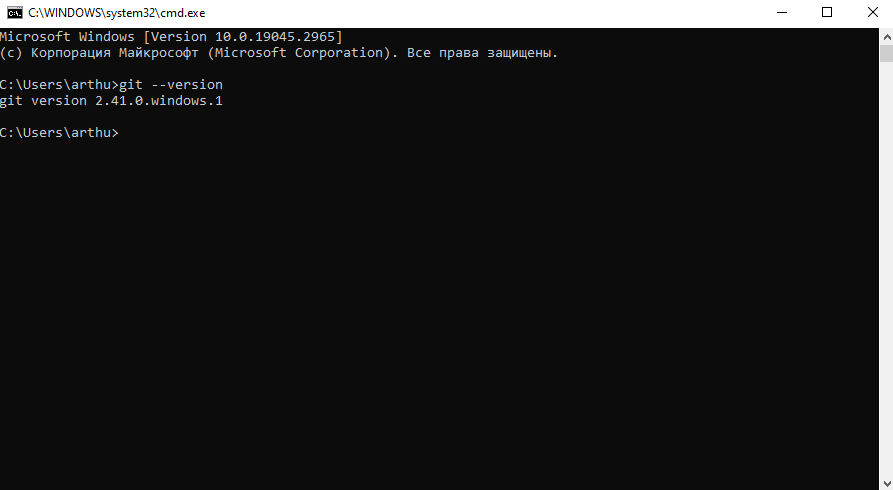
**Установка GIT**

<https://git-scm.com/> - скачиваем файл

При установки все параметры по умолчанию

Проверяем установился ли git -> win+r -> cmd (командная строка) ->

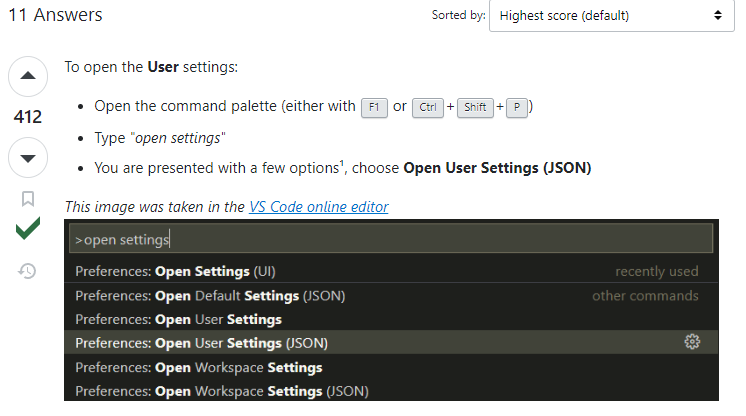


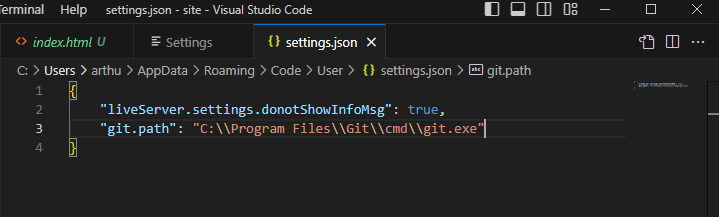
**Создание локального репозитория**

**git init** – создание репозитория (в командной строке git bash)

Но команды по началу не работают в терминале редактора VSCode. Для этого нужно в файле **settings.json** прописать путь к git.exe (Путь:)

Открытие файла settings.json





**Облако для репозиториев**

Можно использовать GitHub и BitBucket

**GitHub**

* Пишем код -> Далее необходимо выполнить commit т.к. файлы имеют статус modified (красный).

**git add .** – добавление к отправке всех файлов или

**git add “index.html”** – конкретный файл

После этой команды статус modified (зеленый)

**git commit -m "first commit"** – написали коммит

После этой команды файлы из статуса пропадут.

