Московский авиационный институт

(Национальный исследовательский университет)

**Лабораторные работы 1 - 10**

**Архитектура программной среды**

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

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

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

**Лабораторная 4.** Разработка потоковых диаграмм их реализация в среде VS Code.

**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.**

**Лабораторная 11.**

|  |  |  |
| --- | --- | --- |
| Тимофеева Н. О. | подпись: | \_\_\_\_\_\_ |
| Семенов А. С. | подпись: | \_\_\_\_\_\_ |
| Дата: | \_\_\_\_\_\_\_ | 2024 |

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

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

IDE (интегрированная среда разработки) - это программное обеспечение, предназначенное для облегчения процесса разработки. Оно обычно включает в себя текстовый редактор, инструменты для отладки кода, компиляции, а также другие функции, например, подсветку синтаксиса, интеграцию с системами контроля версий и т. д. IDE позволяет программистам работать более эффективно, упрощая процесс написания, тестирования и отладки кода.

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

* Текстовый редактор — используется для написания и редактирования исходного кода.
* Компилятор или интерпретатор: Позволяет преобразовать исходный код в исполняемый код. Компиляторы используются для языков, компилируемых в машинный код, а интерпретаторы - для языков, интерпретируемых построчно.
* Отладчик — инструмент для поиска и исправления ошибок в коде пошагово.
* Система управления версиями — интеграция с системами контроля версий позволяет отслеживать изменения в коде, вносить комментарии и совершать коммиты.

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

Текстовый редактор - простой инструмент для написания и редактирования текста, включая исходный код. IDE же предоставляет более широкий спектр инструментов для разработки программного обеспечения. Это включает в себя текстовый редактор, компилятор или интерпретатор, отладчик, систему управления версиями.

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

Плагины представляют собой дополнительные модули, которые расширяют функциональность интегрированных сред разработки (IDE). Они могут добавлять новые функции, поддержку языков программирования, изменять внешний вид и другие возможности, что позволяет настраивать IDE под конкретные потребности разработчика

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

* Увеличение производительности — IDE предоставляют различные инструменты, такие как автодополнение кода, подсвечивание кода и ошибок, что помогает ускорить процесс написания и отладки кода.
* Повышение качества кода — IDE обычно предоставляют инструменты для контроля качества и форматирования, что помогает выявлять и исправлять ошибки и поддерживать стандарты написания кода.
* Интеграция с внешними сервисами — многие IDE интегрированы с системами управления версиями, что облегчает совместную работу и управление проектами.
* Поддержка различных языков программирования — многие IDE поддерживают несколько языков программирования и фреймворков, что позволяет разработчикам работать с разнообразными технологиями в единой среде.
* Расширяемость — большинство IDE поддерживают плагины и расширения, позволяющие настраивать среду разработки под конкретные потребности и добавлять новые функции и инструменты.

**Задача 1: Выберите 2 популярные интегрированные среды разработки и сравните их основные характеристики**

При выборе редактора кода для работы над проектом перед разработчиком часто встает вопрос о том, какой инструмент лучше всего подходит для решения его задач. Рассмотрим два популярных редактора – PhpStorm от компании JetBrains и VSCode от Microsoft.

Основные характеристики

Обе программы предоставляют набор функций, необходимых для разработки на разных языках программирования. Они поддерживают системы контроля версий, такие как Git, предоставляют инструменты для отладки и анализа кода, а также обеспечивают подсветку синтаксиса и автоматическое завершение кода.

Однако, есть различия. Например, PhpStorm имеет более широкий набор инструментов для работы с PHP, включая интеграцию с популярными фреймворками, такими как Laravel и Symfony.

Преимущества PhpStorm:

– Широкий функционал: PhpStorm предоставляет широкий спектр инструментов для разработки на PHP, таких как отладка, анализ кода, автодополнение и другие.

– Интеграция с другими продуктами: PhpStorm хорошо интегрируется с другими продуктами компании JetBrains, что может быть удобно для разработчиков, использующих несколько продуктов.

– В Phpstorm есть идеальный поиск. Если нужно найти файл, класс, настройку или информацию по git, двойной клик по shift открывает поисковик по этим параметрам.

Недостатки PhpStorm:

– Высокая стоимость: PhpStorm стоит дороже, чем Visual Studio Code.

– Ограниченная поддержка других языков: PhpStorm в основном ориентирован на разработку на PHP, и поддержка других языков может быть ограничена.

Преимущества Visual Studio Code:

– Низкая стоимость: Visual Studio Code значительно дешевле, чем PhpStorm.

– Поддержка множества языков: Visual Studio Code поддерживает множество языков программирования, включая JavaScript, TypeScript, Python и многие другие.

Недостатки Visual Studio Code:

– Меньший функционал: Visual Studio Code не предоставляет такого широкого спектра инструментов, как PhpStorm.

– Не такая хорошая интеграция с другими продуктами: Visual Studio Code менее интегрирован с другими продуктами, чем PhpStorm.

Если вы в первую очередь PHP-разработчик, то PHPStorm — очевидный выбор. Он предлагает отличную поддержку PHP с такими функциями, как завершение кода, отладка и тестирование. Однако если вы работаете с несколькими языками программирования или хотите попробовать разное, Visual Studio Code может оказаться лучшим выбором, поскольку он предлагает поддержку широкого спектра языков и платформ.

**Задача 2: Создайте простое приложение на языке программирования вашего выбора, используя одну из интегрированных сред разработки, и продемонстрируйте процесс создания, компиляции и запуска**

Я выбрала написать приложение на языке PHP, так как этот язык наиболее используется мной в последнее время. Как обсуждалось в задаче 1, наиболее удобной для этого языка IDE является PhpStorm.

* Создание проекта:

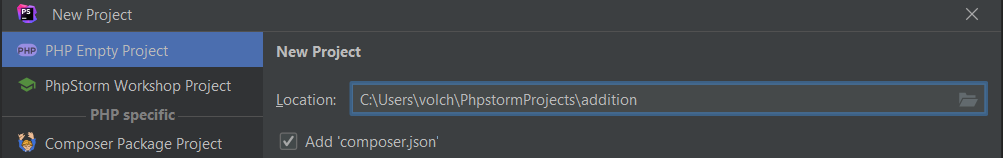
Открываем PhpStorm



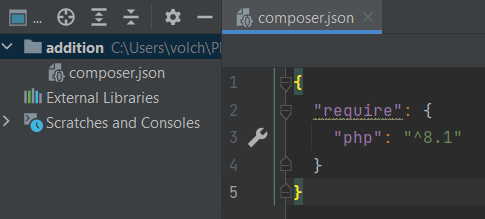
Выбираем File | New Project



Создаём пустой проект c файлом composer.json. Там хранятся все необходимые зависимости



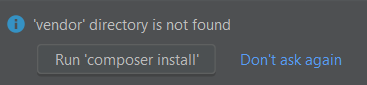
В файле composer.json пишем одну из последних версий языка, чтобы именно она использовалась в коде



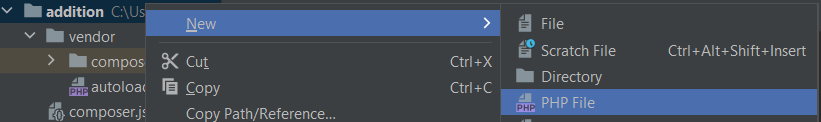
IDE подсказывает, что пакеты не установлены

