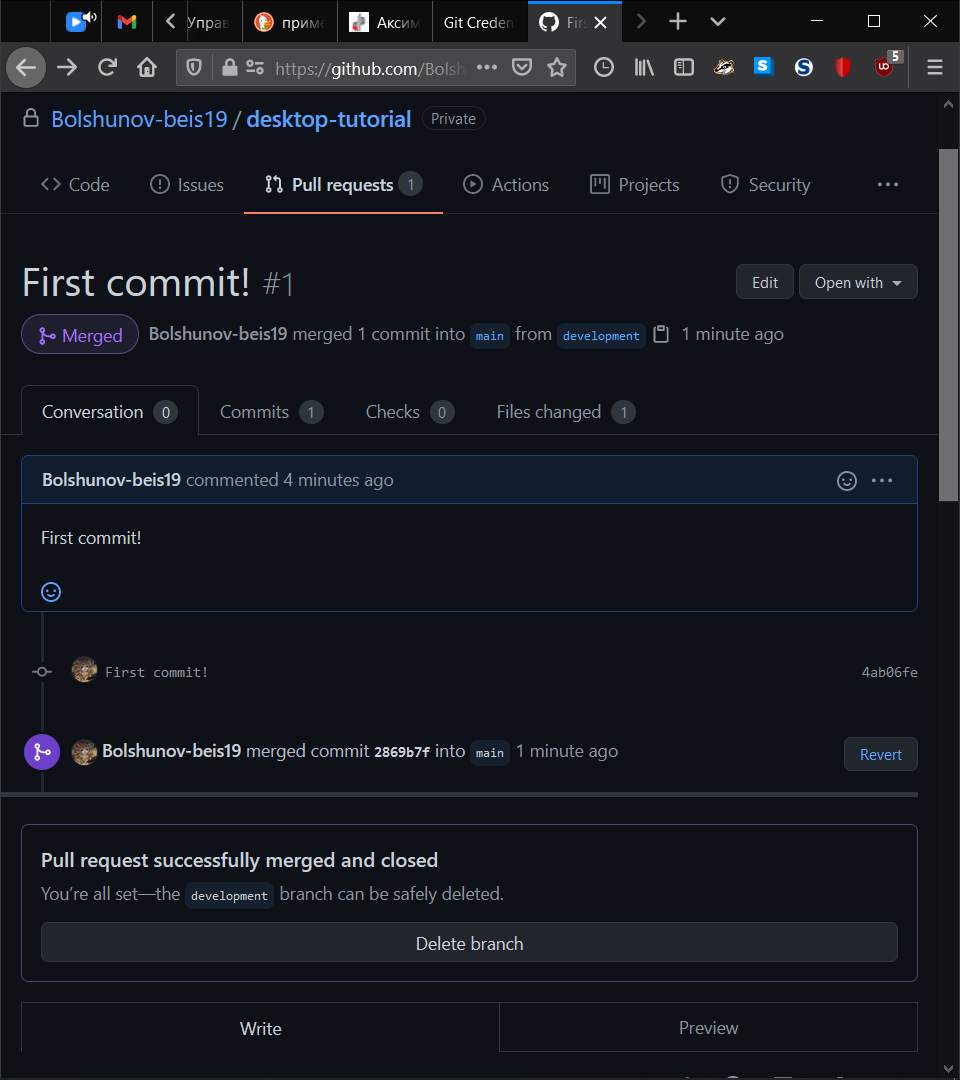


Рисунок . – Результат слияния

При переходе в репозиторий видно, что создан приватный репозиторий, в которой только что была слита ветка development.

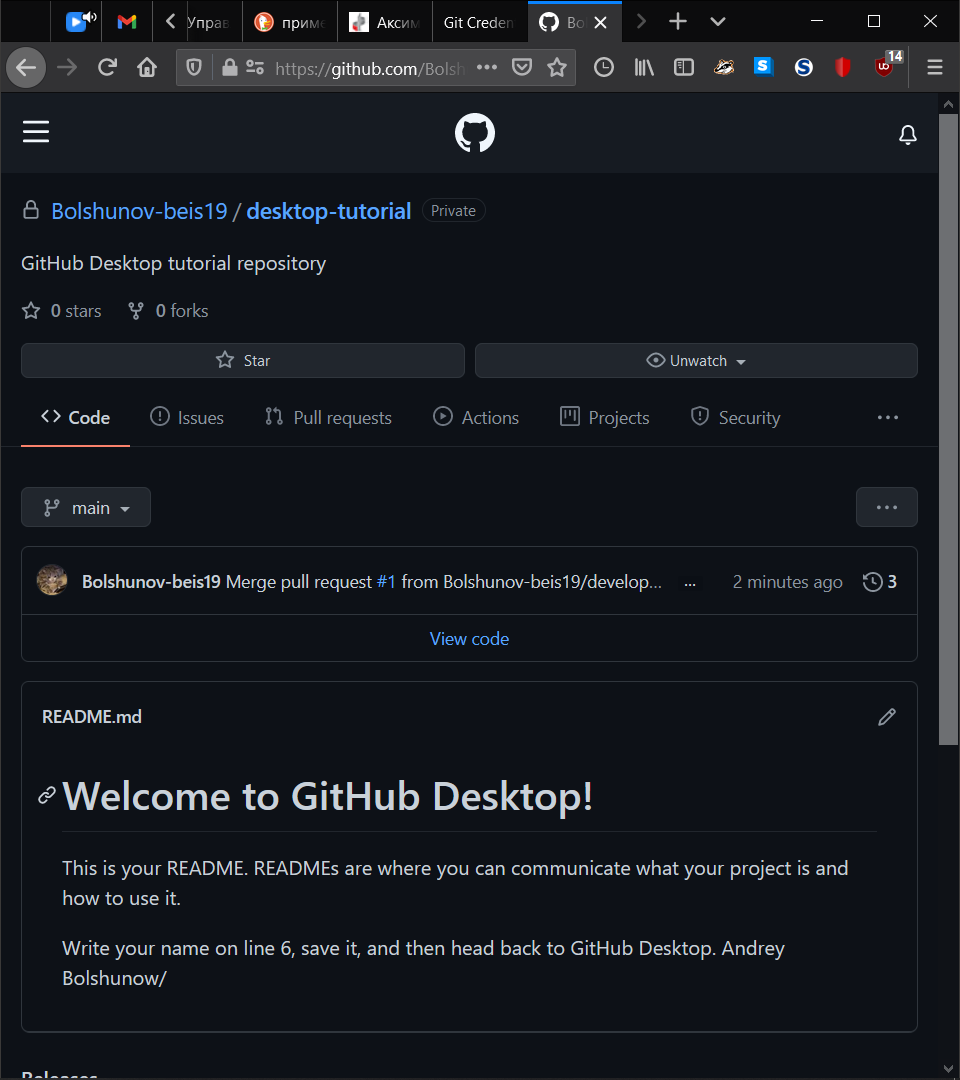


Рисунок . – Внешний вид репозитория после всех изменений

# Лабораторная работа № 3. Разработка программного обеспечения с применением системы контроля версий (VCS)

Задание 1. Установить TortoiseSVN.