* Необходимо установить связь между нашим git и github. Обычно, когда создаём новый репозиторий в аккаунте, нам предлагают список команд для установления связи. Одна из этих команд предтавлена:

git remote add origin <https://github.com/levashoffffff/gittest.git>

* Заливаем наш проект в ветку мастер

**git push -u origin master**

**git push** – отправка файлов

* В GitHub есть возможность добавлять людей у которых появится доступ к нашему репозиторию.

Settings -> Collaborators -> Add people -> После того как добавили, скидываем ссылку другому пользователю. Он по ней проходит.

**Загрузка репозитория**

Для того чтобы загрузить к себе в проект репозиторий необходимо использовать команду:

git clone <https://github.com/jquery/jquery.git> - команда и ссылка на репозиторий

git clone <https://github.com/jquery/jquery.git> -b branch-name - Этот вариант склонирует ПОЛНОСТЬЮ весь репозиторий и просто переключится на ветку branch-name:

git clone --branch=branch-name <http://whatever.git> – клонирование конкретной ветки

**Статусы в git**

**git status** – команда для просмотра статуса

**git pull** – подгрузка всех изменений из облака, которые внесли наши коллеги. ПЕРЕД НАЧАЛОМ РАБОТЫ НАД ПРОЕКТОМ. КЛОНИРУЕМ ТОЛЬКО В ПЕРВЫЙ РАЗ, А ЗАТЕМ ПРОСТО git pull

**Создание алиасов (собственные команды)**

Пусть: C->Пользователи-> artur-> .gitconfig - в этом файле можем писать свои алиасы

[alias]

s = status --short

st = status

l = log --oneline --graph --decorate --all

g = log --graph --abbrev-commit --decorate --all --format=format:'%C(bold blue)%h%C(reset) - %C(bold cyan)%aD%C(dim white) - %an%C(reset) %C(bold green)(%ar)%C(reset)%C(bold yellow)%d%C(reset)%n %C(white)%s%C(reset)'

br = branch

co = checkout

**Отмена изменений и возвращение к предыдущему коммиту**

* Эти команды используются, когда мы не использовали команду git add .

**git checkout -- index.html** – отмена всех изменений в файле index.html

**git checkout .** – отмена всех изменений в файлах

* Команды после применения команды git add . Т.е. сначала вытаскиваем файл из стэйдж, а потом делаем отмену изменений

**git reset index.html**

**git reset .**

После этой команды файл выпал из стэйдж и теперь можем применить

**git checkout -- index.html**

* Если сделали коммит

**git reset --hard HEAD^1** – вернуться на один коммит назад и удалили все изменения

**git reset --soft HEAD^1** – вернуться на один коммит назад и оставить все изменения, но если хотим убрать изменения, то необходимо вытянуть файл из стэйдж и отменить действия

git reset .

git co -- .

co = checkout – alias

**Работа с ветками git branch**

**git branch** – показывает ветку

**git branch –v** – показывает ветку и последний коммит сделанный в этой ветке

**git branch develop** – создали новую ветку develop. Все файлы остались в прежнем состоянии. Когда мы начнем делать коммиты в этой ветке и затем перейдем обратно в ветку мастер, то изменений мы не увидим, т.к. они в ветке develop

* По умолчанию мы находимся в ветке \*master

**git checkout develop** – переходим в ветку develop

**git checkout –b about** – создали новую ветку about и сразу в нее перешли

**git br –m myfix** – переименовываем ветку ту на которой находимся

**git co c070744** – переход к нужному коммиту. С этого коммита мы можем создавать новые ветки

**git br –d develop** – удаление ветки develop, но в момент удаления мы не должны находится в этой ветке

**git push --set-upstream origin new** – заливаем ветку new на github, для того чтобы ее залить нужно на ней находиться

**Создание gist (фрагменты кода)**

Необходимо Создать gist в github

1. Устанавливаем расширение "gist" в "Visual Studio";
2. Заходим на главную своего профиля в Github, то есть "https://github.com/settings/profile" далее "Developer sesttings" далее "Personal acces tokens" выбираем "generate new token", Генерируя токен, нужно поставить галочку на gist. и копируем наш сгенерированный код(token);
3. Заходим в "Visual Studio", жмем F1, вводим "gist" и выбираем "GIST: select profile";
4. Вводим в появившееся окно наш сгенерированный TOKEN, далее вас попросят ввести имя пользователя, то есть ВАШЕ имя привязанное к аккаунту на github.
5. Проверить все ли работает ТЕПЕРЬ