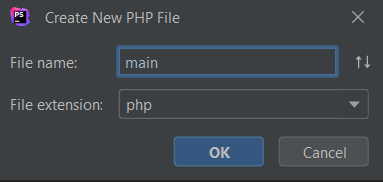


А также предлагает их установить. Нажмём на эту кнопку

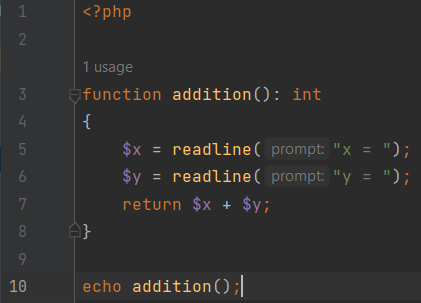


Теперь можно добавить новый файл, где будем писать код

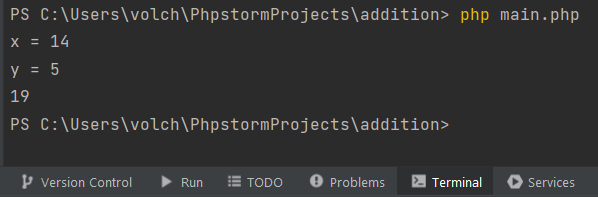




Напишем код для сложения двух целых чисел



Так как язык интерпретируемый, то нам не нужно компилировать. Запустим код в терминале и посмотрим ответ



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

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

Потоковая диаграмма в контексте GitHub представляет собой визуальное представление процесса работы над проектом с использованием системы контроля версий GitHub. Она обычно включает основные этапы или шаги работы над задачей или проектом, а также демонстрирует последовательность выполнения этих шагов.

В потоковой диаграмме GitHub обычно выделяются следующие элементы:

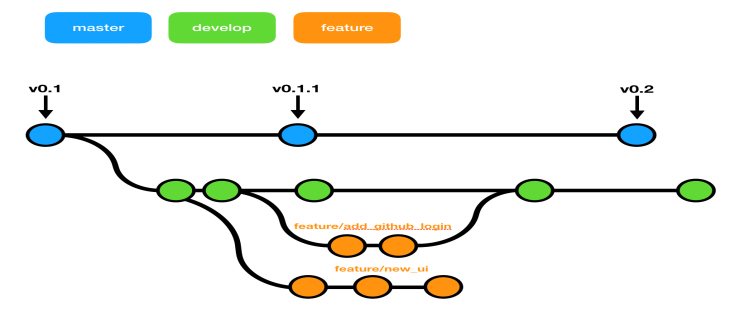
* Workspace – это рабочее пространство, где располагаются все файлы и директории, над которыми мы работаем. Здесь происходит редактирование, создание и удаление файлов.
* Index – это промежуточное хранилище измененных файлов перед фиксацией изменений (коммитом).
* Local repository – это локальное хранилище, куда коммитятся изменения. Здесь также хранятся ветки.
* Remote repository – это централизованное удаленное хранилище, где хранятся версии проекта. Оно позволяет делиться изменениями между несколькими компьютерами и членами команды.

Для синхронизации изменений между локальным и удаленным репозиториями используются операции, такие как push (отправка изменений из локального репозитория в удаленный) и pull (получение изменений из удаленного репозитория в локальный).

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

Ветки играют ключевую роль в потоковой диаграмме GitHub, обеспечивая эффективную организацию работы над проектом. Вот основные функции и преимущества веток:

* Создание различных версий кода — ветки позволяют создавать отдельные версии кода, например, ветка продакшена (master), ветка разработка (develop) и ветки для работы над конкретными функциями или задачами (feature). Это позволяет изолировать различные изменения и эксперименты, обеспечивая четкую структуру разработки.
* Параллельная работа над разными функциями — разработчики могут создавать и работать над несколькими ветками параллельно. Это позволяет разделить работу на небольшие задачи и управлять ими независимо друг от друга, что ускоряет процесс разработки.
* Отслеживание изменений и истории разработки — GitHub автоматически отслеживает и сохраняет историю изменений в каждой ветке, что облегчает восстановление предыдущих версий кода и отслеживание прогресса работы над проектом.



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

Такими принципами являются:

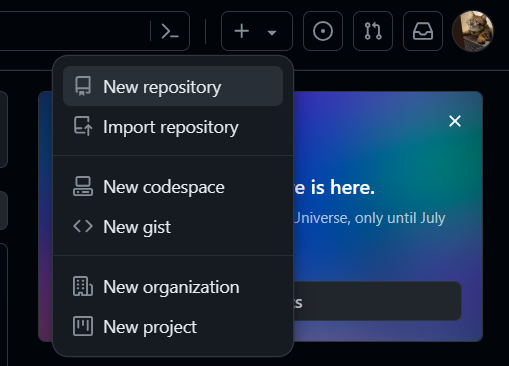
* Git — GitHub полностью интегрирован с Git, распределенной системой контроля версий. Это позволяет разработчикам отслеживать изменения в коде, создавать ветки для параллельной разработки, вносить правки и объединять их в основную ветку.
* Ветки — это разные версии кода проекта, которые позволяют работать над различными функциями или исправлениями параллельно.
* Pull Requests — это запросы на внесение изменений в код проекта. Они позволяют разработчикам обсуждать и проверять предлагаемые изменения перед их интеграцией в основную ветку.

1. **Каким образом создаётся локальный репозиторий на базе существующего удалённого репозитория GitHub?**

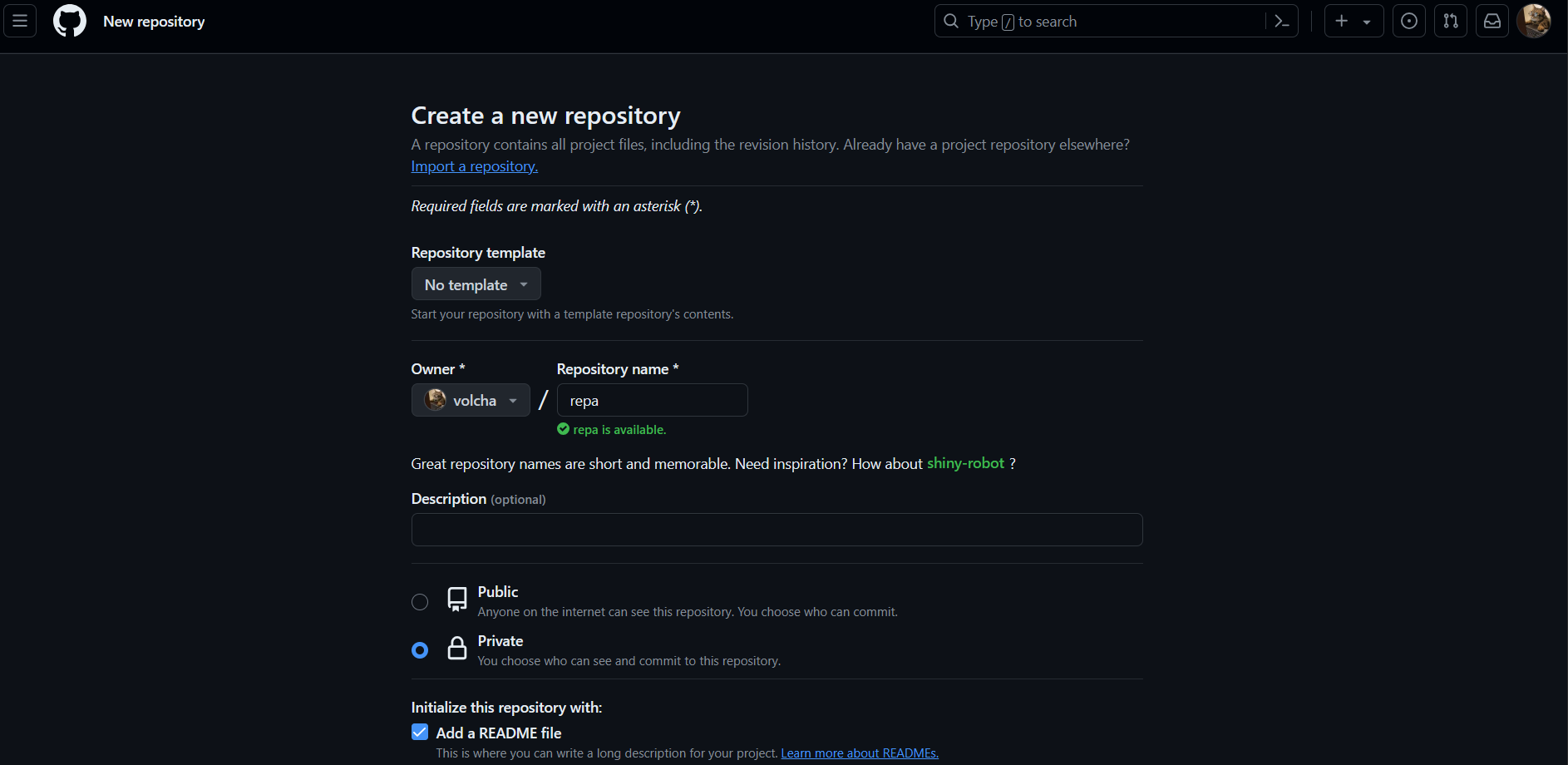
* Заходим в терминале с помощью cd в папку, где будет храниться репозиторий
* копируем HTTPS адрес удалённого репозитория из GitHub
* В терминале вводим: git **clone** <your repository name>
* С помощью cd проваливаемся внутрь проекта