TortoiseSVN – бесплатный клиент для Subversion, выполненный как расширение оболочки Windows.

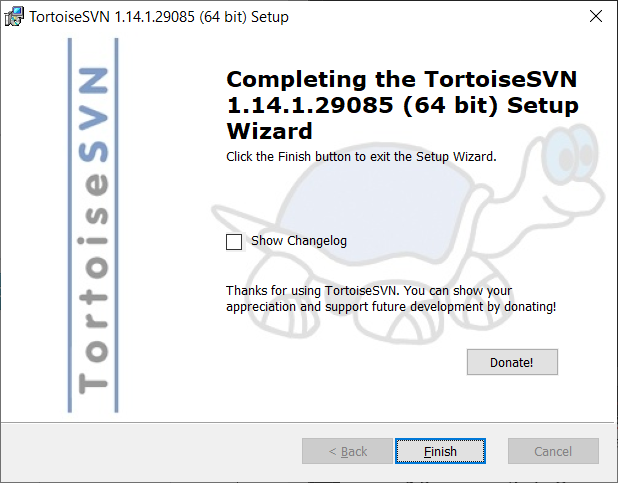


Рисунок . – Успешная установка TortoiseSVN

Задание 2. Установить VisualSVN.

VisualSVN представляет собой плагин, интегрирующийся в среду разработки Microsoft Visual Studio и делающий работу программиста проще.



Рисунок . – Успешная установка VisualSVN

Задание 3. Создать проект и репозиторий.

Создан новый проект Windows Forms «Web Browser».

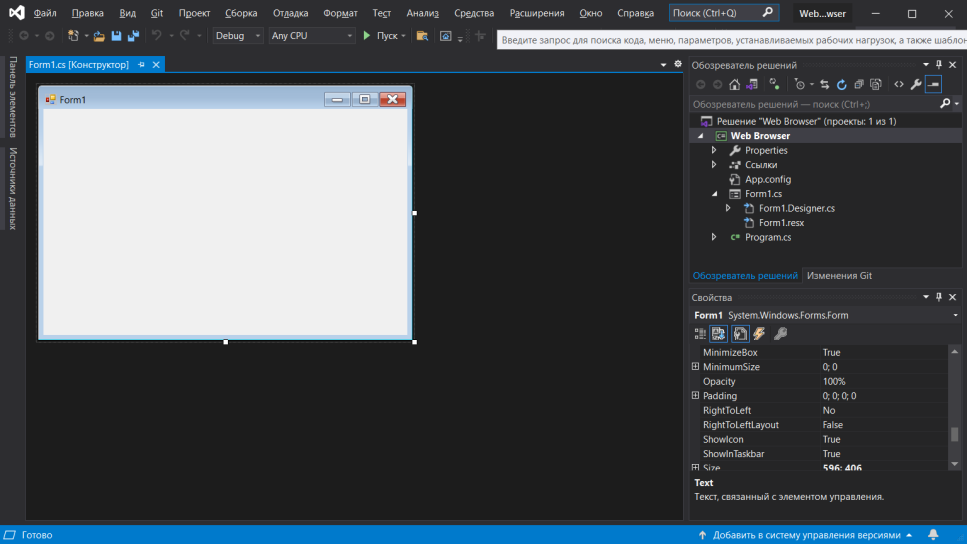


Рисунок . – Новый проект Web Browser

Далее создан новый репозиторий в папке «Repository/Web Browser».

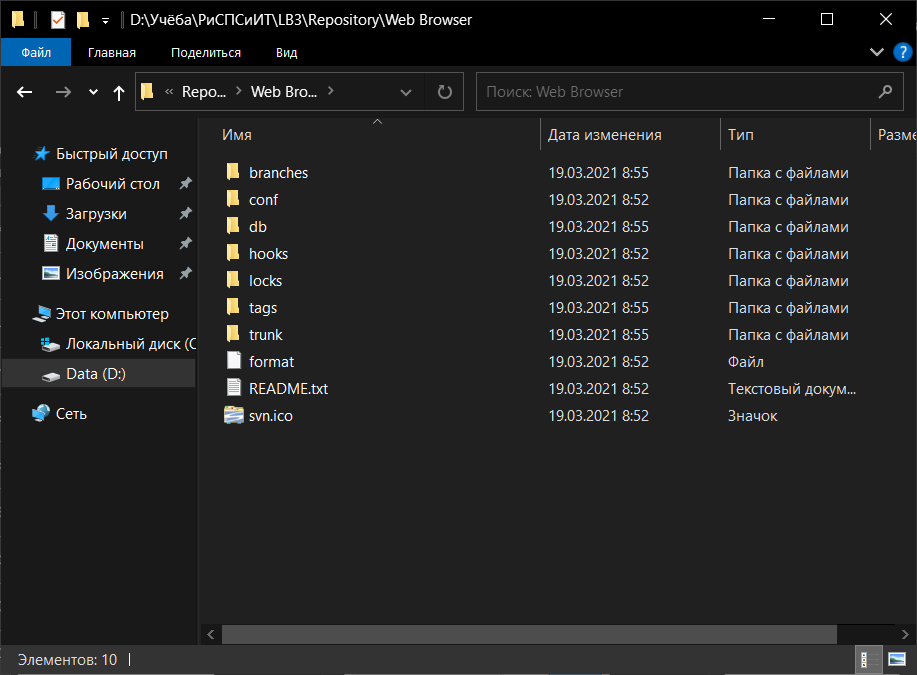


Рисунок . – Создан новый репозиторий

Далее сделан импорт файлов в систему контроля версий и выполнен check-out для извлечения документа из хранилища и создание рабочей копии.