Затем F1 -> gist: open gist -> и выбираем нужный гист -> код вставится в отдельный файл

F1 -> gist: insert …. -> выбираем нужный гист -> код вставится в наш файл рабочий на котором находимся

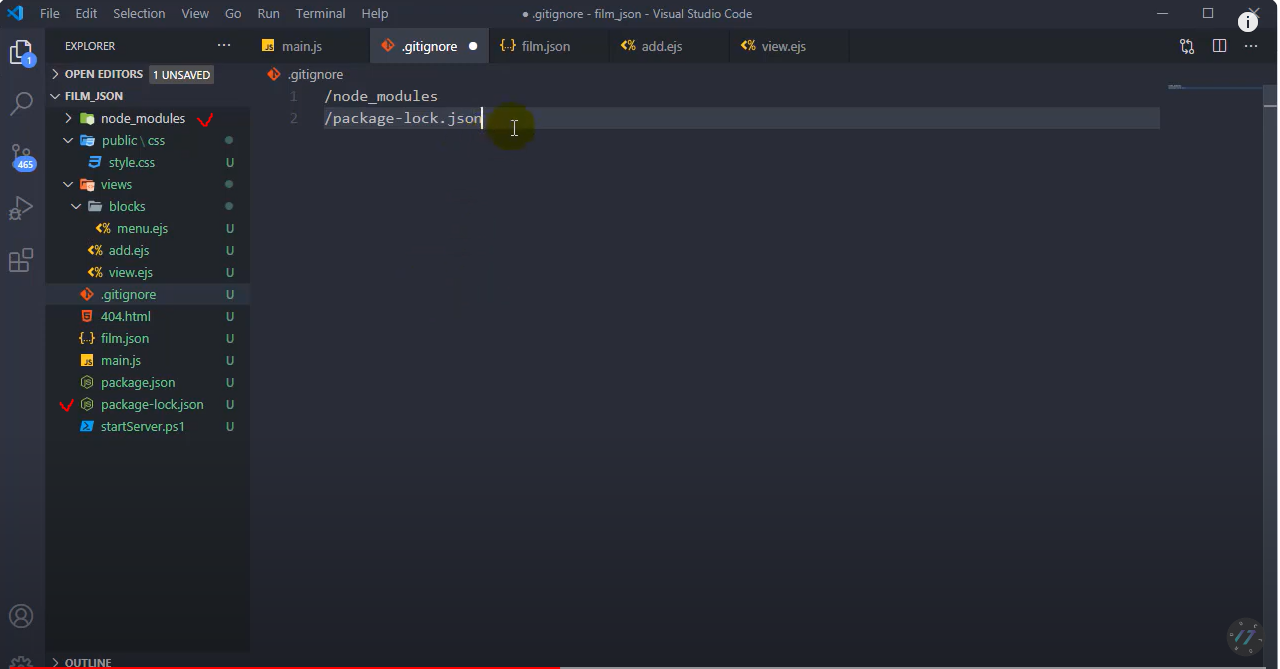
**Работа с панелью управления**

****

**Создание файла .gitignore**

С помощью файла .gitignore можно указать, что исключить из нашего репозитория. Те файлы и каталоги, которые укажем в этом файле НЕ БУДУТ push на сервер.

В данном случае исключили node\_modules и json файл.

