Московский авиационный институт

(Национальный исследовательский университет)

**Лабораторные работы 1 - 10**

**Архитектура программной среды**

**Лаборатория 1.** Интегрированные среды разработки и их базовые отличия.

**Лаборатория 2.** Потоковая диаграмма GitHub и Архитектура среды.

**Лабораторная 3.** Разработка потоковых диаграмм их реализация в среде VS Code.

**Лабораторная 4.** Создание профайла VS Code для работы в среде Python. Виды профайлов. Их использование при распределенной разработки. Структура профайла. GitHub Copilot. UML (Unified Model Language) - диаграмма взаимодействий.

**Лабораторная 5.** Создание ветвей в GitHub для распределенной разработки и их слияние из среды VS Code.

**Лабораторная 6.**

**Лабораторная 7.**

**Лабораторная 8.**

**Лабораторная 9.**

**Лабораторная 10.**

**\*Laboratory 11.**

**\*Laboratory 12.**

|  |  |  |
| --- | --- | --- |
| Калугин К.А. | подпись: | \_\_\_\_\_\_ |
| Семенов А. С. | подпись: | \_\_\_\_\_\_ |
| Дата: | \_\_\_\_\_\_\_ |  |

**Лабораторная 1.**  Интегрированные среды разработки и их базовые отличия

**1. Что такое интегрированная среда разработки (IDE) и для чего она используется?**

**Интегрированная среда разработки (IDE)** – это программное приложение, которое объединяет в себе различные инструменты для разработки программного обеспечения. IDE помогает программистам писать, редактировать, компилировать, тестировать и отлаживать код.

**2. Какие основные компоненты включает типичная IDE?**

* **Редактор кода:** предназначен для написания и редактирования кода.
* **Компилятор:** преобразует код в машинный код, который может быть исполнен компьютером.
* **Отладчик:** позволяет пошагово выполнять код и находить ошибки.
* **Инструменты сборки:** автоматизируют процесс сборки проекта.
* **Система контроля версий:** позволяет отслеживать изменения в коде и восстанавливать предыдущие версии.

**3. В чем отличие между текстовым редактором и интегрированной средой разработки?**

**Текстовый редактор** – это программа, предназначенная для создания и редактирования текстовых файлов. IDE, помимо функций текстового редактора, предлагает множество дополнительных инструментов, которые упрощают и ускоряют процесс разработки программного обеспечения.

**4. Какую роль играют плагины в работе с интегрированными средами разработки?**

**Плагины** – это дополнительные модули, которые расширяют функциональность IDE. Они могут добавлять новые функции, языки программирования, темы оформления и другие возможности.

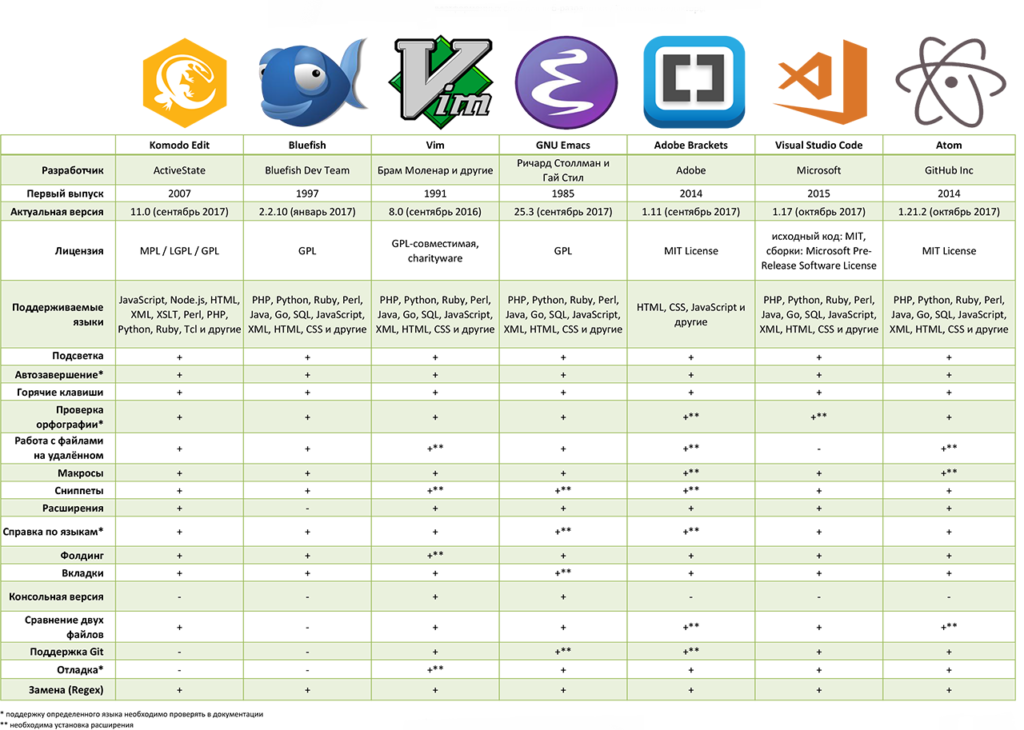
**5. Какие преимущества предоставляет использование IDE при разработке программного обеспечения?**