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

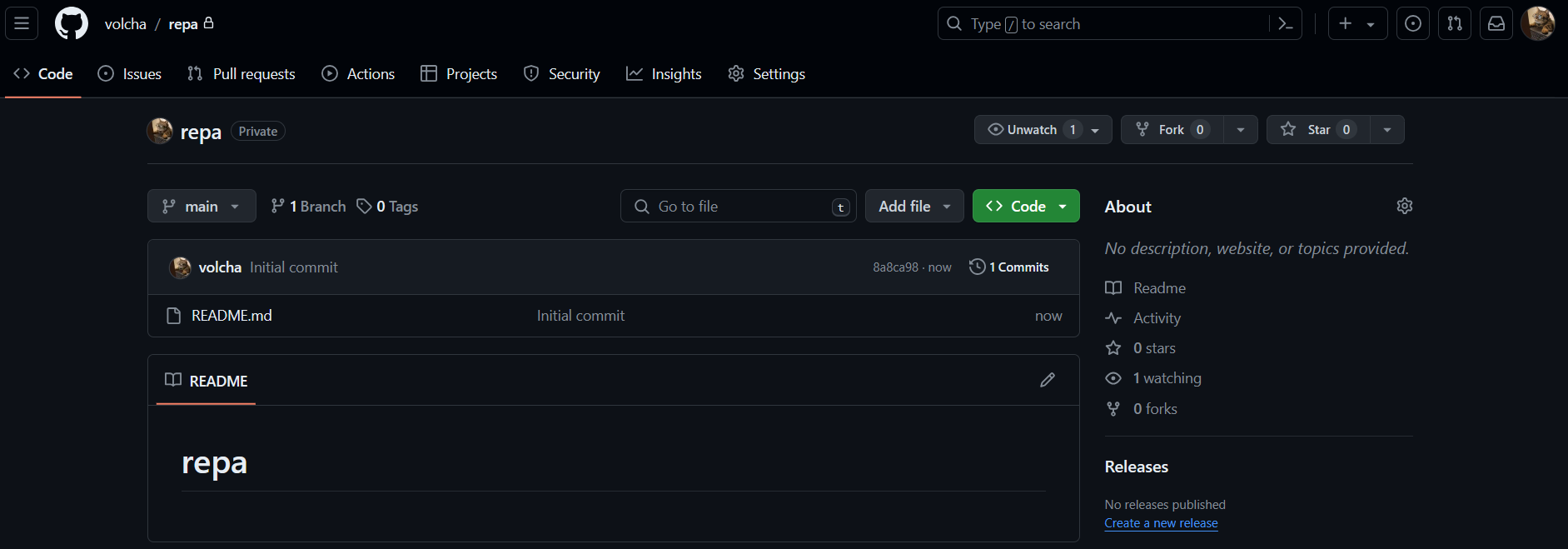
На странице своего профиля в GitHub нажать на New repository

****

Задать параметры, такие как название, добавление файла ля описания проекта, приватность:



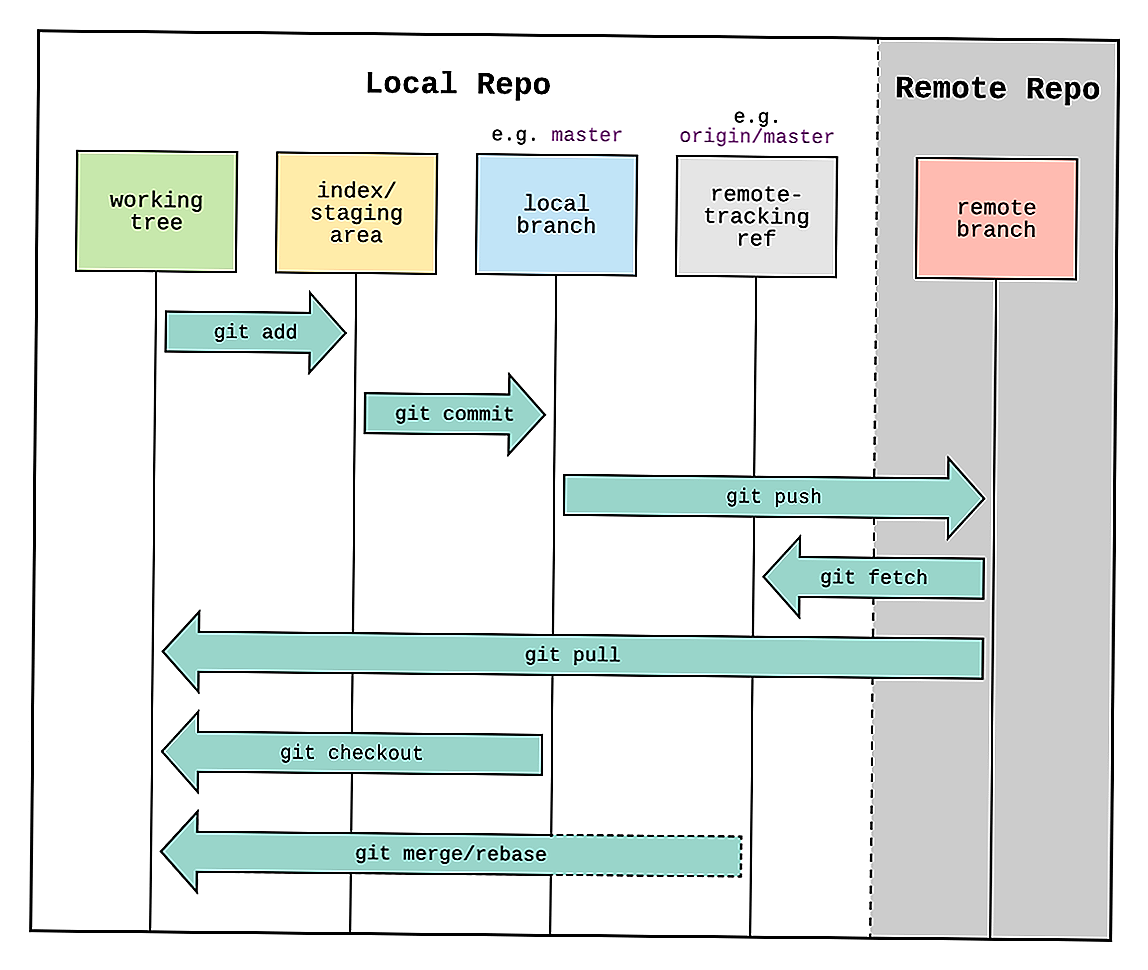
Репозиторий создан:



Преимущества создания системного репозитория в GitHub при работе в команде включают:

* Централизованное хранение — все файлы и конфигурации хранятся в одном месте, что облегчает доступ к ним.
* Общий доступ для команды — участники команды могут легко получить доступ к файлам, клонировать репозиторий и использовать его в своей работе.
* Управление версиями и история изменений — GitHub обеспечивает контроль версий, позволяя отслеживать изменения, вносить комментарии к ним и возвращаться к предыдущим версиям при необходимости.