****

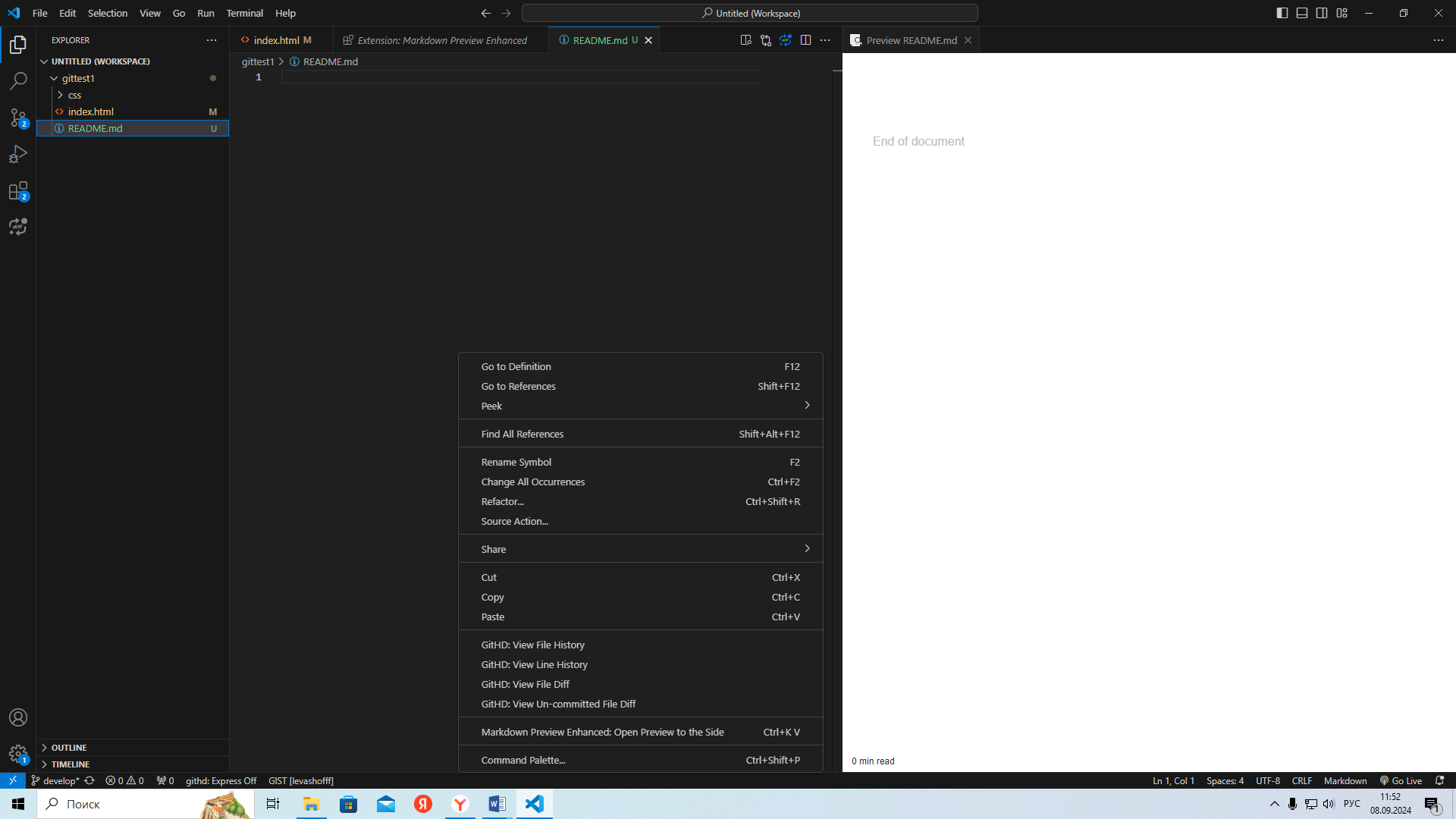
**Создание файла README.md синтаксис Markdown**

Язык разметки Markdown широко распространён в вебе. На нём пишут readme-файлы и документацию, а его принципы работы используют для оформления сообщений и публикаций в мессенджерах и социальных сетях.

Необходимо установить расширение в GitHub. **Markdown Preview Enhanced**

Это плагин позволяющий выполнить предпросмотр нашего .md файла перед тем как сделать push в наш репозиторий.

Создаем .md файл -> ПКМ -> Markdown Preview ….



