

Министерство образования Российской Федерации
Московский государственный институт электронной техники
(технический университет)
Кафедра Информатики и программного обеспечения
вычислительных систем

УТВЕРЖДАЮ
Зав. Кафедрой ИПОВС,
д.т.н., проф. Гагарина Л.Г.
«__»_____2024 г.

ЛАБОРАТОРНАЯ РАБОТА №3

Выбор платформы и декомпозиция проекта. Основы использования системы
контроля версий Git.

Руководитель, к.т.н., доцент_____

Исполнитель, студентс гр. ПИН-36 Волков Р.С.

Горбань В.Г.
Кашпаров Г.И.
Котляров Д.Н.
Хаснаш А.А.

МОСКВА, 2024

1. Перечень технологий для реализации программы автоматизации автосервиса

Для реализации программы автоматизации работы автосервиса были выбраны следующие технологии:

1. **Язык программирования:** C++
2. **Система управления базами данных (СУБД):** PostgreSQL
3. **Среда разработки (IDE):** Visual Studio или CLion
4. **Фреймворк для разработки графического интерфейса:** Qt
5. **Библиотеки для взаимодействия с базой данных:** libpqxx (C++ клиентская библиотека для PostgreSQL)
6. **Система контроля версий:** Git
7. **Средства тестирования:** Google Test или Catch2 для юнит-тестов

2. Обоснование выбора технологий и удобство их использования

1. **C++:**
 - **Высокая производительность.** Автосервис может обрабатывать множество данных и запросов, связанных с клиентами, автомобилями и услугами, что требует быстрой обработки.
 - **Широкие возможности управления памятью.** C++ позволяет более эффективно использовать память и ресурсы системы.
 - **Большая экосистема библиотек.** C++ поддерживает множество библиотек, в том числе для работы с базами данных, создания пользовательских интерфейсов и многопоточного программирования.
2. **PostgreSQL:**
 - **Надежность и масштабируемость.** PostgreSQL — одна из самых стабильных и надежных СУБД с поддержкой транзакций и целостности данных, что важно для ведения учета в автосервисе.
 - **Широкие возможности SQL.** PostgreSQL поддерживает сложные запросы, индексы и внешние ключи, что удобно для работы с большим объемом данных.
 - **Поддержка расширений.** PostgreSQL имеет широкие возможности по расширению функциональности через плагины и модули.
3. **Qt:**
 - **Многоплатформенность.** Qt позволяет разрабатывать программы для Windows, Linux и macOS с минимальными изменениями в коде.
 - **Интуитивный интерфейс.** Qt предоставляет возможности для создания удобных и привлекательных пользовательских интерфейсов, что важно для конечных пользователей.
 - **Мощные библиотеки для работы с графикой, файлами и сетью.**
4. **libpqxx:**
 - **Прямое взаимодействие с PostgreSQL.** libpqxx предоставляет интерфейс для работы с PostgreSQL на C++ без лишней сложности.
 - **Поддержка всех функций PostgreSQL.** Библиотека поддерживает работу с транзакциями, сложными запросами и подготовленными выражениями.
5. **Git:**
 - **Управление версионностью.** Для командной разработки и отслеживания изменений в коде необходима система контроля версий. Git позволяет эффективно управлять историей изменений, что повышает продуктивность команды.
6. **Google Test/Catch2:**

- **Юнит-тестирование.** Необходимо для обеспечения качества кода и своевременного выявления ошибок. Google Test и Catch2 — популярные библиотеки для создания модульных тестов на C++.

3. Декомпозиция разрабатываемой системы и описание модулей

Система будет состоять из следующих основных модулей:

1. **Модуль управления клиентами:**
 - Регистрация и хранение информации о клиентах.
 - Редактирование и удаление данных клиентов.
 - Поиск клиентов по параметрам (имя, номер телефона, история заказов).
2. **Модуль управления автомобилями:**
 - Ведение базы данных автомобилей (марка, модель, год выпуска, VIN-код).
 - Привязка автомобилей к клиентам.
 - Учет истории ремонта и технического обслуживания автомобилей.
3. **Модуль учета услуг и заказов:**
 - Создание и управление заказами на услуги.
 - Управление услугами автосервиса (ремонт, диагностика, замена запчастей).
 - Хранение информации о выполненных работах, сроках и стоимости услуг.
4. **Модуль управления сотрудниками:**
 - Управление информацией о сотрудниках (имя, должность, контакты).
 - Привязка заказов и услуг к сотрудникам, ответственным за выполнение.
5. **Модуль складского учета:**
 - Учет запасных частей и расходных материалов.
 - Автоматическое списание со склада при выполнении заказа.
 - Уведомления о необходимости пополнения запасов.
6. **Модуль отчетности:**
 - Генерация отчетов по заказам, услугам, выручке.
 - Статистика по клиентам, автомобилям и сотрудникам.
 - Возможность экспорта отчетов в формате PDF или Excel.
7. **Модуль безопасности и управления доступом:**
 - Авторизация и аутентификация пользователей.
 - Разграничение прав доступа (администратор, менеджер, механик).
 - Журналирование действий пользователей.

4. Полный перечень задач с указанием их приоритета:

Высокий приоритет:

1. **Настройка среды разработки:**
 - Установка и настройка инструментов: C++, PostgreSQL, Qt, libpqxx, IDE (Visual Studio или CLion), Git.
 - Конфигурация системы контроля версий (создание репозитория, настройка CI/CD).
2. **Разработка базы данных:**
 - **Проектирование структуры базы данных:** создание таблиц для хранения данных клиентов, автомобилей, заказов, услуг, сотрудников, запчастей.
 - **Реализация схемы данных:** использование SQL для создания таблиц, связей между ними и ограничений целостности данных (например, внешние ключи).
3. **Интеграция с PostgreSQL через libpqxx:**
 - **Настройка подключения:** реализация модуля, который устанавливает соединение с базой данных и обрабатывает запросы.