**Задача 1: Создайте потоковую диаграмму для типичного рабочего процесса разработки программного обеспечения с использованием GitHub, от начала работы до завершения**

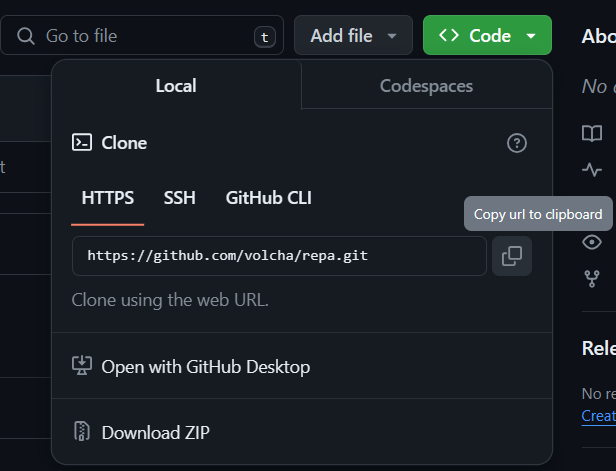


**Задача 2: Проведите практическое упражнение по созданию локального репозитория на вашем компьютере, связанного с удалённым репозиторием на GitHub, и продемонстрируйте основные этапы этого процесса**

Заходим в папку, куда будем клонировать удалённой репозиторий, с помощью командной строки



Копируем адрес репозитория, созданного в вопросе 5



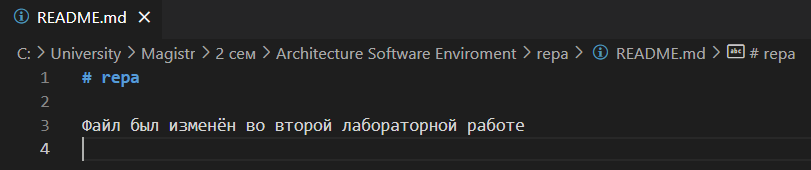
В терминале вводим строку, чтобы склонировать репозиторий локально



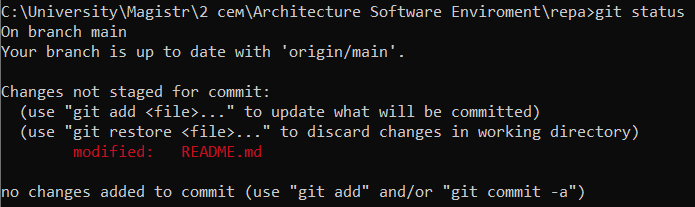
Провалимся внутрь проекта



В папке лежит только README файл, добавим туда информацию



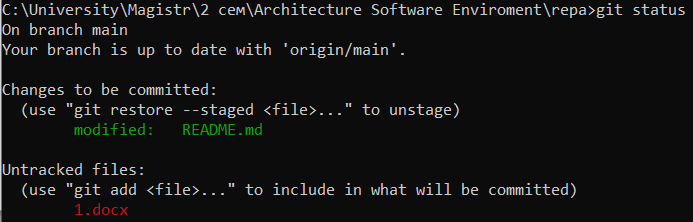
Git пишет, что файл был изменён



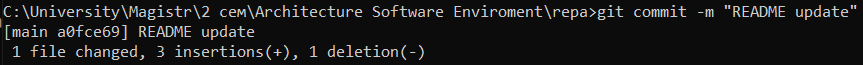
Фиксируем, что нам нужно изменить именно файл README



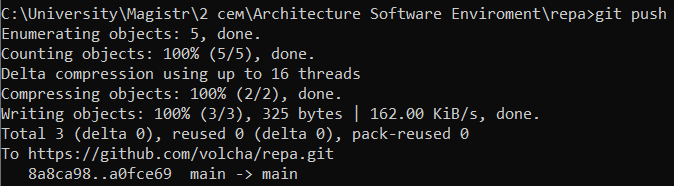
Также добавим ещё текстовый файл, чтобы показать, что он не был зафиксирован, а значит не будет закомичен

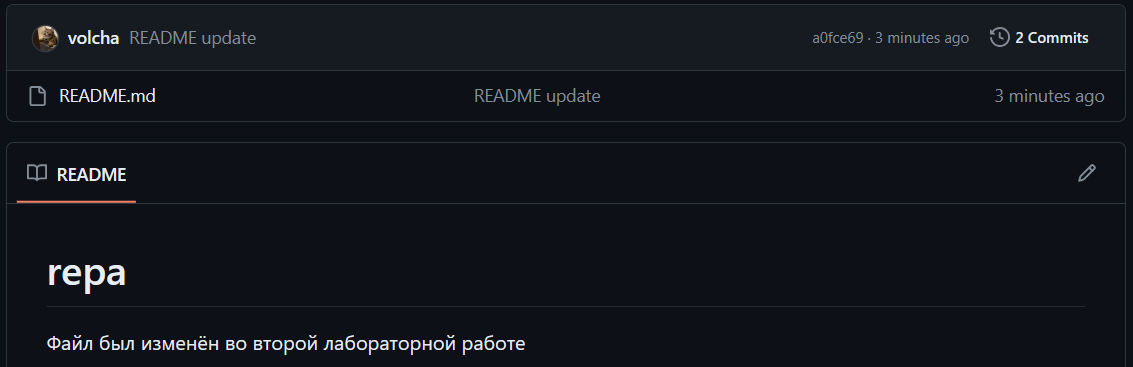


Закоммитим



И запушим на GitHub





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

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

Профайл в Visual Studio Code (VS Code) - это набор настроек и параметров, определяющих окружение и поведение редактора для конкретного пользователя. В контексте работы с Python, профайл позволяет настроить среду разработки с учетом индивидуальных предпочтений и потребностей разработчика, включая выбор и настройку расширений для работы с Python, отображение и темы оформления, а также настройку средства автодополнения и подсказок. Позволяет делить данной настройкой с другими пользователями.

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

* Общий профайл (Default Profile) — этот профайл используется по умолчанию и предоставляет базовые настройки для работы с любым языком программирования, включая Python. Он включает в себя основные функции редактора и поддержку расширений.
* Python профайл — для работы с Python можно также создать специальный профайл, который будет содержать дополнительные настройки и расширения, оптимизированные для работы с этим языком. Например, этот профайл может включать в себя расширение Python для Visual Studio Code, интегрированные средства отладки, автодополнение и другие инструменты, специфические для Python.
* Python Data Science — этот профайл в Visual Studio Code предназначен для разработки в области науки о данных с использованием Python. Он включает специальные расширения и инструменты, оптимизированные для работы с данными, их анализа, машинного обучения и визуализации.

Каждый вид профайла в VS Code для работы с Python имеет свои особенности и настройки, которые позволяют адаптировать среду разработки под конкретные потребности и предпочтения разработчика.

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

* Использование профайлов при разработке программного обеспечения с Python и VS Code позволяет стандартизировать окружение разработки, обеспечивая единые настройки и конфигурации для всех участников команды.
* Упрощает вход новых разработчиков в проект.
* Уменьшает количество ошибок из-за несовместимых настроек у разных участников

1. **Какова типичная структура профайла в VS Code для работы с Python, и какие основные настройки обычно включаются в такой профайл?**

Профайл в VS Code включает в себя следующие элементы:

* Настройки — в файле settings.json.
* Расширения — список расширений, включённых в текущий профайл.
* Привязки клавиш — в файле keybindings.json.
* Snippets — в файлах {language}.json.
* Пользовательские задачи — в файле tasks.json.