Синтаксис

**# Привет**

**## Как дела**

\* Текст

   \* Текст 2

   \* Текст 3

- Текст

   - Текст 2

   - Текст 3

1. Текст

    1.2 Текст

    1.3 Текст

---

**\_\_Жирный\_\_**

\_\_\_

*\_Курсивный\_*

\*\*\*

*\_\_\_*Жирный курсивный\_\_\_

```

a + b = c

b + a = c

```

> Цитата

Продолжение

>> Вложенная цитата

[Ссылка на мой канал](https://otvet.mail.ru/question/70574361)

Сноска первая[^1] и вторая [^2]

[^1]: Текст сноски 1

[^2]: Текст сноски 2

![Ссылка на мой канал](logo.jpg)

[![Ссылка на мой канал](logo.jpg)](https://otvet.mail.ru/question/70574361)

цвет | количество | размер

:----|:----------:|-------:

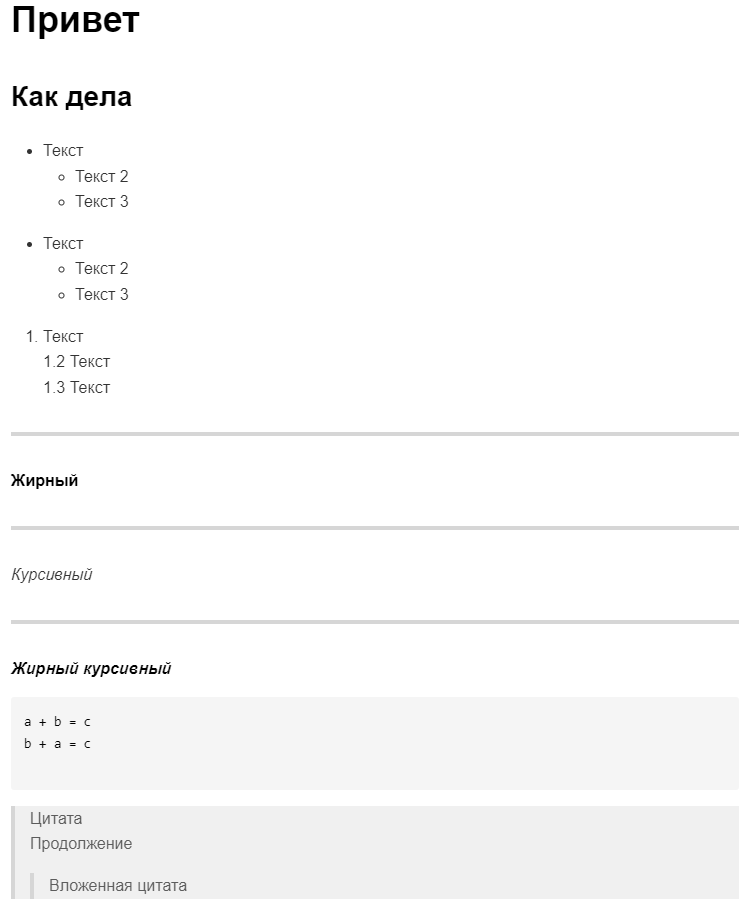
красный | 1 | 256

синий   | 2 | 300

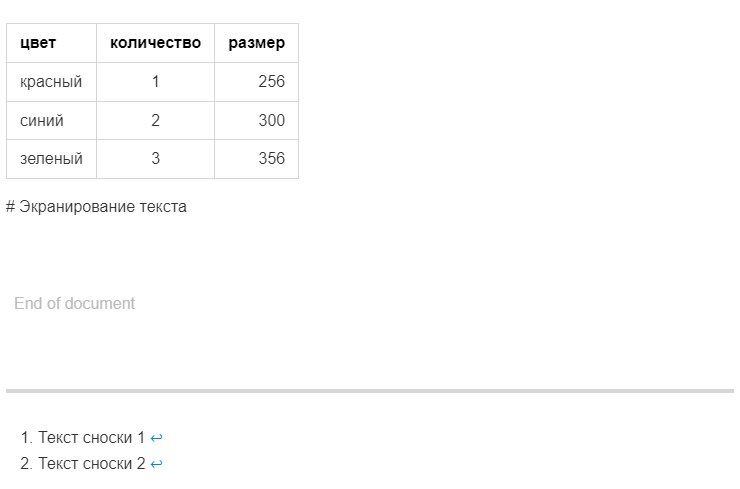
зеленый | 3 | 356

\# Экранирование текста

Результат







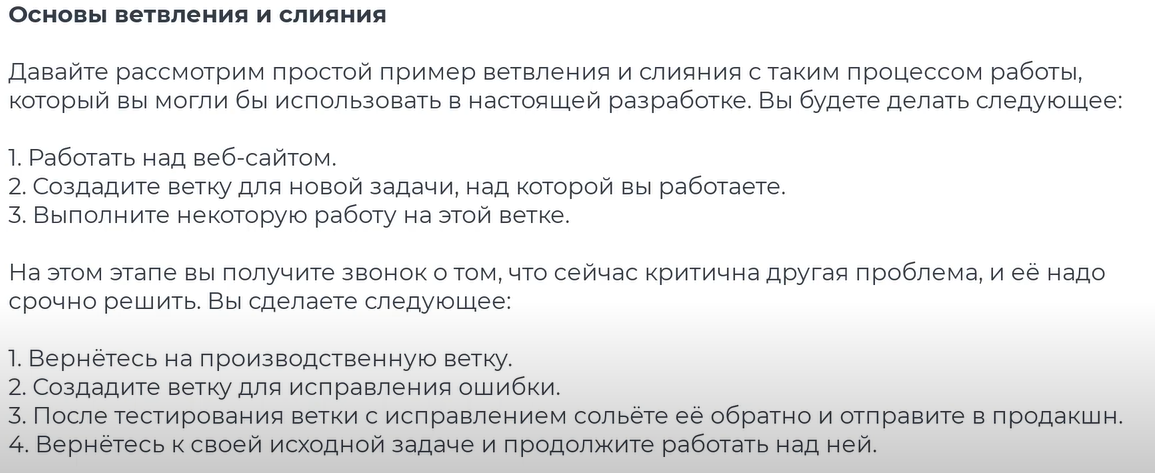
# Откат к коммиту, расширение Git Graph для VS Code

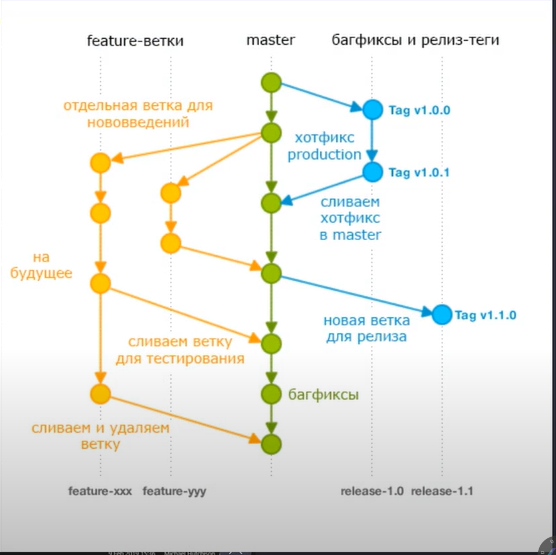
# Необходимо установить расширение Git Graph

# 

# Слияние веток Git, разрешение конфликта слияния и работа в VS Code, Git Graph и Терминале

Ветки нужны для того, чтобы отдельно от основного проекта выполнять какие-либо работы.





Нужно находится в той ветке которую хотим соединить

Git merge

Проблемы

1. Возникает ошибка 403 при попытке подключиться к хранилищу git. Т.е после того как клонирую репозиторий, должно выводиться диалоговое окно с вводом логина и пароля. Если этого не происходит, значит в панели управления в учетных данных уже установился логин и пароль другого пользователя, и он автоматически подставляется в форму ввода данных. Для решения этой проблемы: **Панель управления -> Учетные записи пользователей -> Управления учетными данными Windows –> Общие учетные данные (gitea) или другое любое хранилище. Там меняем логин и пароль на свой.**