- **Тестирование базовых операций:** выполнение CRUD-операций (создание, чтение, обновление, удаление) для проверки работы с базой данных.
 - 4. **Разработка модуля управления клиентами:**
 - **CRUD-операции для клиентов:** создание, редактирование, удаление и поиск клиентов.
 - **Интерфейс для взаимодействия с клиентами:** удобный UI для работы с данными клиентов.
 - 5. **Разработка модуля управления автомобилями:**
 - **CRUD-операции для автомобилей:** хранение и управление данными о транспортных средствах, включая привязку к клиентам.
 - **Интеграция с заказами:** возможность привязки автомобиля к конкретным заказам и услугам.
 - 6. **Разработка модуля учета заказов и услуг:**
 - **Создание и управление заказами:** функционал для создания новых заказов, назначения услуг, управления статусом заказа (например, "ожидается", "в процессе", "завершен").
 - **Учет стоимости услуг:** расчет и хранение данных о стоимости услуг.
-

Средний приоритет:

1. **Модуль управления сотрудниками:**
 - **CRUD-операции для сотрудников:** создание и редактирование профилей сотрудников (механиков, администраторов).
 - **Привязка сотрудников к заказам:** назначение сотрудников, ответственных за выполнение услуг.
2. **Модуль складского учета:**
 - **Учет запасов:** хранение данных о запчастях, расходных материалах.
 - **Инвентаризация:** списание со склада при выполнении заказа и автоматические уведомления о низких остатках.
 - **Заказ новых запасов:** интерфейс для заказа и управления поступлениями запчастей.
3. **Создание пользовательского интерфейса (UI):**
 - **Основные экраны:** разработка интуитивно понятного графического интерфейса для всех модулей системы.
 - **Навигация:** реализация навигации между разделами (клиенты, автомобили, заказы, склад).
 - **Поиск и фильтрация данных:** удобные функции для поиска клиентов, заказов и автомобилей по ключевым параметрам.

Низкий приоритет:

1. **Модуль отчетности:**
 - Генерация отчетов по заказам, услугам, сотрудникам.
 - Экспорт в PDF/Excel.
 - Статистика и аналитика (выручка, популярные услуги).
2. **Модуль безопасности и управления доступом:**
 - Аутентификация пользователей (логин/пароль).
 - Разграничение прав доступа по ролям (администратор, механик).
 - Логирование действий пользователей (кто, что и когда изменил).
3. **Тестирование и оптимизация:**
 - Юнит-тесты и интеграционные тесты.

- Нагрузочное тестирование для проверки производительности.
 - Оптимизация запросов к базе данных и кода.
4. **Документация:**
- Техническая документация по архитектуре и структуре системы.
 - Руководства для разработчиков и пользователей.
 - FAQ для решения типичных проблем.

Работа с Git

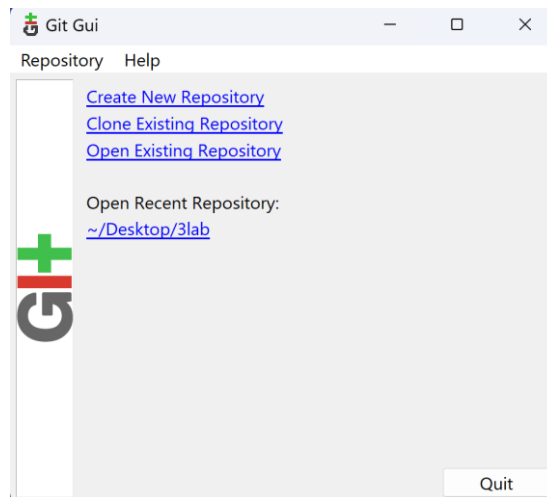


Рисунок 1. Запуск Git Gui.

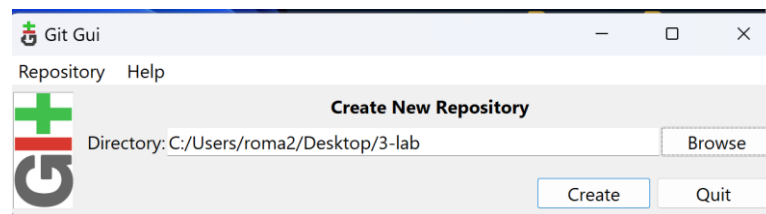


Рисунок 2. Создание нового репозитория для проекта.

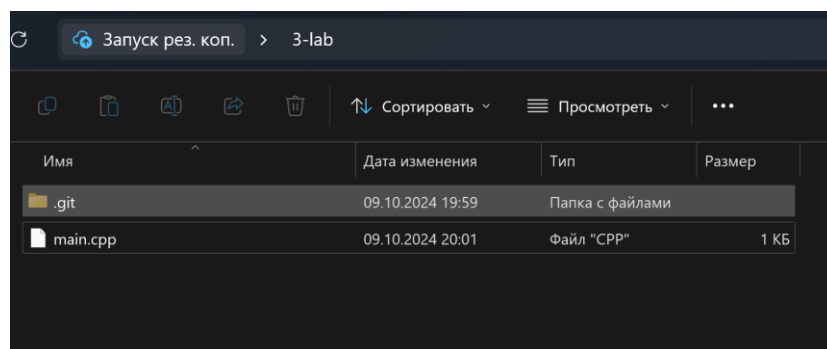


Рисунок 3. Добавление в папку репозитория файла main.cpp.

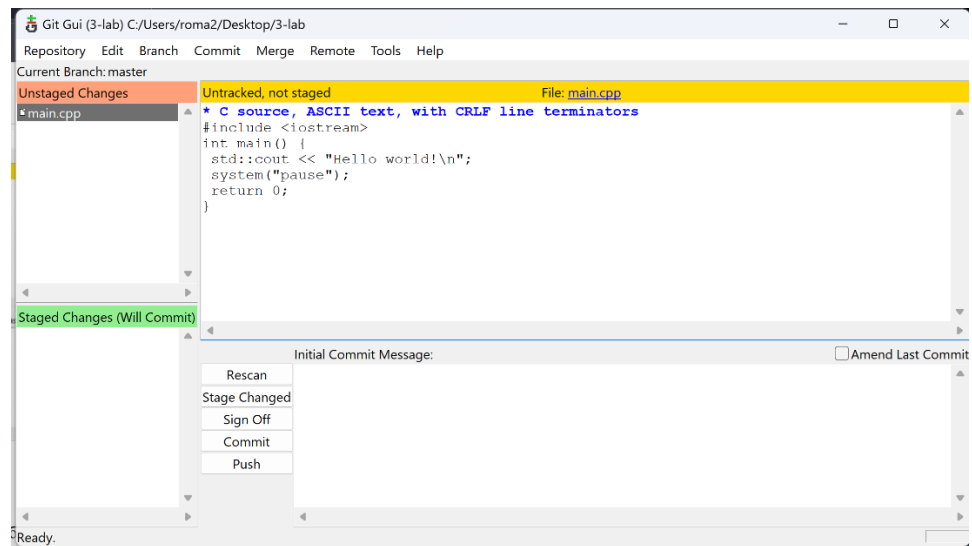


Рисунок 4. Git Gui после нажатия "Rescan".