Основные настройки профайла в Visual Studio Code для работы с Python включают:

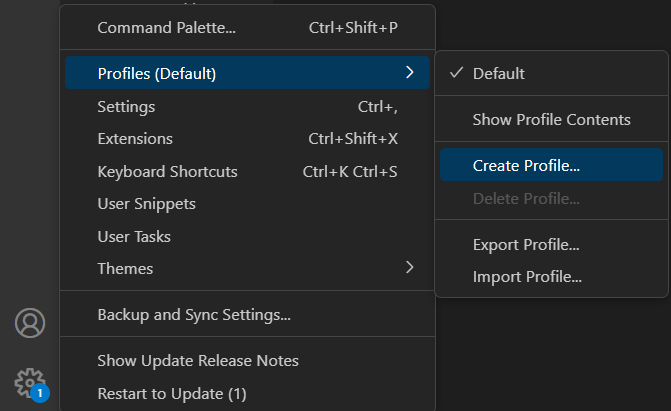
* Интерпретатор Python — определяет версию для проекта.
* Библиотеки и пакеты — определяет зависимости проекта и используемые библиотеки.
* Настройки автодополнения кода — Параметры автодополнения кода, такие как импорт модулей и подсказки по функциям. Это улучшает удобство и быстроту написания кода.
* Форматирование кода — Правила форматирования кода, такие как отступы и расстановка скобок. Это обеспечивает единообразие и читаемость кода.
* Правила проверки кода на наличие ошибок и потенциальных проблем — это помогает выявлять и исправлять ошибки на ранних этапах разработки.
* Запуск и отладка — параметры запуска и отладки кода, такие как точки останова и режим отладки.

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

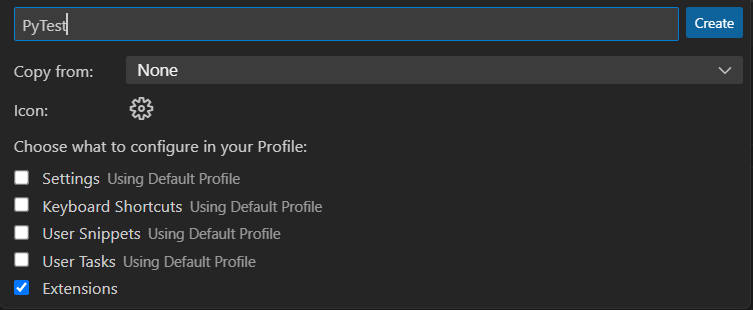
* Выбор эффективного интерпретатора — используйте легковесные и быстрые версии интерпретатора Python, такие как Python 3.8 или более поздние версии.
* Установка необходимых расширений — установите только необходимые расширения для работы с Python, чтобы избежать лишней нагрузки на ресурсы системы и улучшить производительность.
* Использование виртуальных сред — используйте виртуальные среды Python для изоляции зависимостей проекта и оптимизации использования памяти и ресурсов.
* Горячие клавиши — настройте горячие клавиши для часто используемых действий.
* Настройка автоматического форматирования — включите автоматическое форматирование кода при сохранении файлов. Это позволяет поддерживать единый стиль кодирования и улучшает читаемость кода.

**Задача 1: Создайте профиль в VSC для работы с Python**

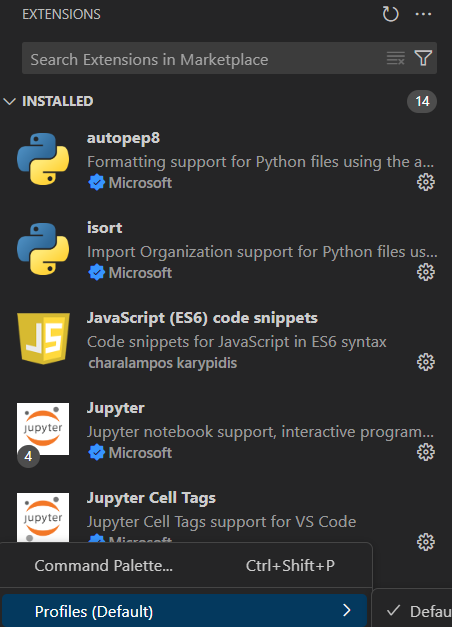
Нажмём на “создать профиль”



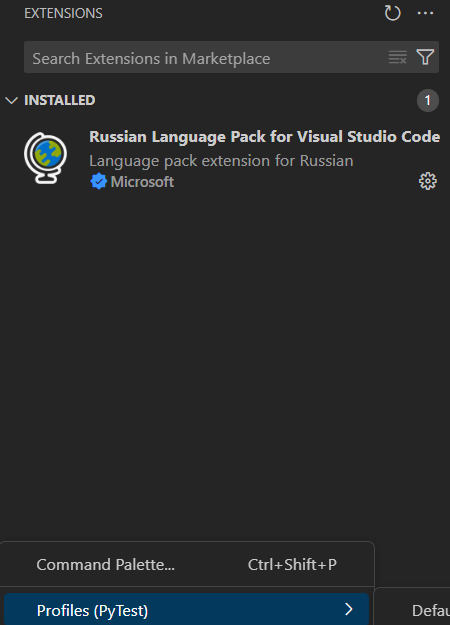
Назовём его PyTest



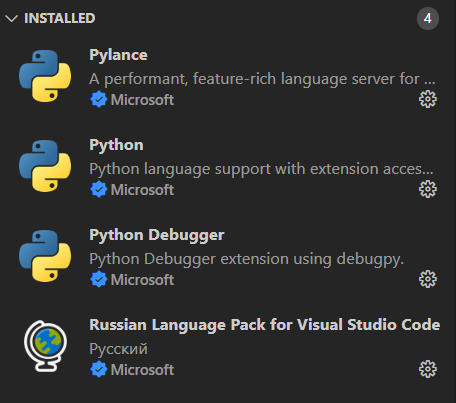
Можно заметить, что в Дефолтном профиле уже скачаны различные разрешения



А в PyTest почти ничего нет



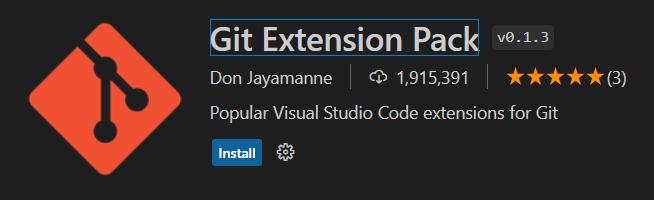
В новый профиль установим нужные для него расширения



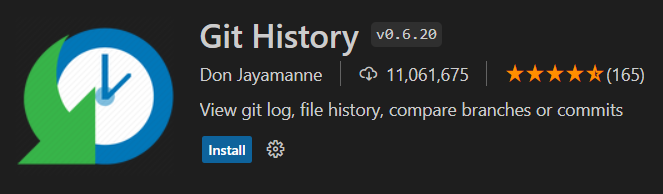
Таким образом, для разных задач и языков можно создавать различные профили, заточенные для конкретных целей

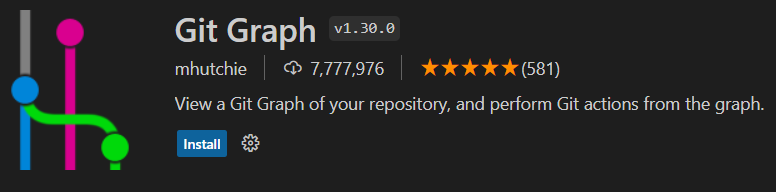
**Задача 2: Разработайте структуру профиля в VSC, которая бы учитывала требования к распределённой разработке, включая настройки для коллективной работы над проектом, контроля версий и совместном редактировании кода**

Для этого можно скачать такие расширения как



Для работы с гитом





Для удобного отслеживания изменений

**Лабораторная 4. Разработка потоковых диаграмм их реализация в среде VS Code.**

1. **Что представляют собой потоковые диаграммы, и какие основные элементы включает типичная потоковая диаграмма?**

Потоковая диаграмма данных (Data Flow Diagram) - это диаграмма, которая отображает потоки данных между системами и базами данных.