Рисунок . – Импорт файлов проекта в репозиторий

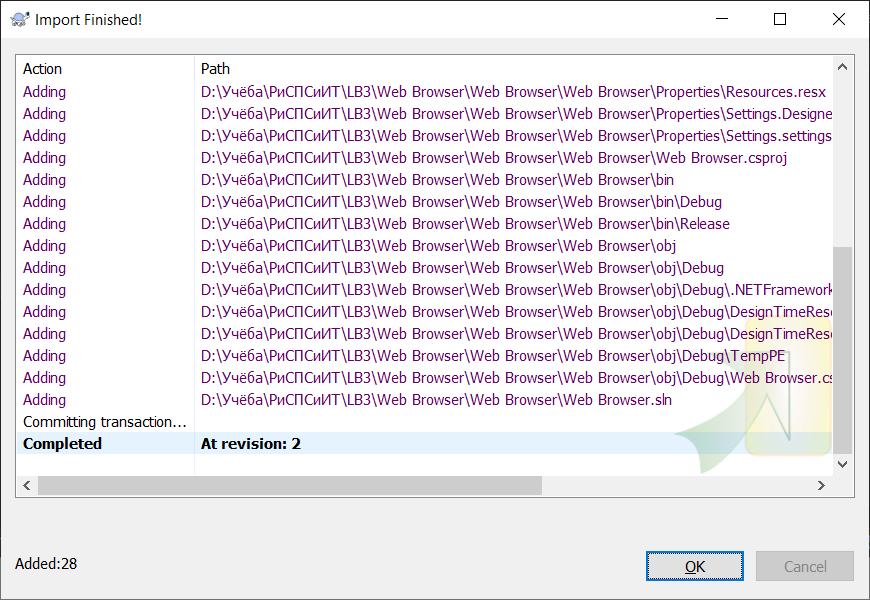


Рисунок . – Результат импорта файлов в репозиторий

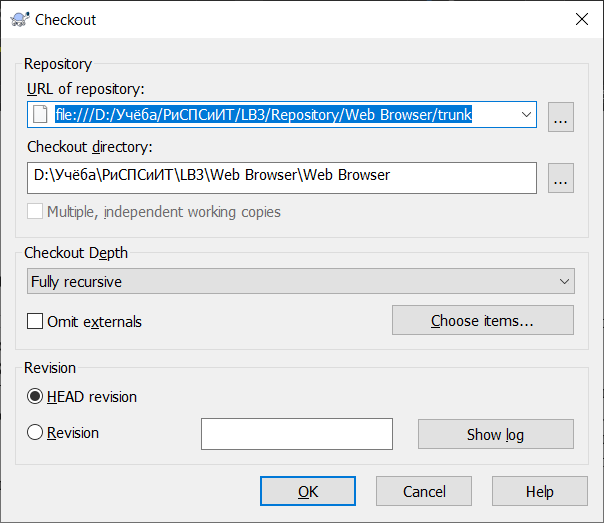


Рисунок . – Настройки для совершения checkout

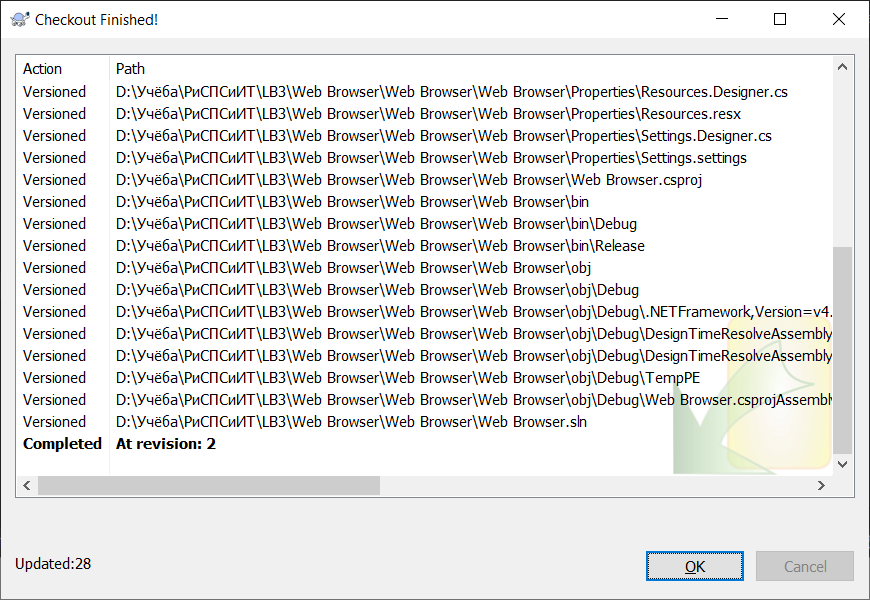


Рисунок . – Результат проведения checkout

В нем сообщается, что операция завершена, а файлы проекта были помещены под версионный контроль.

Нужно, чтобы среда MS Visual Studio знала, что данный проект находится под версионным контролем. В окне уже указан путь к корневой папке проекта, что соответствует расположению рабочей копии.

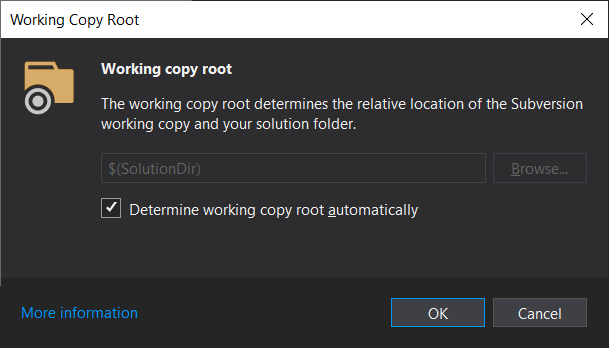


Рисунок . – Установка пути рабочей копии

Далее сделан первый commit для фиксации изменений. После этого можно отследить все изменения, внесенные с момента начала работы над проектом. В верхнем окне написано описание commit: «Первый commit перед началом работы». Внизу показан список файлов, которые были изменены и которые будут отправлены в репозиторий с присвоением нового номера ревизии.