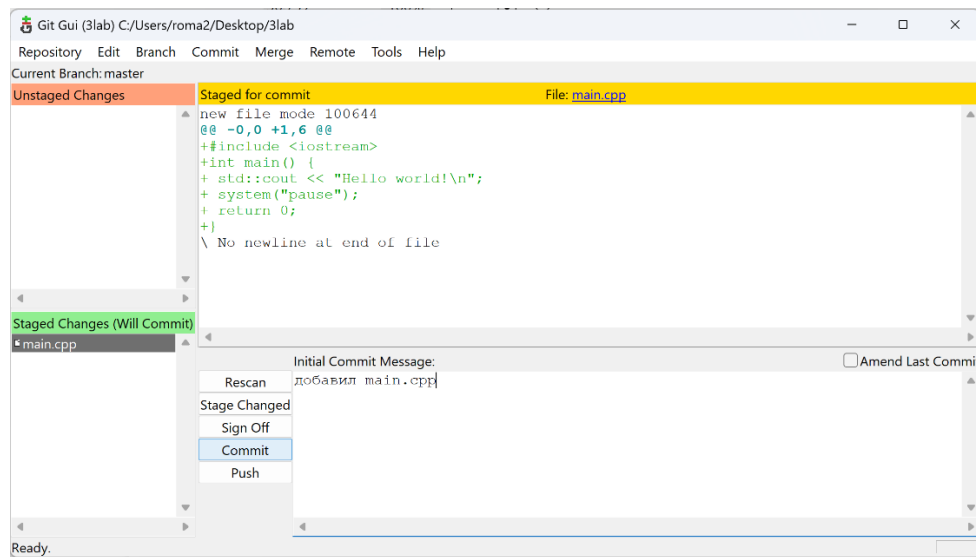


Рисунок 5. Коммит файла main.cpp на ветке master.

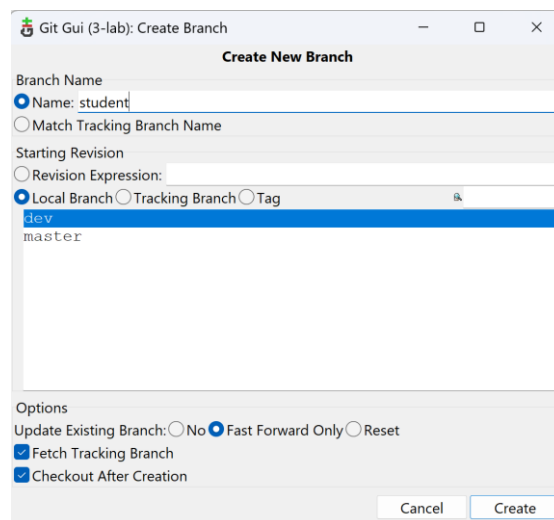


Рисунок 6. Добавление веток dev, student.

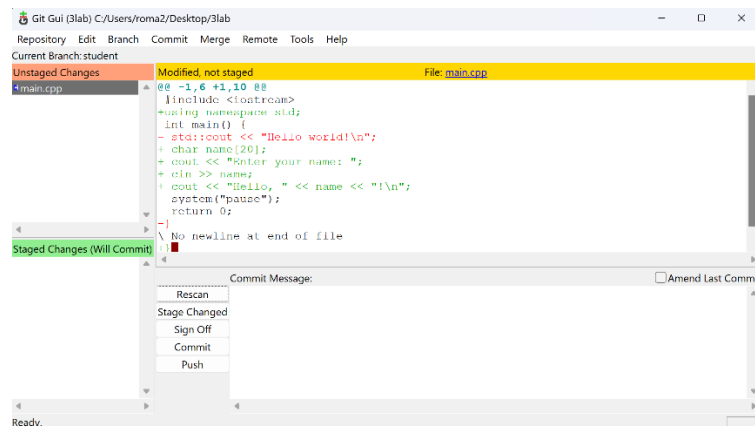


Рисунок 7. Изменение файла main.cpp и commit

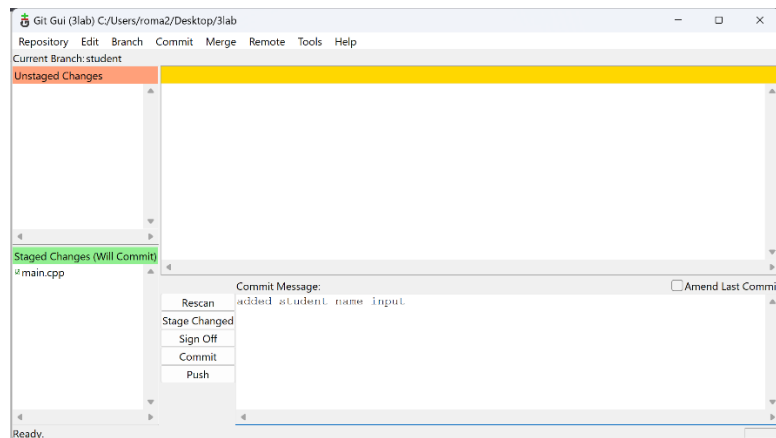


Рисунок 8. Commit изменённого main.cpp на ветке student.

Далее через Branch – Checkout выбираем ветку dev и проделываем ту же работу, что и на ветке student.