DFD Нотация содержит следующие объекты:

* Процесс (Process). Последовательность действий, которые нужно предпринять, чтобы данные были обработаны. Например “Регистрация клиента” или  “Обработать заказ”.
* Внешние сущности (External Entity). Объекты, которые не входят в систему, но являются для неё источником, либо получателем данных. Например клиентский сайт.
* Хранилища данных (Data Storage). Внутренние хранилища данных, в которых сохраняется / извлекается информация, используемая в процессах. Например хранилище заявок или данных о клиентах.
* Поток данных (Data flow). На диаграмме показывает направление движения данных.

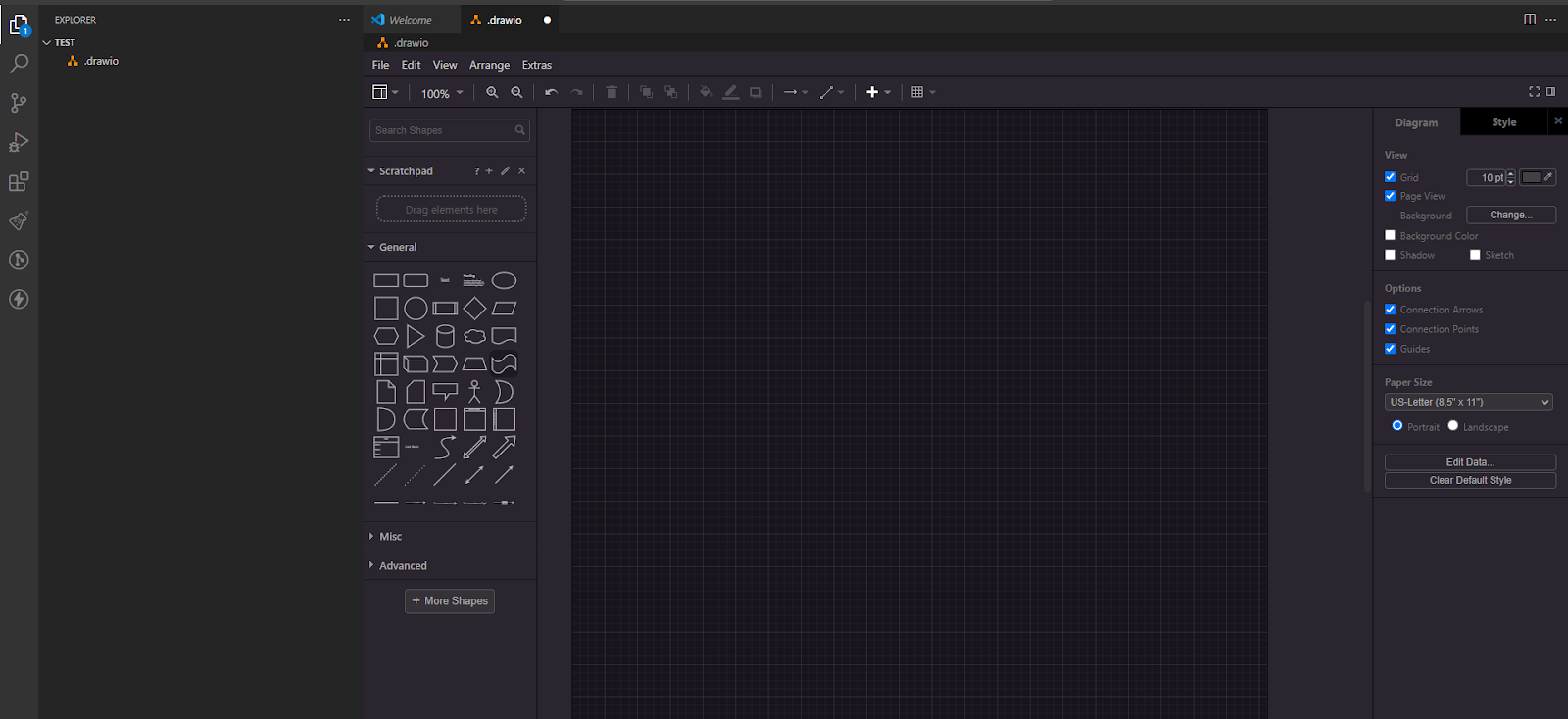
1. **Какова роль разработки потоковых диаграмм в процессе создания программного обеспечения, какие преимущества они предоставляют разработчикам?**

Потоковая диаграмма визуализирует поток данных в системе. Это облегчает понимание, поддержку и разработку системы. С её помощью можно увидеть архитектурные проблемы и все взаимодействия модулей. Также позволяет поделиться своим видением с другими участниками проекта.

1. **Как можно создать потоковую диаграмму в среде Visual Studio Code (VS Code), и какие инструменты или расширения могут быть использованы для этого?**

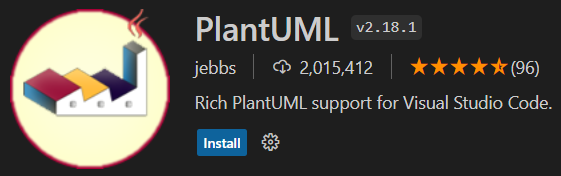
VSC имеет множество инструментов по созданию диаграмм. Диаграммы можно создавать в визуальном редакторе, манипулируя самими объектами диаграммы.

Например, есть расширение “Draw.io Integration”, которое интегрирует сервис draw.io в VSC:



Но, на практике, бывает быстрее и удобнее писать формальные описания диаграмм в виде текста. Самым популярным инструментом является PlantUML, который автоматически генерирует UML диаграммы из кода.

В VS Code есть расширение PlantUML.



1. **Что такое диаграмма взаимодействий в рамках UML (Unified Modeling Language), и какие основные виды взаимодействий она позволяет моделировать?**

UML (Unified Modeling Language - Унифицированный язык моделирования). UML предоставляет набор правил для создания моделей.

Диаграмма взаимодействий в рамках Unified Modeling Language (UML) - это графическое представление взаимодействия между объектами системы.

Основные виды взаимодействий, которые UML позволяет моделировать:

* Диаграмма классов — описывает общую структуру иерархии классов системы, их атрибуты, методы, интерфейсы и взаимосвязи.
* Диаграмма развертывания — показывает программные компоненты и их взаимосвязи. Предлагает наглядное представление, где именно развернут каждый программный компонент.
* Диаграмма состояний — описывает поведение сущности, и то, как оно изменяется в зависимости от внешних и внутренних событий.
* Диаграмма деятельности — описывает процесс от начала и до его конца. Представляет из себя набор операций и то, как каждое действие ведёт к следующему.
* Диаграмма последовательности — показывает последовательность сообщений между объектами в хронологическом порядке.

1. **Каким образом диаграммы взаимодействий могут быть полезны при проектировании и анализе систем на уровне UML?**

Диаграммы взаимодействия имеют следующие плюсы:

* Визуализация взаимодействий между объектами — они позволяют разработчикам наглядно представить, как объекты в системе взаимодействуют друг с другом.
* Понимание порядка выполнения — особенно важно при разработке сложных систем, где порядок операций критичен.
* Анализ требований — диаграммы взаимодействий могут использоваться для проверки того, соответствует ли разрабатываемая система функциональным требованиям.
* Облегчение коммуникации между участниками проекта — диаграммы предоставляют удобный способ общения между разработчиками, аналитиками, тестировщиками и другими заинтересованными сторонами.
* Обнаружение проблем в проектировании — при рассмотрении диаграмм взаимодействий можно обнаружить ошибки или недостатки в системе.
* Использование в тестировании — диаграммы взаимодействий могут служить основой для разработки тестовых сценариев.