* **Повышение производительности:** IDE автоматизирует многие рутинные задачи, что позволяет программистам работать быстрее.
* **Улучшение качества кода:** IDE помогает находить и исправлять ошибки в коде.
* **Улучшенная отладка:** IDE предоставляет инструменты для пошагового выполнения кода и поиска ошибок.
* **Сокращение времени разработки:** IDE позволяет создавать и тестировать код быстрее, чем при использовании текстового редактора.

**Заключение:**

Использование IDE может значительно повысить эффективность разработки программного обеспечения. IDE предоставляет программистам все необходимые инструменты для написания, редактирования, компиляции, тестирования и отладки кода.

Сравнение IDE:



## Алгоритм создания простого консольного приложения на Python в VSC:

**1. Создание проекта:**

* Откройте VSCode.
* Нажмите Ctrl+Shift+N (или Cmd+Shift+N на Mac), чтобы создать новый файл.
* Сохраните файл с расширением .py.
* Введите имя проекта (например, calculator.py).

**2. Написание кода:**

* Введите код вашего приложения в файл .py.
* Для простого консольного приложения достаточно нескольких строк кода.
* Например, код для вывода приветствия:

Python

print("Приветствую!")

**3. Запуск приложения:**

* Нажмите F5 (или Cmd+F5 на Mac), чтобы запустить приложение.
* VSCode автоматически создаст и запустит виртуальную среду Python.
* Вывод приложения появится в консоли VSCode.

**4. Отладка приложения:**

* Если в приложении есть ошибки, VSCode поможет вам их найти.
* Установите точки останова, чтобы пошагово выполнять код.
* Используйте окно отладки для просмотра значений переменных.

**5. Дополнительные возможности:**

* VSCode предлагает множество дополнительных функций для разработки Python-приложений.
* Используйте IntelliSense для автодополнения кода.
* Установите плагины для добавления новых функций.

**Пример:**

Python

def sum\_numbers(a, b):

"""

Функция для сложения двух чисел.

Args:

a: первое число.

b: второе число.

Returns:

Сумма a и b.

"""

return a + b

# Вызов функции

print(sum\_numbers(10, 20))

**Вывод:**

30

**Заключение:**

VSCode – это удобная и многофункциональная IDE для разработки Python-приложений. Она позволяет создавать простые и сложные приложения, используя различные инструменты и функции.

**Лабораторная 2.** Потоковая диаграмма GitHub и Архитектура среды.

**1. Что такое потоковая диаграмма в контексте GitHub и какие основные элементы она включает?**

**Потоковая диаграмма GitHub** – это визуальное представление процесса разработки программного обеспечения, которое использует ветки (branches) GitHub для управления версиями кода. Она помогает разработчикам

* **Визуализировать** этапы разработки.
* **Определить** роли и обязанности участников проекта.
* **Отслеживать** ход разработки.
* **Обеспечить** согласованность кода.

**Основные элементы потоковой диаграммы GitHub:**

* **Репозиторий**: Хранилище кода проекта.
* **Ветки**: Разные версии кода проекта.
* **Pull Requests**: Запросы на изменение кода.
* **Issues**: Отслеживание ошибок и проблем.
* **Merge**: Объединение разных версий кода.

**2. Какую роль играют ветки (branches) в потоковой диаграмме GitHub и каким образом они помогают в организации работы над проектом?**

**Ветки** – это ключевой элемент потоковой диаграммы GitHub.

* **Создание** разных версий кода (например, master, develop, feature/new-feature).
* **Параллельная** работа над разными функциями.
* **Тестирование** изменений перед их слиянием с основной веткой (master).
* **Отслеживание** изменений и истории разработки.