Продолжаем ту же работу для ветки master и выводим историю коммитов через Repository – Visualize All Branch History:

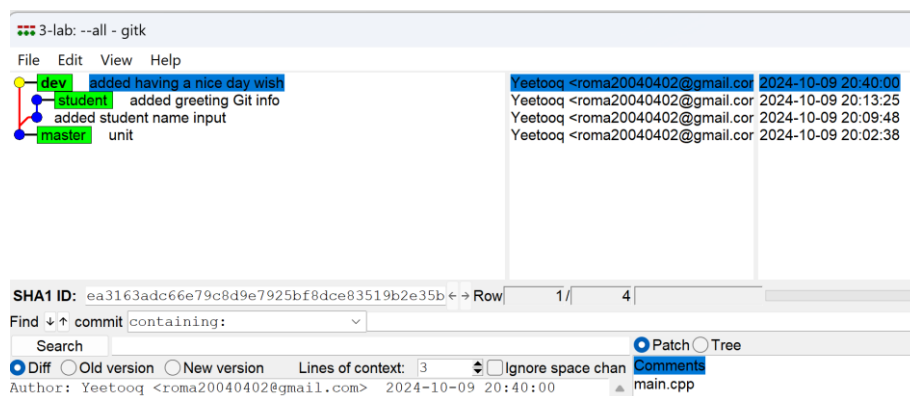


Рисунок 9. История коммитов.

Изменяем в ветке dev файл main.cpp и слиаем dev со student:

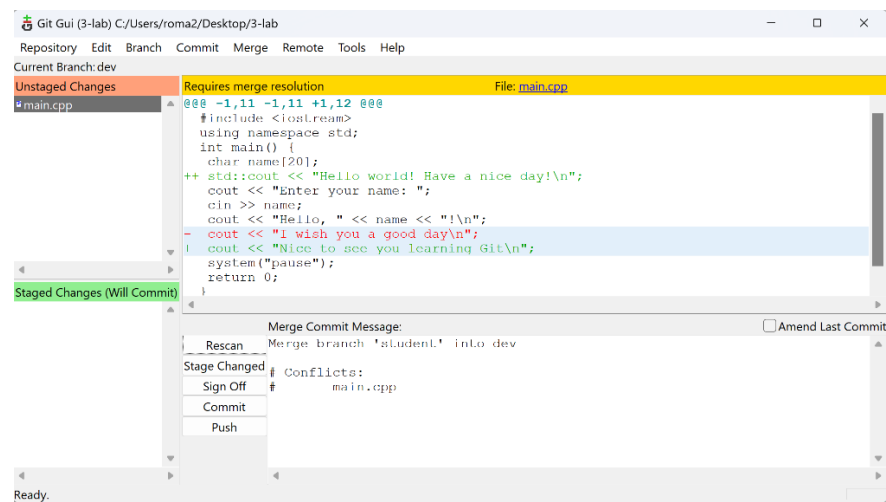


Рисунок 10. Изменённый main.cpp на ветке dev.

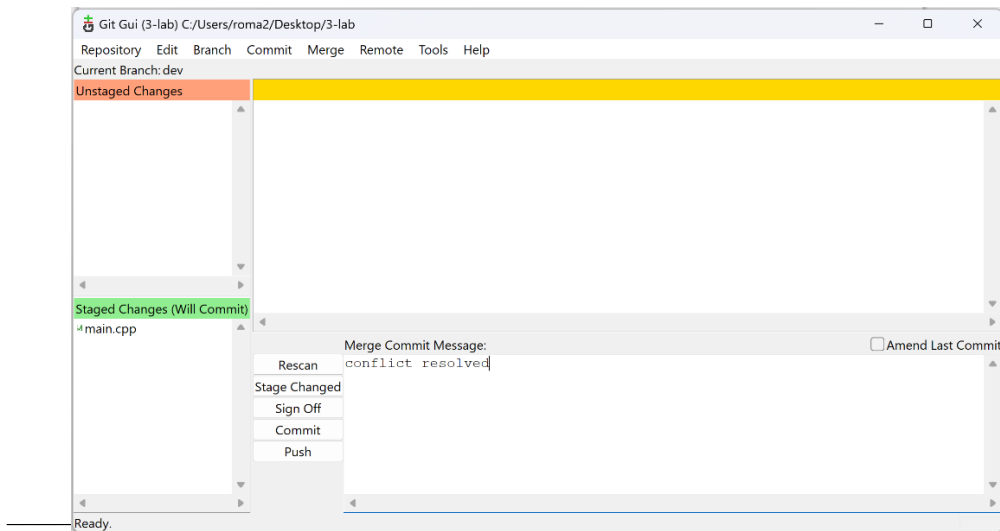


Рисунок 11. Commit изменённого main.cpp на ветке dev.

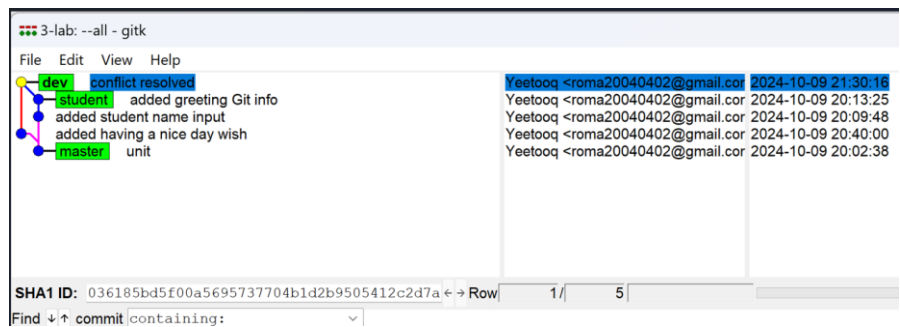


Рисунок 12. История коммитов после изменений в dev.

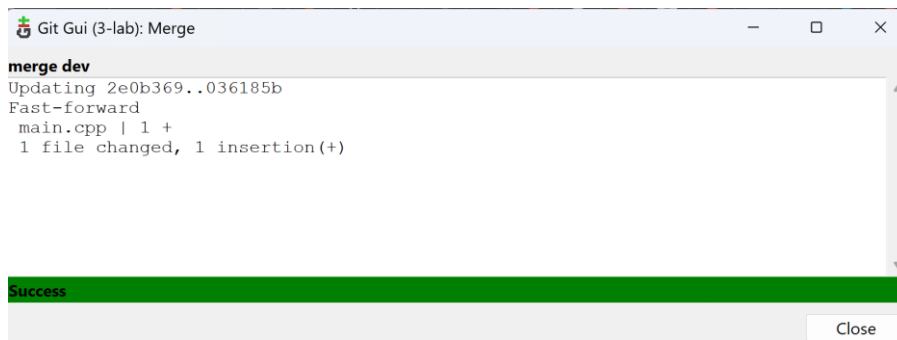


Рисунок 13. Удачное слияние student с dev.

Далее по аналогии сливаем ветку dev в ветку master.

Создание удалённого репозитория:

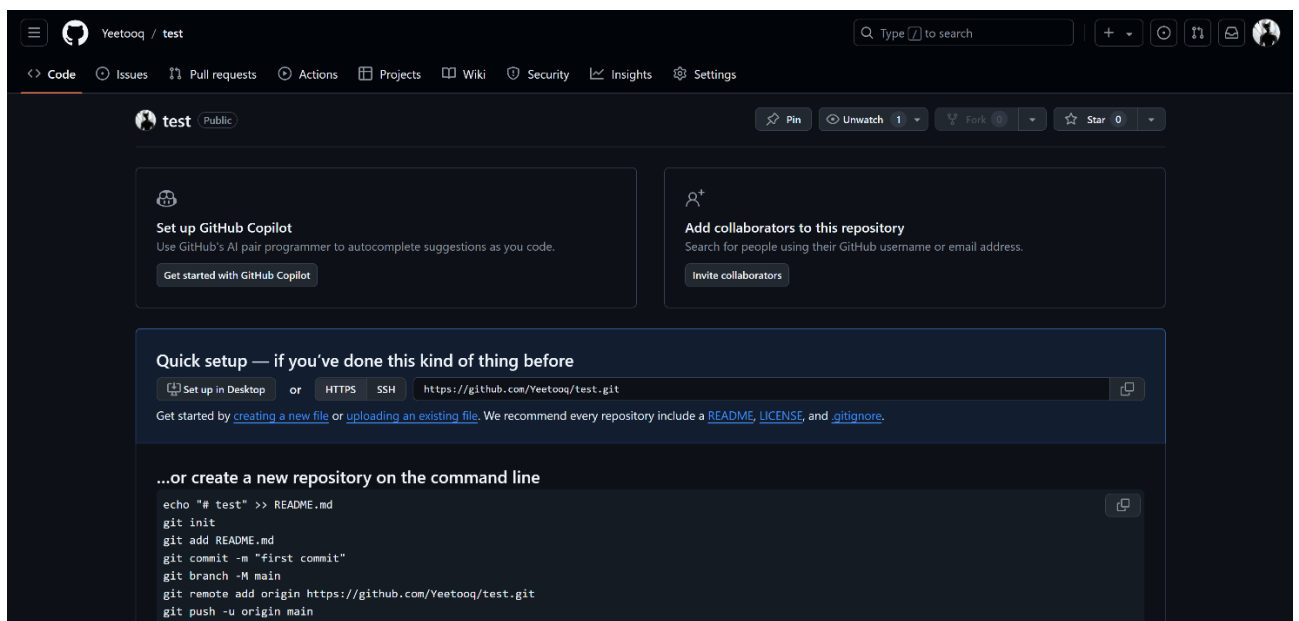


Рисунок 14. Удалённый репозиторий test на GitHub.

Через Remote – Push загружаем свой проект на удалённый репозиторий:

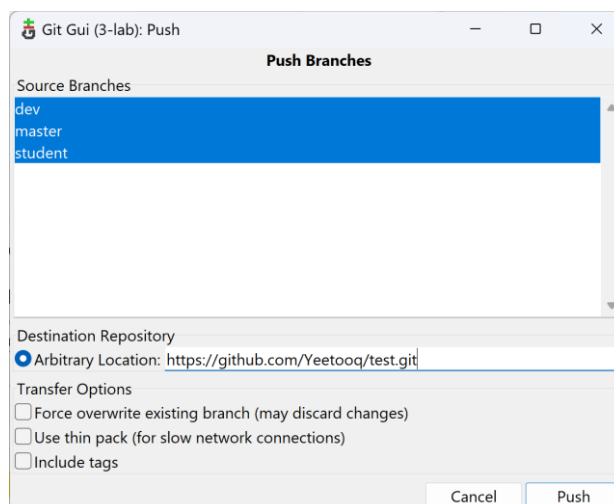
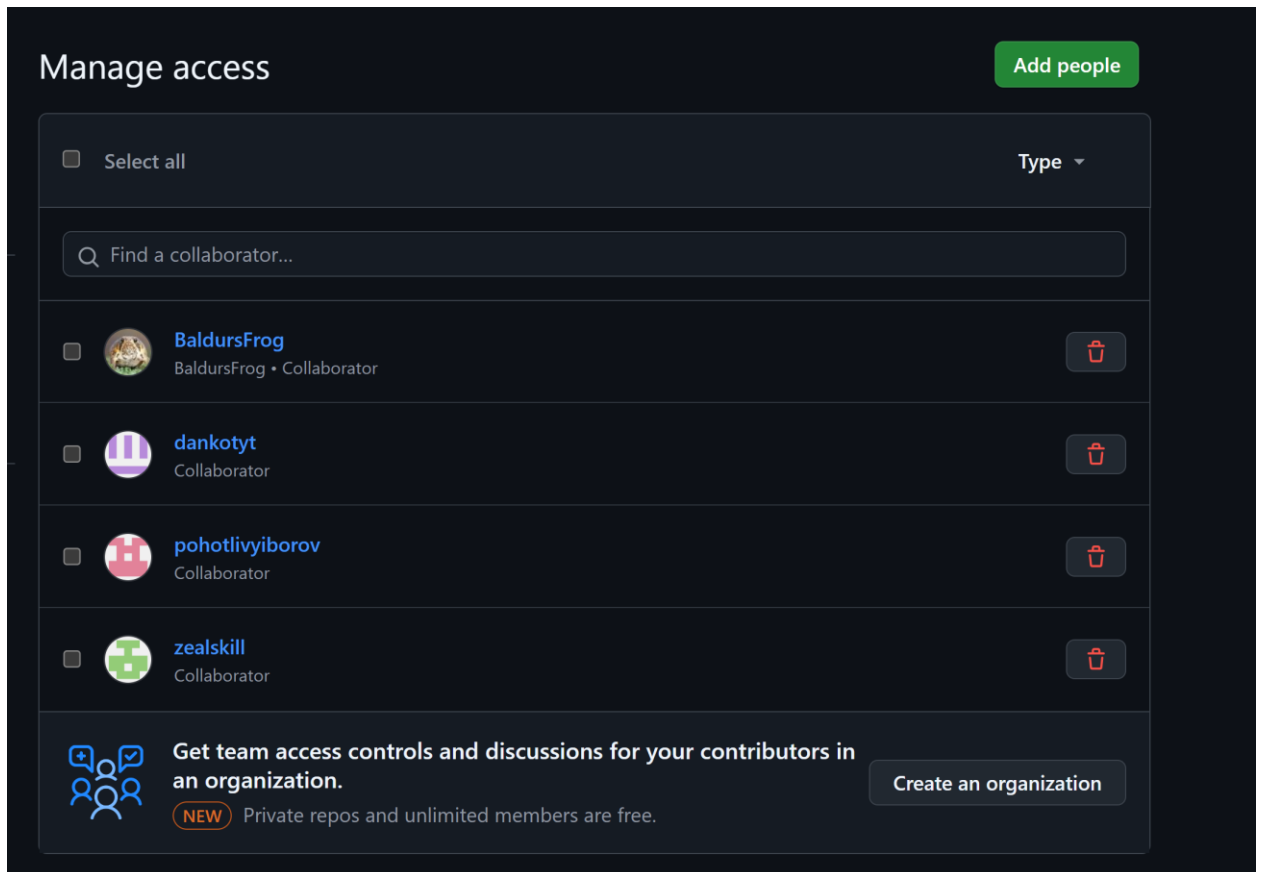
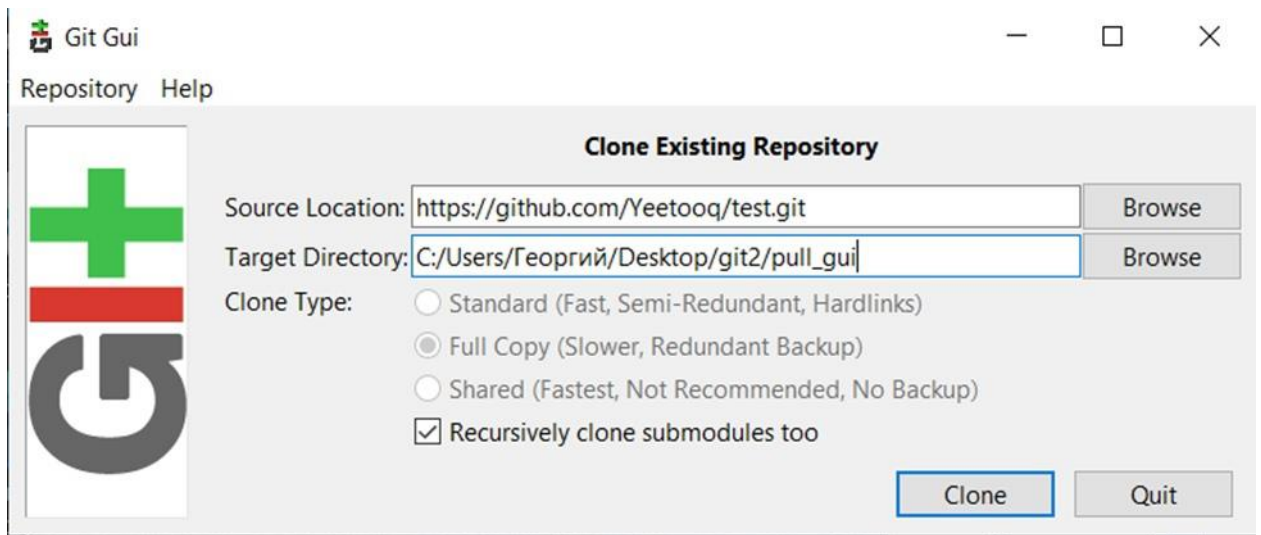


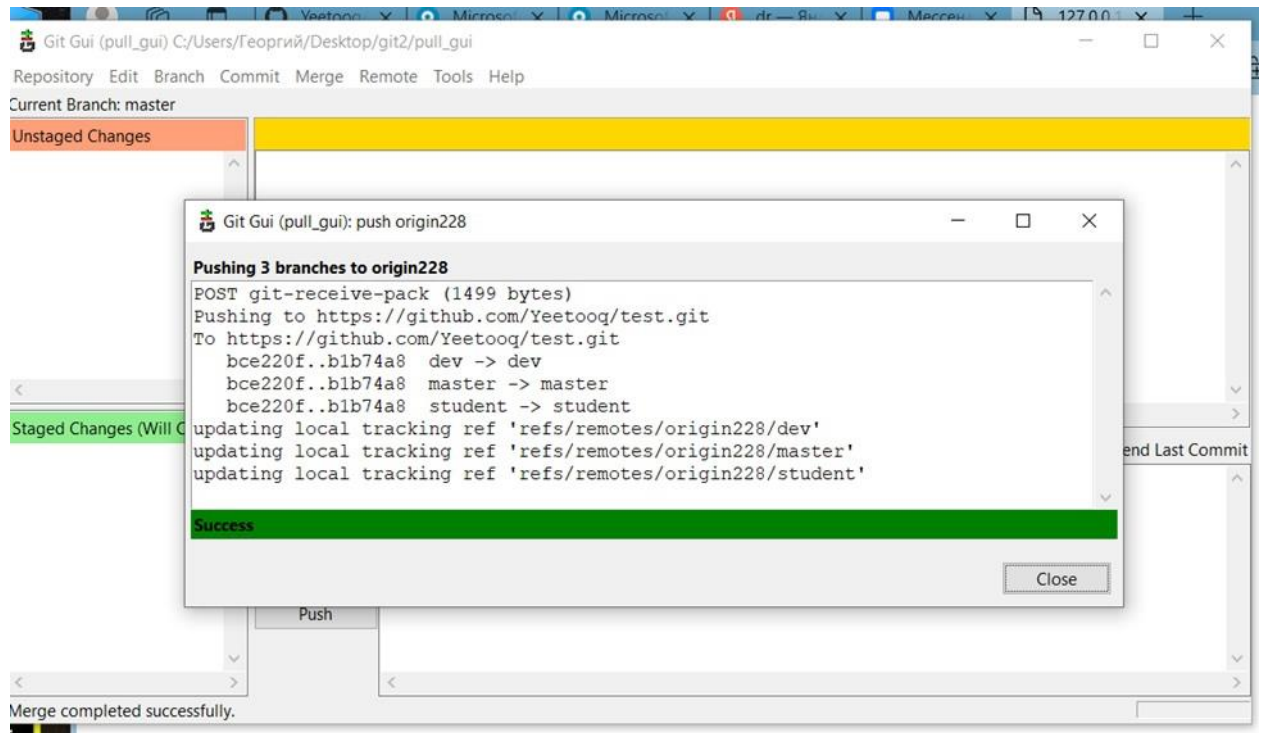
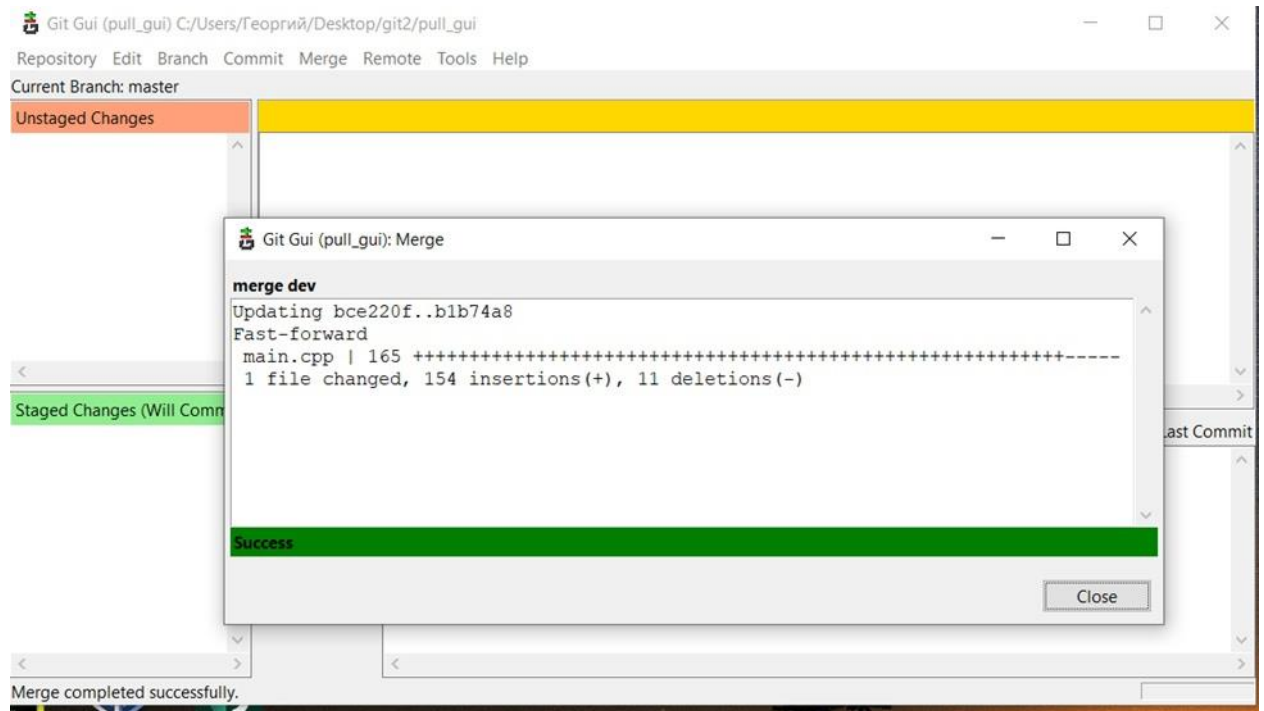
Рисунок 15. Загрузка на удал. репозиторий нашего проекта.

Обеспечиваем доступ к репозиторию для всей команды:



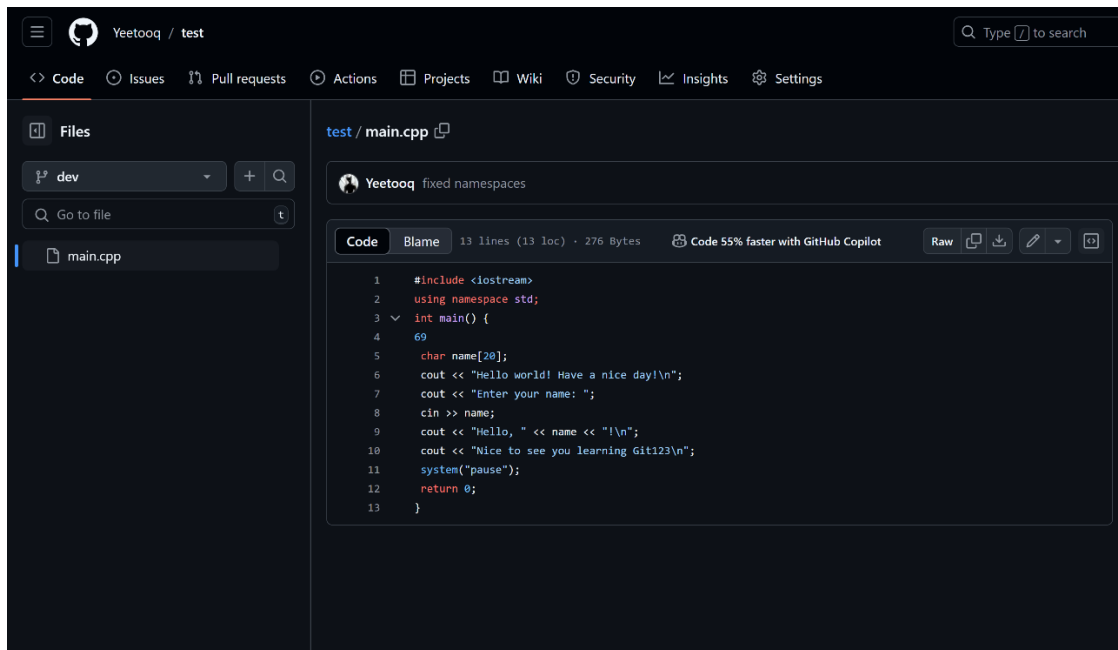
Далее Георгий (BaldursFrog) изменяет main.cpp:





Видим изменения в репозитории гитхаб.

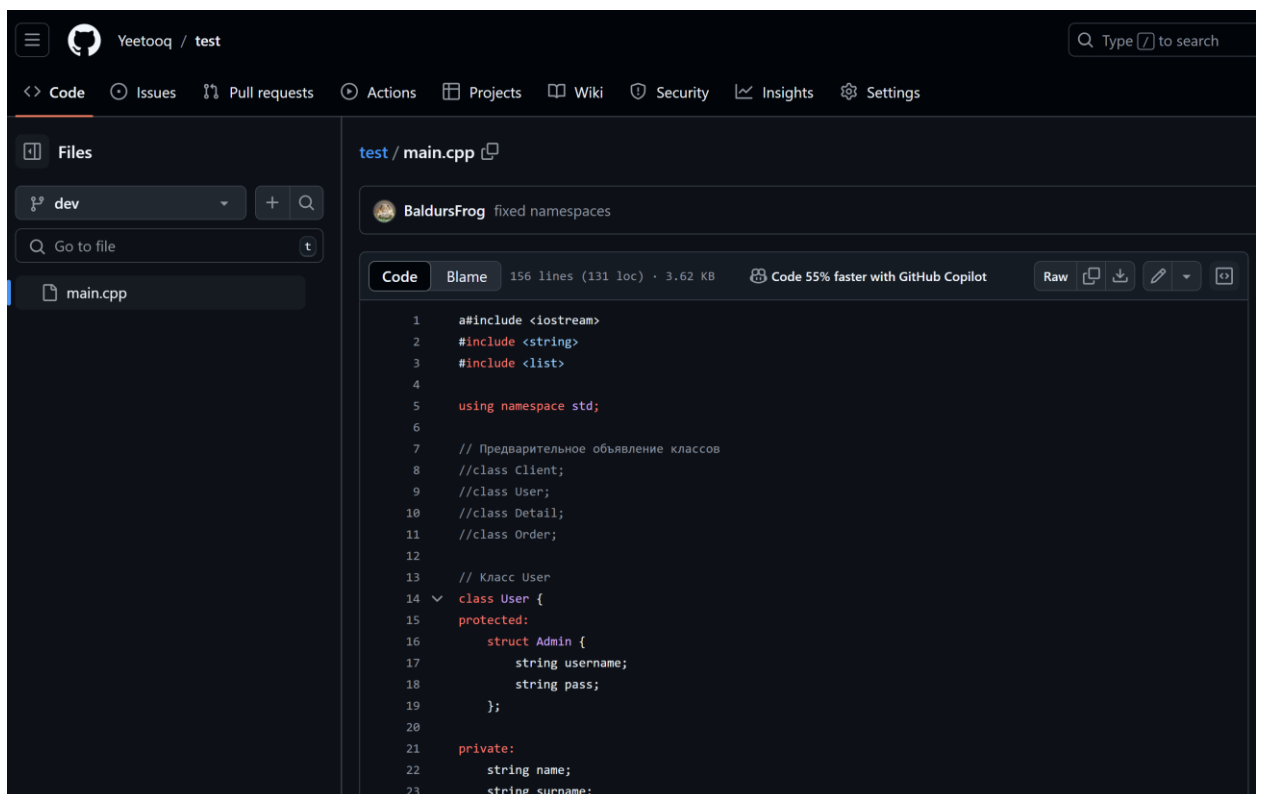
До изменения:



The screenshot shows the GitHub interface for the repository 'Yeetooq / test'. The 'Files' sidebar on the left shows a file named 'main.cpp'. The main content area displays the code for 'test / main.cpp' by user 'Yeetooq'. The code is a simple C++ program that prints 'Hello world! Have a nice day!', prompts for a name, and prints a greeting. The code is 13 lines long, 13 loc, and 276 Bytes. The interface includes a search bar at the top right and a navigation bar with links to Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings.

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     69
5     char name[20];
6     cout << "Hello world! Have a nice day!\n";
7     cout << "Enter your name: ";
8     cin >> name;
9     cout << "Hello, " << name << "!\n";
10    cout << "Nice to see you learning Git123\n";
11    system("pause");
12    return 0;
13 }
```

После изменения:



The screenshot shows the same GitHub repository 'Yeetooq / test', but the code in 'main.cpp' has been updated by user 'BaldursFrog'. The code is now 156 lines long, 131 loc, and 3.62 KB. It includes additional headers for <string> and <list>, and defines several classes: Client, User, Detail, Order, and Knacc User. The User class is protected and has a struct Admin with username and pass attributes. The code is 156 lines long, 131 loc, and 3.62 KB. The interface is the same as the previous screenshot, but the code content is different.

```
1 a#include <iostream>
2 #include <string>
3 #include <list>
4
5 using namespace std;
6
7 // Предварительное объявление классов
8 //class Client;
9 //class User;
10 //class Detail;
11 //class Order;
12
13 // Knacc User
14 class User {
15     protected:
16         struct Admin {
17             string username;
18             string pass;
19         };
20
21     private:
22         string name;
23         string surname;
```