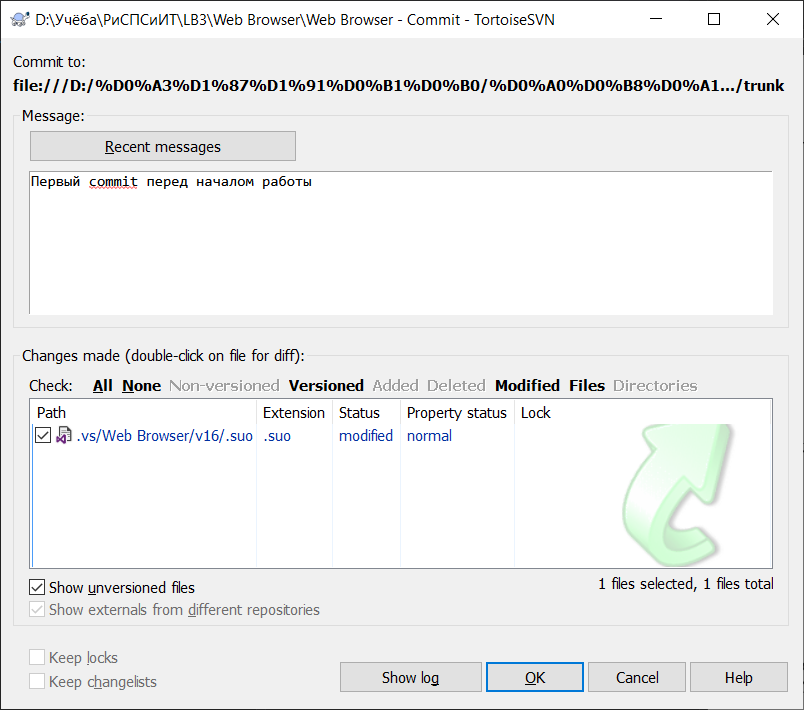


Рисунок . – Окно совершения commit

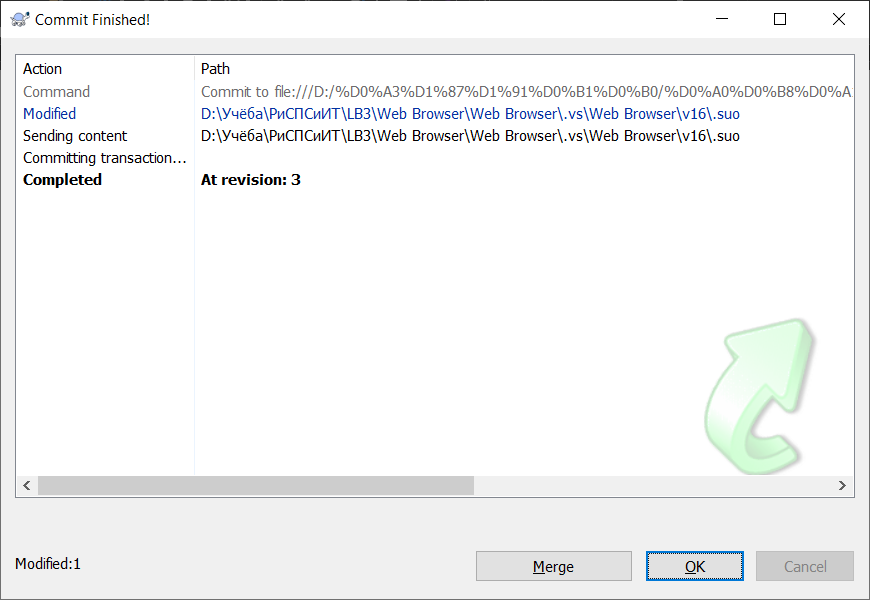


Рисунок . – Завершение commit

Здесь видно, что проект получил новую ревизию (в данном случае – 3) и количество модифицированных файлов – в данном случае такой файл один.

Далее изменены надпись и имя формы в проекте и сделан соответствующий commit.

В открытом окне логов теперь можно увидеть кто сделал определенный commit, время, когда это произошло, какова цель этого действия и так далее.

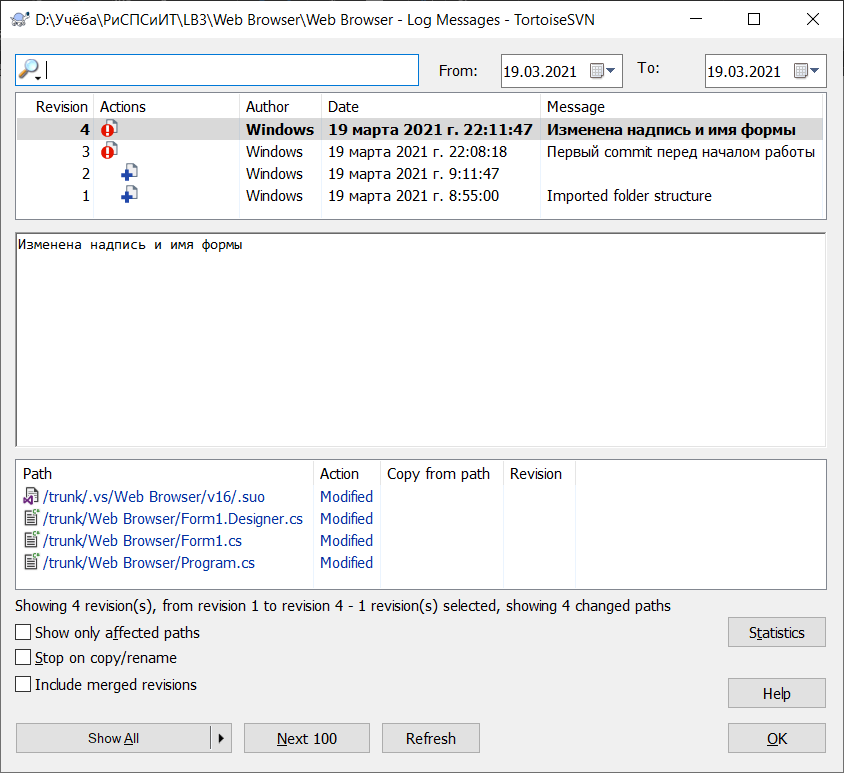


Рисунок . – История системы контроля версий

В проект добавлена и заполнена строка меню. И снова сделан commit «Добавили верхнее меню».

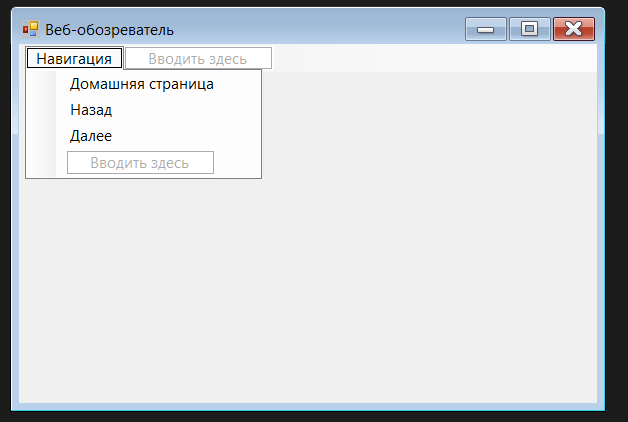


Рисунок . – Новое меню

В проекте добавил на форму компоненты ComboBox, Button и WebBrowser и настроил их согласно заданию. После этого создан соответствующий commit.

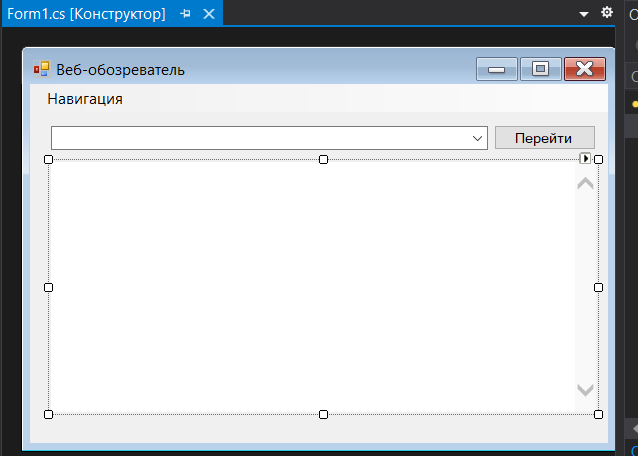


Рисунок . – Добавление компонентов ComboBox, Button и WebBrowser

Затем эти компоненты удалены и создан соответствующий commit.

Далее через лог системы контроля версий восстановлен commit, который был сделан после добавления компонента WebBrowser. Компоненты вернулись на прежнее место.



Рисунок . – Успешный откат версии

Далее добавил код, который выполняется по нажатию кнопки в соответствии с заданием. Опять же создан commit.

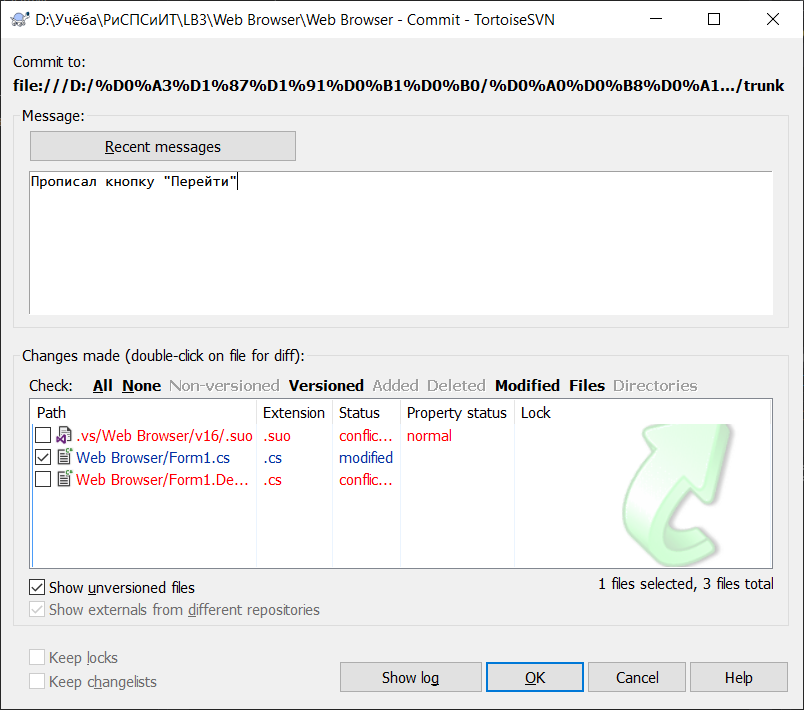


Рисунок . – Очередной commit

Затем был создан обработчик загрузки формы. В результате элемент управления WebBrowser отобразит домашнюю страницу по умолчанию и будет присвоено начальное значение элементу управления ComboBox. Снова сделан соответствующий commit.

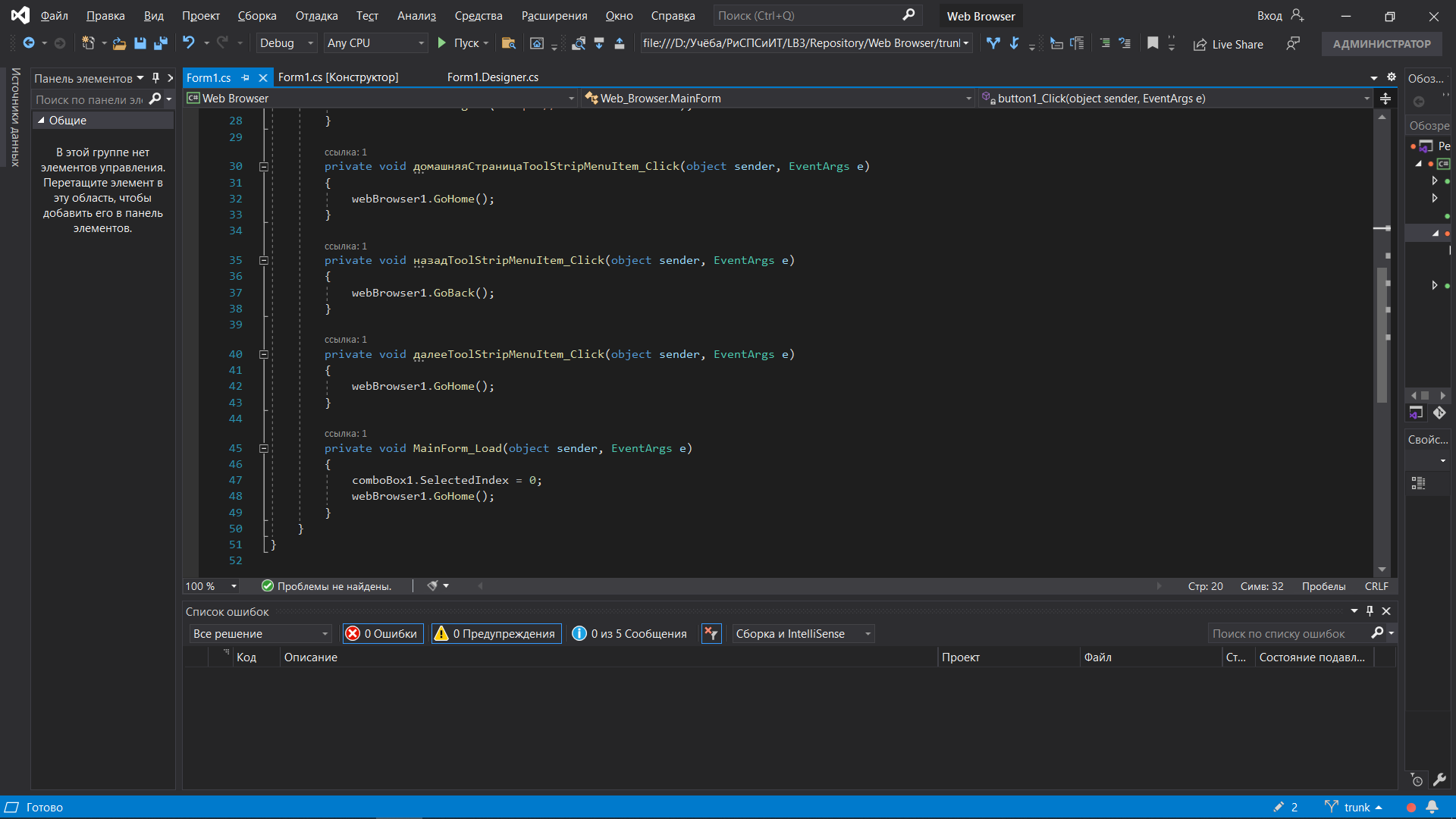


Рисунок . – Добавление кода для обработки

Далее в соответствии с заданием добавлен код и сделан commit. После этого программа перестала работать, поэтому нужно удалить «this.Close();» в конце кода. После этого программа работает исправно. Сам сайт не открывается из-за отсутствия разрешений брандмауэра, это не ошибка в программе.

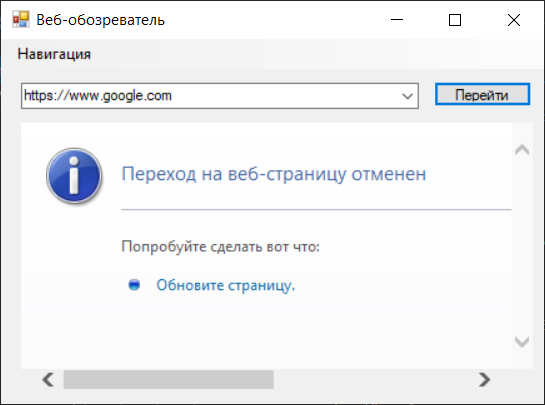
 

Рисунок . – Исправная работа программы

# Лабораторная работа № 4. Оценка характеристик программ на основе лексического анализа

Задание 1. Анализ задачи вычисления функции G(x)=sin2(π\*x)\*cos (π\*x) на интервале a<x<=b.

Для начала стоит представить полный текст программы с пронумерованными строками. На скриншоте из Visual Studio хорошо видна нумерация всех строк, поэтому строить для этого таблицу из примера не имеет смысла.

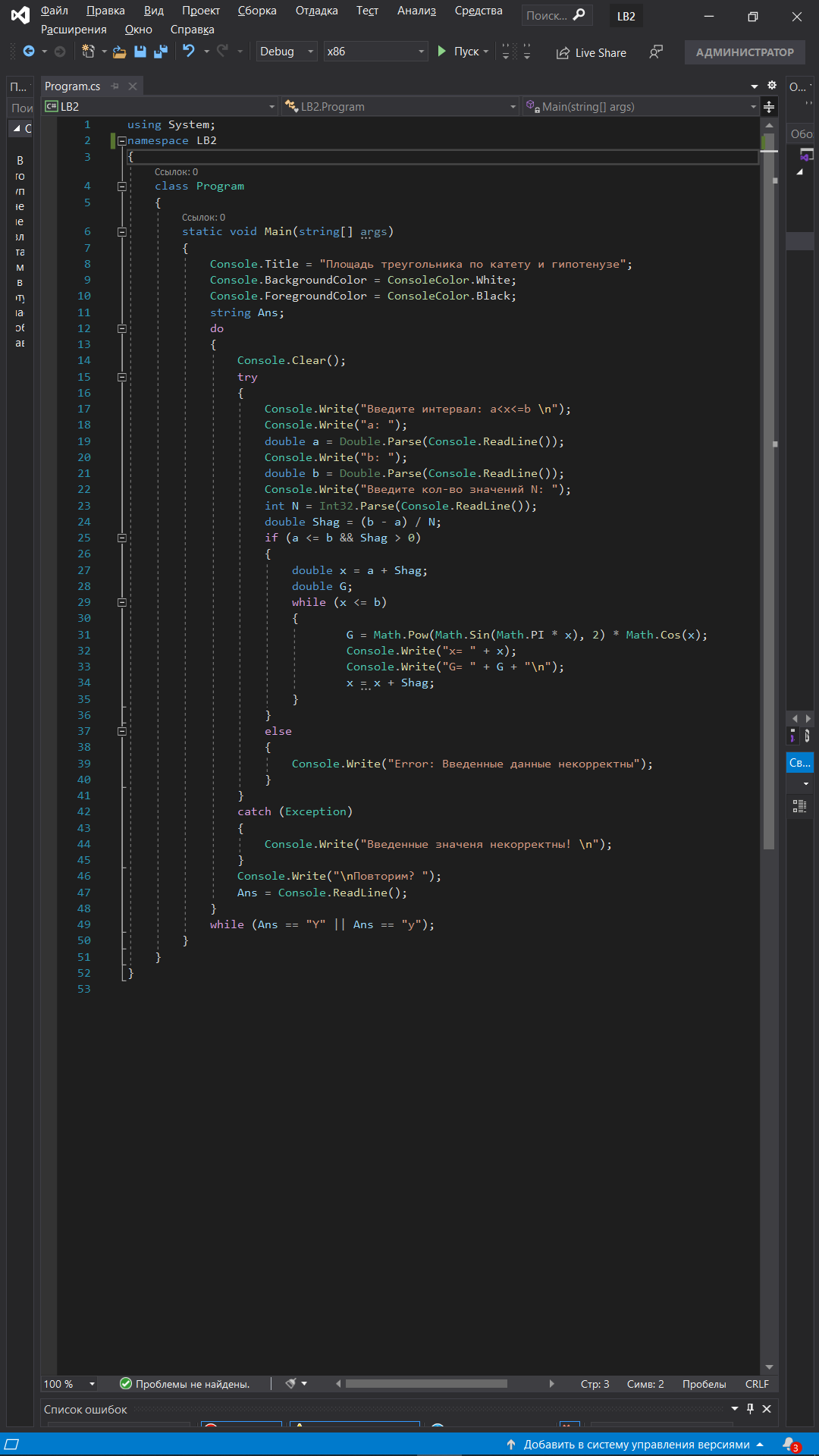


Рисунок . – Текст программы

Далее все операторы записаны в таблицу с указанием номеров строк и количества входов.

Таблица . – Словарь операторов и операций программы

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Операторы, операции | Номера строк | Количество повторений |
| 1 | using...; | 1 | 1 |
| 2 | namespace ... | 2 | 1 |
| 3 | class ... | 4 | 1 |
| 4 | static void... | 6 | 1 |
| 5 | double... | 19, 21, 24, 27, 28 | 5 |
| 6 | string... | 11 | 1 |
| 7 | int… | 23 | 1 |
| 8 | Console.Title… | 8 | 1 |
| 9 | Console.BackgroundColor… | 9 | 1 |
| 10 | Console.ForegroundColor… | 10 | 1 |
| 11 | Console.Clear(); | 14 | 1 |
| 12 | do...while() | 12-49 | 1 |
| 13 | try…catch | 15-45 | 1 |
| 14 | Console. Write() | 17, 18, 20, 22, 32, 33, 39, 44, 46 | 9 |
| 15 | ....Parse() | 19, 21, 23 | 3 |
| 16 | Console. ReadLineQ | 19, 21, 23, 47 | 4 |
| 17 | if ()...else... | 25-40 | 1 |
| 18 | while ()… | 29-35 | 1 |
| 19 | Math.Pow() | 31 | 1 |
| 20 | Math.Sin() | 31 | 1 |
| 21 | Math.Cos() | 31 | 1 |
| 22 | ; | 1, 8, 9, 10, 11, 14, 17, 18, 19, 20, 21, 22, 23, 24, 27, 28, 31, 32, 33, 34, 39, 44, 46, 47, 49 | 25 |
| 23 | , | 31 | 1 |
| 24 | \* | 31 | 1 |
| 25 | = | 8, 9, 10, 19, 21, 23, 24, 27, 31, 34, 47 | 11 |
| 26 | + | 27, 32, 33, 34 | 4 |
| 27 | – | 24 | 1 |
| 28 | / | 24 | 1 |
| 29 | <= | 25, 29 | 2 |
| 30 | > | 25 | 1 |
| 31 | = = | 49, 49 | 2 |
| 32 | && | 25 | 1 |
| 33 | {} | 3(52), 5(51), 7(50), 13(48), 16(41), 26(36), 30(35), 38(40), 43(45) | 9 |
| 34 | () | 6, 14, 17, 18, 19, 19, 20, 21, 21, 22, 23,23 24, 25, 29, 31, 31, 31, 32, 33, 39, 42, 44, 46, 47, 49 | 25 |
| 35 | || | 49 | 1 |
| 36 | “ “ | 8, 17, 18, 20, 22, 32, 33, 39, 44, 46, 49, 49 | 12 |
| 37  n1 = 37 | . | 8, 9, 10, 14, 17, 18, 19, 19, 20, 21, 21, 22, 23, 23, 31, 31, 31, 32, 33, 39, 44, 46, 47 | 24 |
| Всего | | | N 1  = 159 |

Далее все операнды записаны в таблицу с указанием номеров строк и количества входов.