**3. Какие основные принципы лежат в основе архитектуры среды GitHub , позволяющей эффективно управлять версиями кода и совместной разработкой?**

* **Git**: Система контроля версий, позволяющая отслеживать изменения кода.
* **GitHub**: Платформа для хостинга Git-репозиториев.
* **Ветки**: Разные версии кода проекта.
* **Pull Requests**: Запросы на изменение кода.
* **Issues**: Отслеживание ошибок и проблем.
* **Wiki**: Создание документации для проекта.
* **GitHub Pages**: Публикация статических веб-сайтов.

**4. Каким образом создается локальный репозиторий в GitHub?**

1. Установите Git на свой компьютер.
2. Откройте терминал.
3. Перейдите в папку, где хотите создать локальный репозиторий.
4. Выполните команду git init.
5. Добавьте файлы проекта в папку .git.
6. Выполните команду git add.
7. Выполните команду git commit -m "Initial commit".

**5. Какие шаги необходимо выполнить для создания системного репозитория в GitHub и какие преимущества это может предоставить при работе в команде?**

**Создание системного репозитория:**

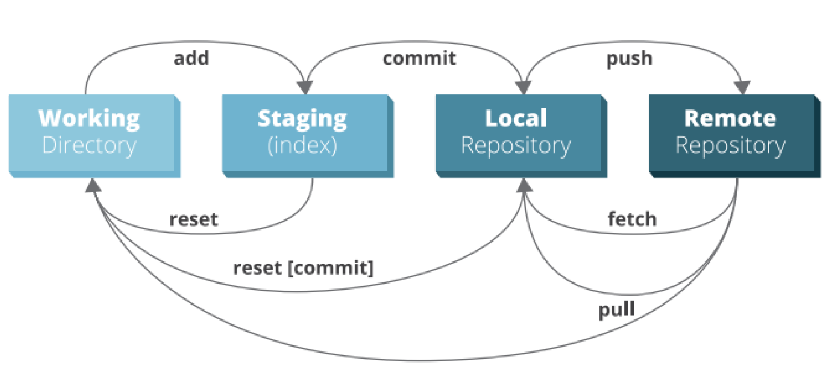
1. Создайте учетную запись на GitHub.
2. Создайте новый репозиторий.
3. Клонируйте репозиторий на локальный компьютер.
4. Добавьте файлы проекта.
5. Выполните git add.
6. Выполните git commit.
7. Выполните git push.

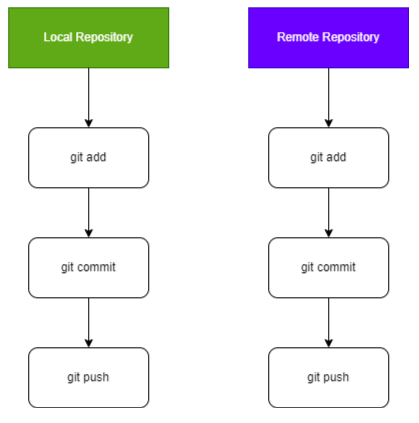
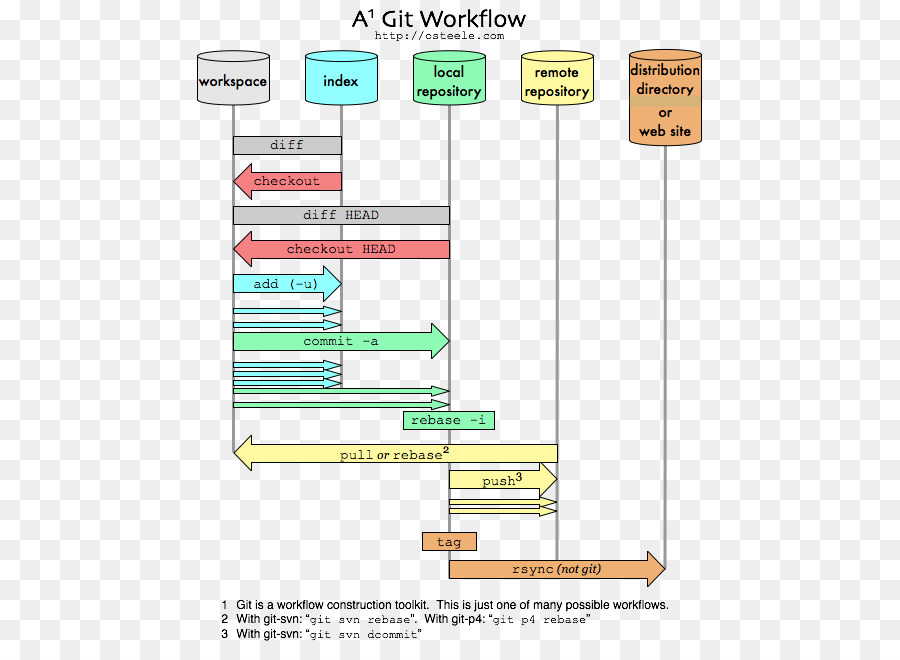
**Преимущества:**