**Задача: Постройте диаграмму взаимодействий в UML, моделирующую обмен сообщениями между пользователем и системой при регистрации нового пользователя в веб-приложении.**

@startuml Процесс регистрации пользователя

actor "Клиент" as Client

entity "Сервис регистрации" as Service

database "База данных" as Database

Client -> Service: Отправить данные регистрации (имя, email, пароль)

activate Service

    Service -> Database: Проверить, существует ли пользователь

    activate Database

        Database -> Database: Поиск в базе данных

        Database -> Service: Результат проверки

    deactivate Database

    alt Если пользователь уже существует

        Service -> Client: Ошибка: пользователь уже существует

    else Если новый пользователь

        Service -> Database: Сохранить нового пользователя

        activate Database

            Database -> Database: Запись данных пользователя

            Database -> Service: Подтверждение сохранения

        deactivate Database

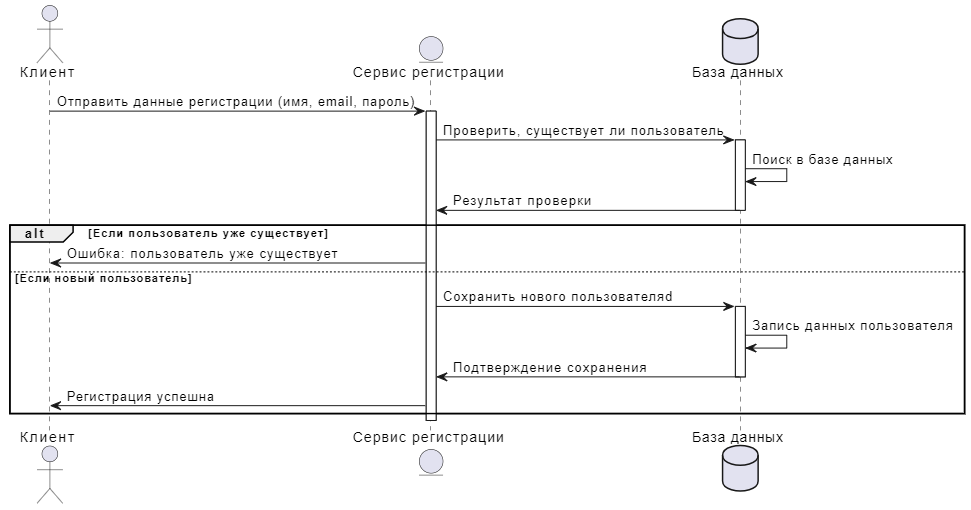
        Service -> Client: Регистрация успешна

    end

deactivate Service

@enduml

Зарендерим диаграмму, нажав alt+D



Здесь видно главное преимущество PlantUML по сравнению с ручным построением диаграммы: процесс описан в виде кода, его можно легко модифицировать, даже для сложных диаграмм.

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

1. **Как создать новую ветвь (branch) в репозитории на GitHub c использованием командной строки и через веб-интерфейс?**

С использованием командной строки:

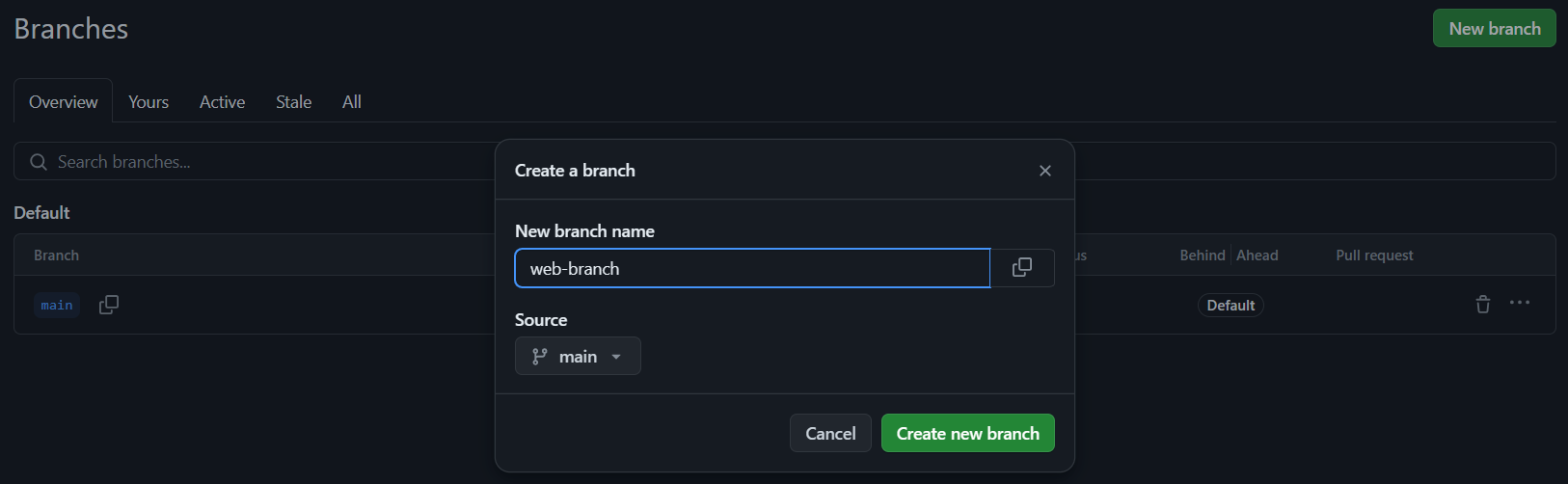


Через веб-интерфейс:

Зайти в Branch



New branch --> имя ветви --> Create new branch



1. **Какие основные принципы следует учитывать при работе с ветками Git для эффективной распределённой разработки?**

* Частые и небольшие коммиты — чтобы упростить отслеживание изменений, сократить вероятность конфликтов при слиянии и обеспечить более гибкую историю разработки.
* Основная ветка (обычно называется main или master) — должна отражать стабильное состояние проекта. Все изменения должны быть интегрированы в основную ветку только после тщательного тестирования и проверки качества кода.
* Создание отдельных веток для функциональности — для каждой новой функциональности или задачи создайте отдельную ветку.
* Регулярное слияние и обновление — регулярно сливайте изменения из основных веток в рабочие ветки, чтобы поддерживать их актуальность и избегать накопления большого количества изменений.

1. **Как осуществить слияние (merge) ветвей в Git на GitHub, чтобы объединить изменения из одной ветви в другую?**

git checkout <имя ветки, в которую нужно залить код (например, develop)>

git merge <имя\_ветки, откуда нужно взять код (например, origin/develop)>

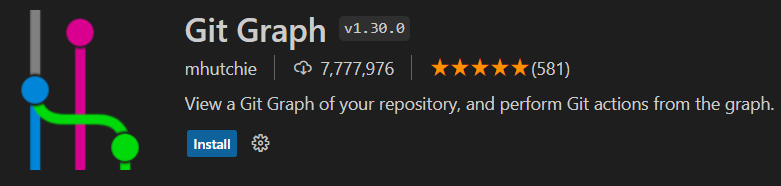
1. **В чём заключается понятие конфликтов слияния (merge conflicts) при объединении ветвей, и как их решить с помощью среды VSC?**

Конфликты слияния (merge conflicts) возникают при попытке объединить две ветки, когда Git не может автоматически определить, какие изменения следует применить. Это может произойти, когда две ветки внесли изменения в одну и ту же часть файла или когда одна ветка удалила файл или строку, которую другая ветка изменила.