Таблица . – Словарь операндов программы

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Операнды | Номера строк | Количество повторений |
| 1 | System | 1 | 1 |
| 2 | LB2 | 2 | 1 |
| 3 | Program | 4 | 1 |
| 4 | Main | 6 | 1 |
| 5 | Ans | 11, 47, 49, 49 | 4 |
|  | a | 19, 24, 25, 27 | 4 |
|  | b | 21, 24, 25, 29 | 4 |
|  | N | 23, 24 | 2 |
|  | Shag | 24, 25, 27, 34 | 4 |
|  | x | 27, 29, 31, 31, 32, 34, 34 | 7 |
|  | G | 28, 31, 33 | 3 |
| 8 | "Введите интервал: a<x<=b \n" | 17 | 1 |
| 9 | "a =: " | 18 | 1 |
|  | "b =: " | 20 | 1 |
| 10 | "Введите кол-во значений N: " | 22 | 1 |
| 11 | "x= " | 32 | 1 |
| 12 | "G= " | 33 | 1 |
|  | "Error: Введенные данные некорректны" | 39 | 1 |
|  | "Введенные значения некорректны!" | 44 | 1 |
| 13 | "Повторим?" | 46 | 1 |
| 14 | “Y” | 49 | 1 |
| 15  n2 = 15 | “y” | 49 | 1 |
| Всего | | | N 2  = 43 |

Также составлена таблица входных и выходных переменных.

Таблица . – Входные и выходные переменные программы

|  |  |
| --- | --- |
| Входные переменные | Выходные переменные |
| Ans | "Введите интервал: a<x<=b" |
| a | "a =: " |
| b | "b =: " |
| N | "Введите кол-во значений N: " |
|  | "x= " |
|  | "G= " |
|  | "Error: Введенные данные некорректны" |
|  | "Введенные значения некорректны!" |
|  | "Повторим?" |

Задание 2. Оценка характеристик программы.

Используя сформированные таблицы с необходимыми парамет­рами для расчета и применяя соотношения Холстеда, вычислены ха­рактеристики рассматриваемой программы:

* словарь программы: n = n1 + n2 = 37 + 15 = 52;
* длина реализации: N = N1 + N2 = 159 + 43 = 202;
* длина программы: N ~ = n1\* log2 n1+ n2\* log2n2 *=* 37\* log237+15\* log215=37\*5,2+15\*3,9=250,9;
* объем программы в битах: V = (N1+ N2) \* log2 (n1 + n2) = (159 + 43) \* log2(37 + 15) = 202 \* log252 = 202\*5,7 = 1151,5;
* потенциальный объем программы: V\* *=* (n2\*+ 2) \*log2(n2 + 2) = (13 + 2) \* log2(15 + 2) = 15 \*4,1 = 16,3;
* уровень программы: L = V\*/ V= 16,3 / 1151,5 = 0,014;
* уровень языка: λ = L\* V\* = 0,014 \* 16,3 = 0,23;
* интеллектуальное содержание программы: I= L \*V =0,014 \* 1151,5 = 16,121;
* работа по программированию: E=V / L = 1151,5 / 0,014 = 82250.

Все результаты расчетов метрик Холстеда сведены в таблицу 4.4.

Таблица 4.4 – результаты расчетов метрик Холстеда

|  |  |  |
| --- | --- | --- |
| Наименование характеристики | Обозначение и формулы для вычислений | Значение |
| Число простых операторов и операций | n1 | 37 |
| Число простых (уникальных) операндов | n2 | 15 |
| Общее число всех операторов и операций | N1 | 159 |
| Общее число всех операндов | N2 | 43 |
| Число входных и выходных переменных (параметров) | n2\* | 13 |
| Словарь программы | n = n1 + n2 | 52 |
| Длина реализации программы | N = N1 + N2 | 202 |
| Объем программы (в битах) | V = (N2+ N1) \* log2 (n1 + n2) | 1151,5 |
| Потенциальный объем программы | V\* *=* (n2\*+ 2) \*log2(n2 + 2) | 16,3 |
| Уровень реализации программы | λ = L\* V\* | 0,23 |
| Работа по программированию | E=V / L | 82250 |

Вывод: Уровень исследуемой программы весьма низкий, так как потен­циальный объем программы в значительной степени меньше ее ре­ального объема.

# Лабораторная работа № 4. Оценка характеристик программ на основе лексического анализа

Задание 1. Написать программу, которая вводит текст, состоящий из нескольких предложений, и выводит каждое слово в обратном порядке.

Схема алгоритма. В соответствии с заданием разработан алгоритм, схема которого отражена блок-схемой на рисунке 5.1.

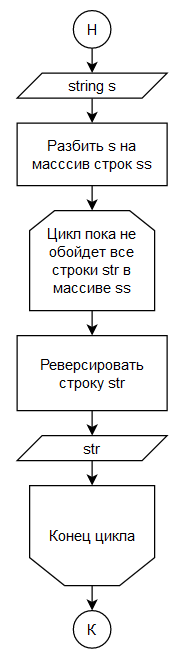


Рисунок 5.1 – Блок-схема алгоритма к лабораторной работе № 5

Разработка программы. Разработка программы проводилась на языке высокого уровня C# [1–2]. Список идентификаторов для программы, разрабатываемой в соответствии с алгоритмом, отраженном на рисунке 5.1, представлен в таблице 5.1.

Таблица 5.1 – Список идентификаторов для задания лаб. раб. № 5

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование переменной и ее смысл | Обозначение в алгоритме | Обозначение в программе |
| 1 | Начальное предложение | s | s |
| 2 | Массив слов предложения | ss | ss |

Текст программы представлен на листинге 5.1

Листинг 5.1– Текст программы на языке C# для задания к лаб. раб. №5

using System;

using System.Linq;

namespace LB5.\_1

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите текст:");

string s = Console.ReadLine();

string[] ss = s.Split(new char[] { ' ', ',', '.', ';', '!', '?' });

foreach (string str in ss)

{

Console.WriteLine(str.ToCharArray().Reverse().ToArray());

}

Console.ReadKey();

}

}

}

Проведенное тестирование показало корректную работу программы на предложенных тестовых наборах входных данных, а выявленные ошибки были исправлены путем введения в программу дополнительных проверок.



Рисунок 5.2 – Результаты работы программы

Задание 2. Дан текстовый файл f и строка s. Переписать в новый файл g все строки файла f, содержащие значения переменной s в качестве подстроки.

Схема алгоритма. В соответствии с заданием разработан алгоритм, схема которого отражена блок-схемой на рисунке 5.3.

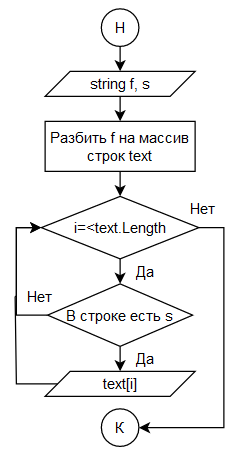


Рисунок 5.3 – Блок-схема алгоритма к лабораторной работе № 5

Разработка программы. Разработка программы проводилась на языке высокого уровня C# [1–2]. Список идентификаторов для программы, разрабатываемой в соответствии с алгоритмом, отраженном на рисунке 5.3, представлен в таблице 5.2.