* **Централизованное хранилище кода:** Все участники команды имеют доступ к одной версии кода.
* **Управление версиями:** Отслеживание изменений кода и возможность откатить к предыдущим версиям.
* **Совместная работа:** Разработчики могут одновременно работать над кодом.
* **Контроль доступа:** Ограничение доступа к коду для разных пользователей.
* **История изменений:** Просмотр истории изменений кода.

**Заключение:**

GitHub – это мощная платформа, которая позволяет эффективно управлять версиями кода

/

**Лабораторная 3.** Создание профайла VS Code для работы в среде Python. Виды профайлов. Их использование при распределенной разработки. Структура профайла.

**1. Что представляет собой профиль в среде Visual Studio Code (VS Code) и какова его роль при работе с Python?**

Профиль в VS Code — это набор настроек, определяющих среду разработки для конкретного языка программирования или проекта. В контексте Python, профиль может включать:

* Интерпретатор Python
* Библиотеки и пакеты
* Настройки автодополнения кода
* Форматирование кода
* Linting
* Запуск и отладка
* Тестирование

**2. Какие виды профилей существуют в VS Code для работы с Python, и как они различаются по функционалу?**

VS Code предлагает несколько предустановленных профилей для Python:

* **Python:** базовый профиль для разработки на Python
* **Django:** профиль для разработки веб-приложений с Django
* **Flask:** профиль для разработки веб-приложений с Flask
* **Jupyter Notebook:** профиль для работы с Jupyter Notebooks

**3. В чем заключается значение использования профилей при распределенной разработке программного обеспечения с использованием Python и VS Code?**

Использование профилей при распределенной разработке:

* **Обеспечивает унифицированную среду разработки** для всех членов команды.
* **Уменьшает количество ошибок**, связанных с несовместимыми настройками.
* **Упрощает совместную работу** над проектом.

**4. Какова типичная структура профиля в VS Code для работы с Python, и какие основные настройки обычно включаются в такой профиль?**

**Структура профиля:**

* **settings.json:** содержит основные настройки профиля, такие как интерпретатор Python, пути к библиотекам и пакетам.
* **launch.json:** содержит настройки для запуска и отладки кода.
* **tasks.json:** содержит настройки для сборки, тестирования и других задач.

**Основные настройки:**

* **Интерпретатор Python:** путь к интерпретатору Python, который будет использоваться в профиле.
* **Библиотеки и пакеты:** список библиотек и пакетов, которые будут доступны в профиле.
* **Настройки автодополнения кода:** параметры автодополнения кода, такие как импорт модулей и подсказки по функциям.
* **Форматирование кода:** правила форматирования кода, такие как отступы и скобки.
* **Linting:** правила проверки кода на наличие ошибок и потенциальных проблем.
* **Запуск и отладка:** параметры запуска и отладки кода, такие как точки останова и режим отладки.
* **Тестирование:** параметры тестирования кода, такие как тестовый фреймворк и команды запуска тестов.

**5. Как можно оптимизировать профиль в VS Code для увеличения производительности и удобства работы при разработке на Python?**

* **Используйте только те библиотеки и пакеты, которые вам нужны.**
* **Отключите ненужные функции**, такие как автодополнение кода для неиспользуемых библиотек.
* **Установите плагины для оптимизации производительности**, такие как Python Language Server.
* **Используйте темы оформления** с низким потреблением ресурсов.
* **Настройте горячие клавиши** для часто используемых действий.