Для решения конфликтов слияния в среде Visual Studio Code (VS Code) следуйте этим шагам:

1. **Открытие файла с конфликтом**: В редакторе VS Code перейдите в раздел "Source Control" (или используйте команду "Ctrl+Shift+G"), где будут отображены файлы с конфликтами. Щелкните по файлу с конфликтом для открытия его в редакторе.
2. **Визуальное разрешение конфликта**: VS Code поможет вам визуально разрешить конфликты, отображая их в специальном режиме. Вам будет предложено выбрать, какие изменения сохранить: из текущей ветки (HEAD), из ветки, которую вы сливаете (Incoming Change), или оба варианта. Выберите нужные изменения, удалив лишние метки "<<<<<<<", "=======" и ">>>>>>>", а затем сохраните файл.
3. **Пометка конфликтов как разрешенных**: После разрешения конфликтов отметьте файл как разрешенный для слияния, нажав на кнопку "Mark as Resolved" в панели инструментов Source Control.
4. **Завершение слияния**: После разрешения всех конфликтов и пометки файлов как разрешенных завершите слияние, выполните коммит и, если необходимо, отправьте изменения на удаленный репозиторий.
5. **Какие инструменты предоставляет VSC для удобного отслеживания и управления ветками Git при работе над проектом на GitHub?**

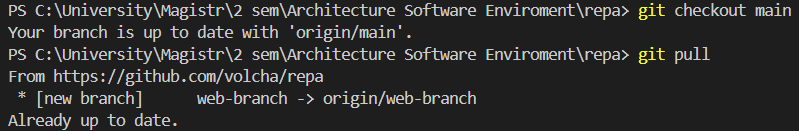
Такое расширение как Git Graph. Это визуальное представление истории коммитов, ветвей и слияний в репозитории Git



**Задача 1: Создайте новую ветвь в вашем репозитории на GitHub, добавьте в неё изменения через VSC, а затем выполните слияние этой ветви с основной ветвью master/main**

Новая ветка в репозитории “web-branch” была создана в вопросе 1.

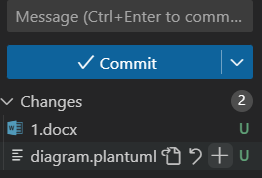
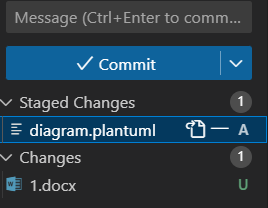
Загружаем ветку



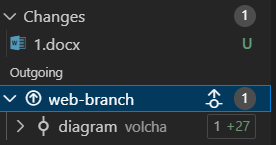
Переключаемся на неё



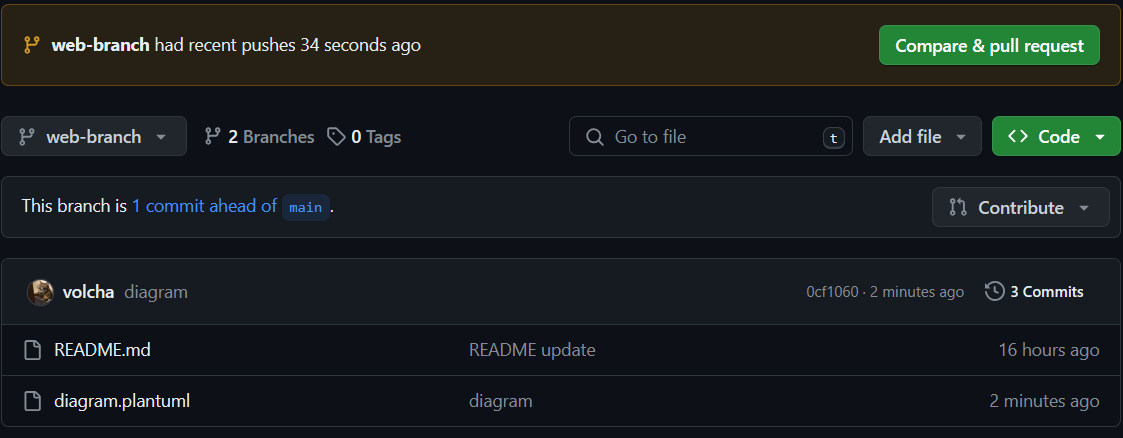
Выбираем код с диаграммой для добавления его в новую ветку

Делаем коммит



И пушим в GitHub

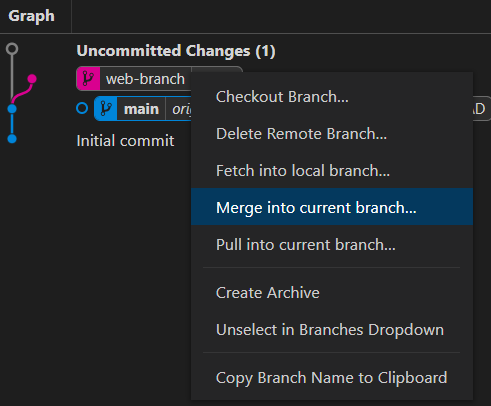


Файл добавился в GitHub

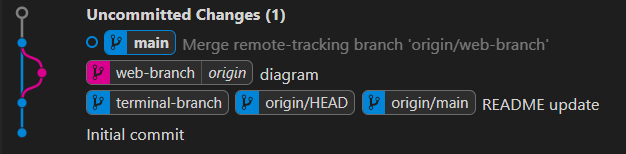
Чтобы сделать слияние с main можно: перейти в ветку main



И нажать

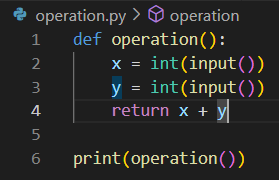


Ветки слиты

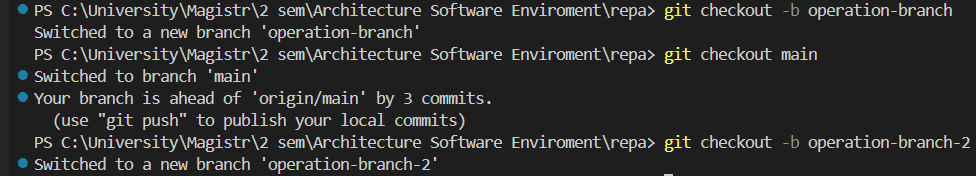


**Задача 2: Разрешите конфликты слияния двух веток в вашем проекте, используя возможности сравнения и редактирования кода в VSC для обеспечения успешного объединения изменений**

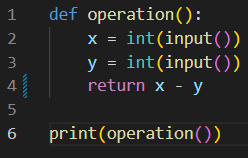
Добавим файл operation.py в main ветку и закоммитим его



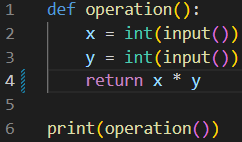
Создаём 2 ветки

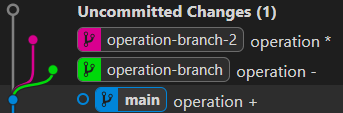


Перейдём в первую и изменим там код и закомитим его в этой ветке

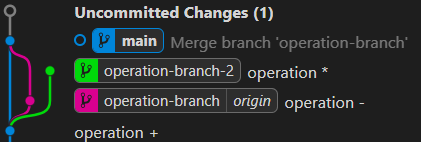


Изменим код во второй ветке

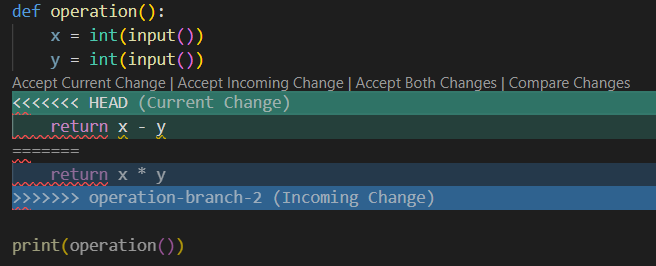




merge изменения из первой ветки в main



merge изменения из второй ветки в main. Возникает конфликт:



Оставим изменения из второй ветки