Таблица 5.2 – Список идентификаторов для задания лаб. раб. № 5

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование переменной и ее смысл | Обозначение в алгоритме | Обозначение в программе |
| 1 | Предложение из файла | f | f |
| 2 | Строка для поиска | s | s |
| 3 | Массив строк из предложений | text | text |

Текст программы представлен на листинге 5.2

Листинг 5.2– Текст программы на языке C# для задания к лаб. раб. №5

using System;

using System.IO;

using System.Text.RegularExpressions;

namespace LB5.\_3

{

class Program

{

static void Main(string[] args)

{

string f = File.ReadAllText(@"D:\Учёба\РиСПСиИТ\LB5\f.txt");

Console.WriteLine("Введите строку: ");

string s = Console.ReadLine();

string[] text = f.Split(new char[] { ' ', ',', '.', ';', '!', '?' });

FileStream fileg = new FileStream(@"D:\Учёба\РиСПСиИТ\LB5\g.txt", FileMode.Create);

StreamWriter writer = new StreamWriter(fileg);

for (int i = 0; i < text.Length; i++)

{

if (Regex.IsMatch(text[i], s) == true)

{

writer.WriteLine(text[i]);

Console.WriteLine(text[i]);

}

}

writer.Close();

Console.ReadKey();

}

}

}

Проведенное тестирование показало корректную работу программы на предложенных тестовых наборах входных данных, а выявленные ошибки были исправлены путем введения в программу дополнительных проверок.

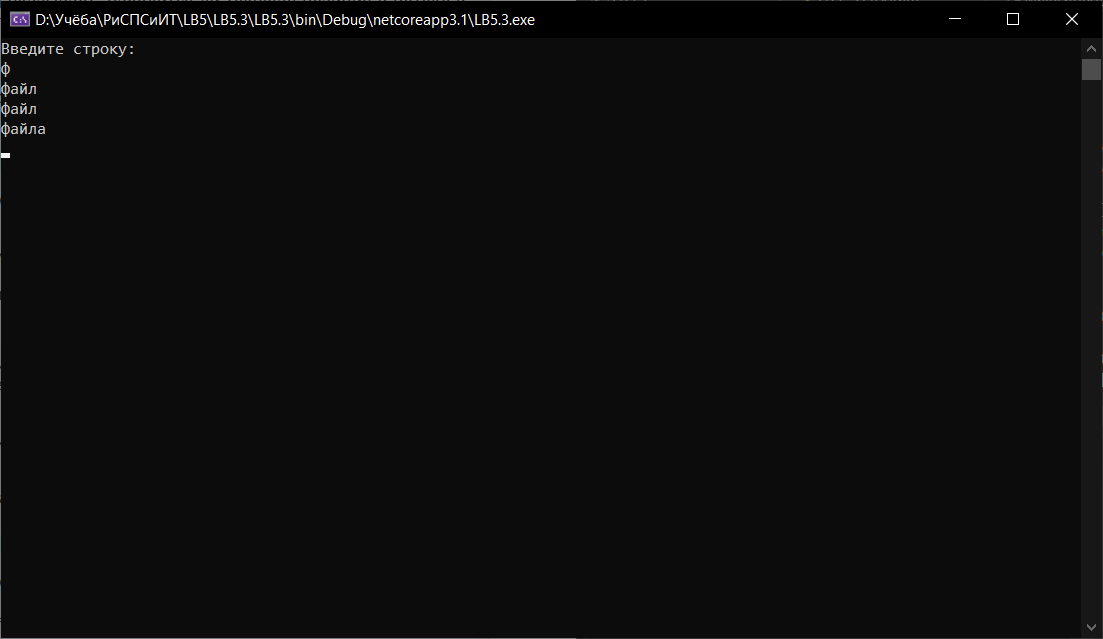
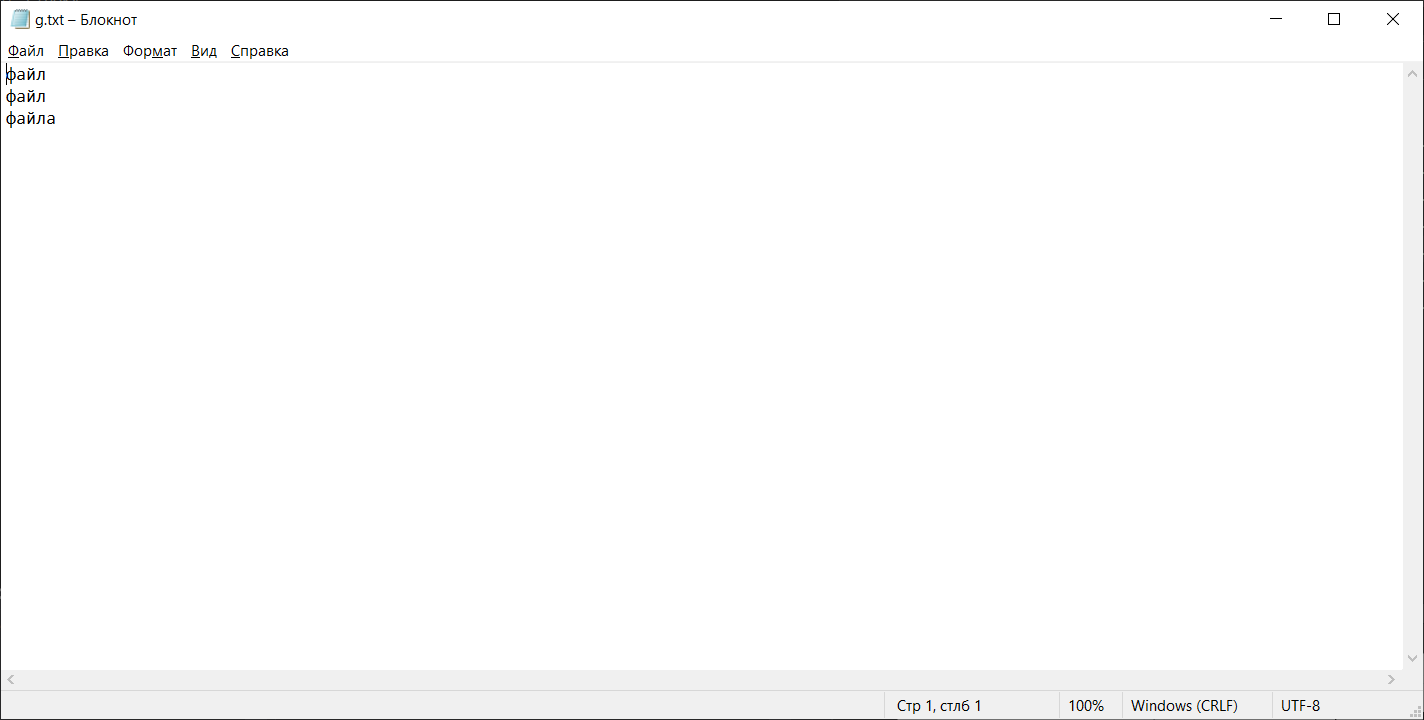
 

Рисунок 5.4 – Результаты работы программы

# Список использованных источников

1 Тюкачев Н.А., Хлебостроев В.Г. C#. Основы программирования: учебное пособие. Уровень образования: бакалавриат, специалитет, магистратура // – 3-е изд., стер. – СПб.: "Лань". 2018, 272 с. URL: <https://e.lanbook.com/book/104962>

2 Залогова, Л. А. Основы объектно-ориентированного программирования на базе языка С# : учебное пособие / Л. А. Залогова. – 2-е изд., стер. – Санкт-Петербург : Лань, 2020. – 192 с. – ISBN 978-5-8114-4757-2. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/126160> – Режим доступа: для авторизованных пользователей.