**Алгоритм:**

**1. Создание профиля:**

* Откройте VS Code.
* Выберите "File" -> "Preferences" -> "Settings".
* Нажмите кнопку "Create New Settings".
* Введите имя профиля, например "Python".
* Выберите "Python" в качестве языка программирования.
* Настройте необходимые параметры.

**2. Разработка структуры профиля для распределенной разработки:**

* **Создайте папку для проекта.**
* **Добавьте файл .gitignore** для игнорирования ненужных файлов.
* **Создайте файл requirements.txt** со списком библиотек и пакетов.
* **Создайте файл settings.json** с настройками профиля.
* **Создайте файл launch.json** с настройками для запуска и отладки кода.
* **Создайте файл tasks.json** с настройками для сборки, тестирования и других задач.

**3. Настройка профиля для распределенной разработки:**

* **Установите плагин GitLens.**
* **Настройте Git для совместной работы.**
* **Установите плагин Live Share.**
* **Настройте Live Share для совместного редактирования кода.**

**Лабораторная 4.** Разработка потоковых диаграмм. UML (Unified Model Language) - диаграмма взаимодействий.

### 1. Потоковые диаграммы

**1.1 Определение и элементы**

**Потоковая диаграмма** (flowchart) – это графическое представление алгоритма, которое использует различные символы для отображения последовательности действий и логических операций.

**Основные элементы:**

* **Начало/конец:** обозначают начало и завершение алгоритма.
* **Действие:** блок, описывающий конкретное действие.
* **Решение:** блок, позволяющий выбрать один из нескольких путей выполнения алгоритма.
* **Объединение:** блок, объединяющий потоки выполнения алгоритма.
* **Цикл:** блок, повторяющий действия до выполнения определенного условия.
* **Ввод/вывод:** блок, предназначенный для ввода или вывода данных.

**1.2 Роль в разработке ПО**

**Потоковые диаграммы:**

* **Визуализируют алгоритмы:** делают их более понятными и удобными для анализа.
* **Упрощают документирование:** позволяют зафиксировать алгоритм в понятном виде.
* **Способствуют отладке:** помогают выявить ошибки и логические противоречия в алгоритме.
* **Улучшают коммуникацию:** позволяют команде разработчиков лучше понимать друг друга.

**1.3 Создание в VSC**

**VSCode** (Visual Studio Code) – популярный редактор кода, который позволяет создавать потоковые диаграммы с помощью:

* **Расширений:**
  + **Flowchart.js:** простой и удобный инструмент для создания диаграмм.
  + **Mermaid:** мощный инструмент с поддержкой различных типов диаграмм.
* **Онлайн-сервисов:**
  + **Creately:** позволяет создавать диаграммы в браузере.
  + **Gliffy:** обладает богатым набором инструментов для создания диаграмм.

### 2. Диаграммы взаимодействия UML

**2.1 Определение и виды**

**Диаграмма взаимодействия UML** (Unified Modeling Language) – это графическое представление взаимодействия между объектами системы.

**Основные виды:**

* **Диаграмма последовательности:** отображает хронологическую последовательность обмена сообщениями между объектами.
* **Диаграмма кооперации:** показывает структурную организацию объектов, участвующих во взаимодействии.
* **Диаграмма用例:** описывает сценарии использования системы.

**2.2 Применение**

**Диаграммы взаимодействия:**

* **Моделируют поведение системы:** позволяют визуализировать сценарии взаимодействия объектов.
* **Анализируют систему:** помогают выявить ошибки и узкие места в дизайне системы.
* **Документируют систему:** позволяют зафиксировать сценарии взаимодействия в понятном виде.

### 3. Алгоритмы

**3.1 Потоковая диаграмма работы с базой данных**

**1.** **Начало** **2.** **Подключение к базе данных** **3.** **Ввод запроса** **4.** **Обработка запроса** **5.** **Вывод результатов** **6.** **Отключение от базы данных** **7.** **Конец**

**3.2 Диаграмма взаимодействия регистрации пользователя**

**1.** **Пользователь:** вводит имя, email и пароль.

**2.** **Система:** проверяет корректность данных.

**3.** **Система:** генерирует код подтверждения.

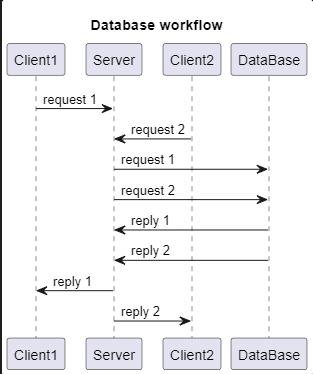
**4.** **Система:** отправляет код подтверждения на email пользователя.

**5.** **Пользователь:** вводит код подтверждения.

**6.** **Система:** проверяет код подтверждения.

**7.** **Система:** регистрирует пользователя.

**8.** **Система:** сообщает об успешной регистрации.



**Лабораторная 5**. Создание ветвей в GitHub для распределенной разработки и их слияние из среды VS Code.

**1. Создание новой ветки:**

**Командная строка:**

git checkout -b имя\_ветки master/main

**Веб-интерфейс:**

* Перейдите в репозиторий на GitHub.
* Нажмите кнопку "Branch".
* Нажмите "New Branch".
* Введите имя ветки и нажмите "Create Branch".

**2. Основные принципы работы с ветками:**

* Используйте ветки для изоляции изменений.
* Создавайте ветки для каждой задачи или функции.
* Регулярно синхронизируйте ветки с master/main.
* Используйте дескриптивные имена для веток.
* Удаляйте ненужные ветки.

**3. Слияние веток:**

git checkout master/main

git merge имя\_ветки

**4. Конфликты слияния:**

* Возникают при одновременном изменении одного и того же файла в разных ветках.
* VS Code:
  + Показывает конфликты в редакторе.
  + Предоставляет инструменты для их разрешения.

**5. Инструменты VS Code:**

* Git: отображение статуса ветки, управление ветками.
* Source Control: просмотр истории коммитов, сравнение файлов.
* Problems: отображение ошибок и конфликтов.

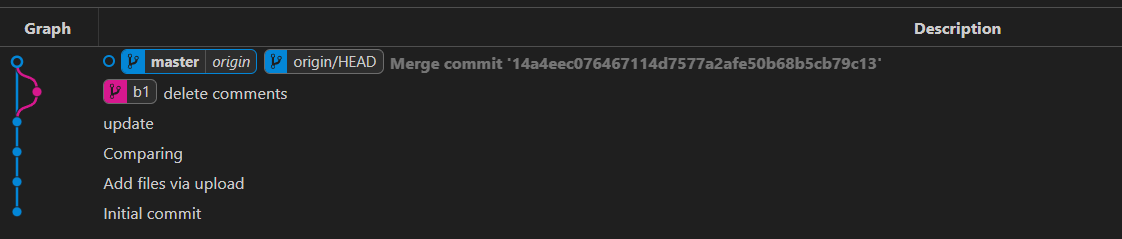
**Задача 1:**

1. Создайте новую ветку feature от master/main.
2. Добавьте изменения в файлы в VS Code.
3. Закоммитите изменения.
4. Перейдите на master/main.
5. Выполните git merge feature.
6. При необходимости разрешите конфликты слияния.

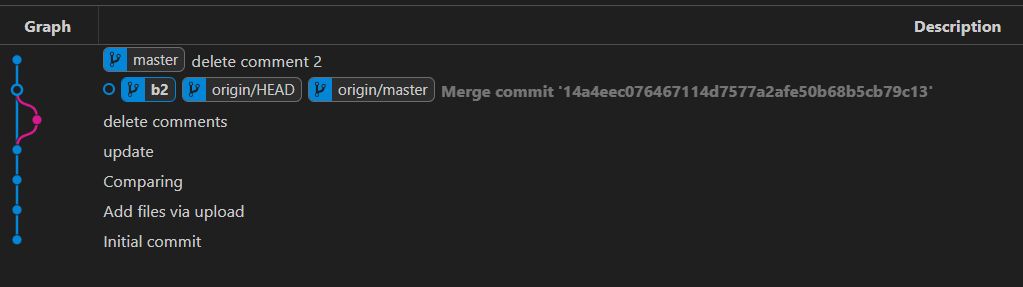
**Задача 2:**

1. Откройте проект в VS Code.
2. Перейдите на ветку с конфликтом.
3. VS Code покажет проблемные файлы.
4. Сравните изменения и выберите нужные фрагменты кода.
5. Сохраните изменения.
6. Закоммитите изменения.
7. Перейдите на master/main.
8. Выполните git merge имя\_ветки.

Результат слияния веток :



Теперь выполним смещения статуса мастер ветки на новую ветку. Для этого создаем новую ветку и изменяем немного исходный код.



И выполняем функцию rebase branch. Результат выполнения операции rebase:

