Git для новичков (часть 1)

[Git\*](https://habr.com/ru/hub/git/)[GitHub\*](https://habr.com/ru/hub/github/)

Из песочницы

[*Часть 2*](https://habr.com/ru/post/542616/)

Что такое Git и зачем он нужен?

Git - это консольная утилита, для отслеживания и ведения истории изменения файлов, в вашем проекте. Чаще всего его используют для кода, но можно и для других файлов. Например, для картинок - полезно для дизайнеров.

С помощью Git-a вы можете откатить свой проект до более старой версии, сравнивать, анализировать или сливать свои изменения в репозиторий.

Репозиторием называют хранилище вашего кода и историю его изменений. Git работает локально и все ваши репозитории хранятся в определенных папках на жестком диске.

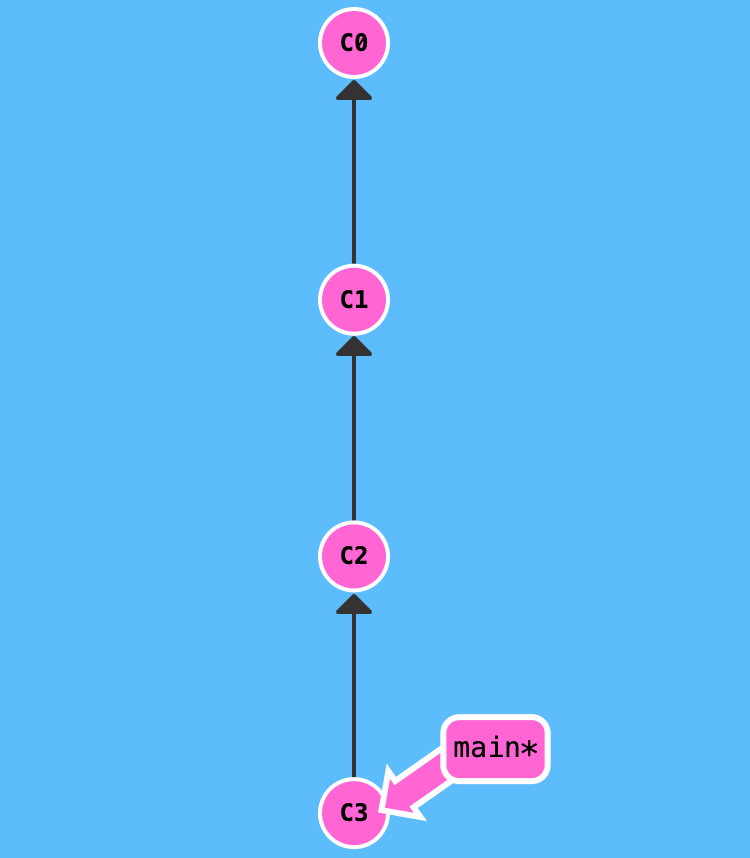
Так же ваши репозитории можно хранить и в интернете. Обычно для этого используют три сервиса:

* [GitHub](https://github.com/)
* [Bitbucket](https://bitbucket.org/)
* [GitLab](https://gitlab.com/)

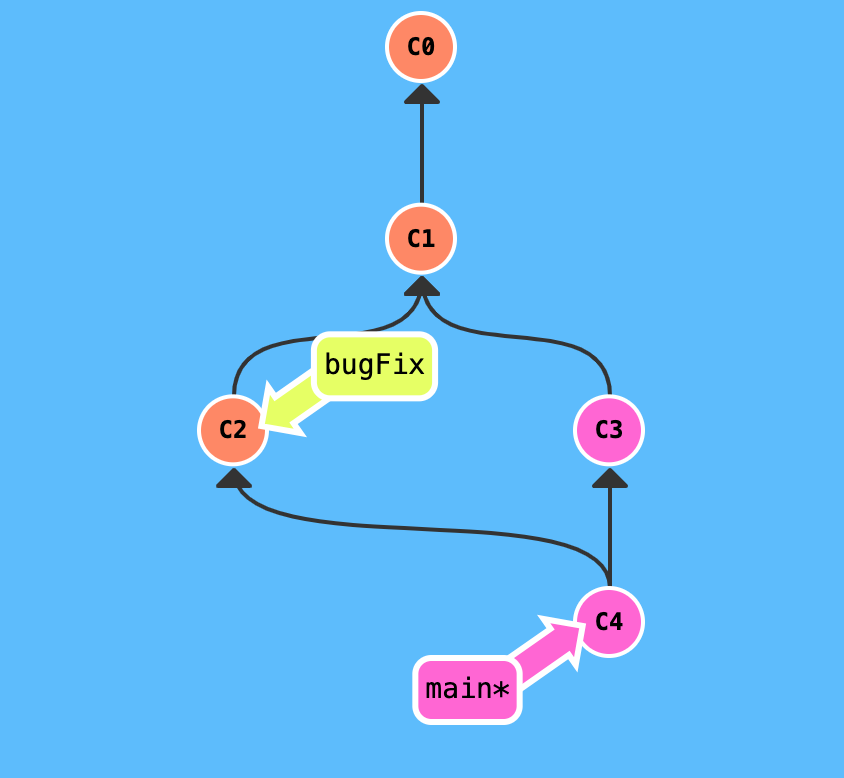
Каждая точка сохранения вашего проекта носит название коммит (commit). У каждого commit-a есть hash (уникальный id) и комментарий. Из таких commit-ов собирается ветка. Ветка - это история изменений. У каждой ветки есть свое название. Репозиторий может содержать в себе несколько веток, которые создаются из других веток или вливаются в них.

Как работает

Если посмотреть на картинку, то становиться чуть проще с пониманием. Каждый кружок, это commit. Стрелочки показывают направление, из какого commit сделан следующий. Например C3 сделан из С2 и т. д. Все эти commit находятся в ветке под названием main. Это основная ветка, чаще всего ее называют master . Прямоугольник main\* показывает в каком commit мы сейчас находимся, проще говоря указатель.



В итоге получается очень простой граф, состоящий из одной ветки (main) и четырех commit. Все это может превратиться в более сложный граф, состоящий из нескольких веток, которые сливаются в одну.



Об этом мы поговорим в следующих статьях. Для начала разберем работу с одной веткой.

Установка

Основой интерфейс для работы с Git-ом является консоль/терминал. Это не совсем удобно, тем более для новичков, поэтому предлагаю поставить дополнительную программу с графическим интерфейсом (кнопками, графиками и т.д.). О них я расскажу чуть позже.

Но для начала, все же установим сам Git.

* Windows. Проходим по [этой ссылке](https://git-scm.com/download/win), выбираем под вашу ОС (32 или 64 битную), скачиваем и устанавливаем.
* Для Mac OS. Открываем терминал и пишем:

#Если установлен Homebrew

brew install git

#Если нет, то вводим эту команду.

git --version

#После этого появится окно, где предложит установить Command Line Tools (CLT).

#Соглашаемся и ждем установки. Вместе с CLT установиться и git

* Linux. Открываем терминал и вводим следующую команду.

# Debian или Ubuntu

sudo apt install git

# CentOS

sudo yum install git

Настройка

Вы установили себе Git и можете им пользоваться. Давайте теперь его настроим, чтобы когда вы создавали commit, указывался автор, кто его создал.

Открываем терминал (Linux и MacOS) или консоль (Windows) и вводим следующие команды.

#Установим имя для вашего пользователя

#Вместо <ваше\_имя> можно ввести, например, Grisha\_Popov

#Кавычки оставляем

git config --global user.name "<ваше\_имя>"

#Теперь установим email. Принцип тот же.

git config --global user.email "<адрес\_почты@email.com>"

Создание репозитория

Теперь вы готовы к работе с Git локально на компьютере.

Создадим наш первый репозиторий. Для этого пройдите в папку вашего проекта.

#Для Linux и MacOS путь может выглядеть так /Users/UserName/Desktop/MyProject

#Для Windows например С://MyProject

cd <путь\_к\_вашему\_проекту>

#Инициализация/создание репозитория

git init

Теперь Git отслеживает изменения файлов вашего проекта. Но, так как вы только создали репозиторий в нем нет вашего кода. Для этого необходимо создать commit.

#Добавим все файлы проекта в нам будующий commit

git add .

#Или так

git add --all

#Если хотим добавить конкретный файл то можно так

git add <имя\_файла>

#Теперь создаем commit. Обязательно указываем комментарий.

#И не забываем про кавычки

git commit -m "<комментарий>"

Отлично. Вы создали свой первый репозиторий и заполнили его первым commit.

Процесс работы с Git

Не стоит после каждого изменения файла делать commit. Чаще всего их создают, когда:

* Создан новый функционал
* Добавлен новый блок на верстке
* Исправлены ошибки по коду
* Вы завершили рабочий день и хотите сохранить код

Это поможет держать вашу ветки в чистоте и порядке. Тем самым, вы будете видеть историю изменений по каждому нововведению в вашем проекте, а не по каждому файлу.

Визуальный интерфейс

Как я и говорил ранее, существуют дополнительные программы для облегчения использования Git. Некоторые текстовые редакторы или полноценные среды разработки уже включают в себя вспомогательный интерфейс для работы с ним.

Но существуют и отдельные программы по работе с Git. Могу посоветовать эти:

* [GitHub Desktop](https://desktop.github.com/)
* [Sourcetree](https://www.sourcetreeapp.com/)
* [GitKraken](https://www.gitkraken.com/)

Я не буду рассказывать как они работают. Предлагаю разобраться с этим самостоятельно.

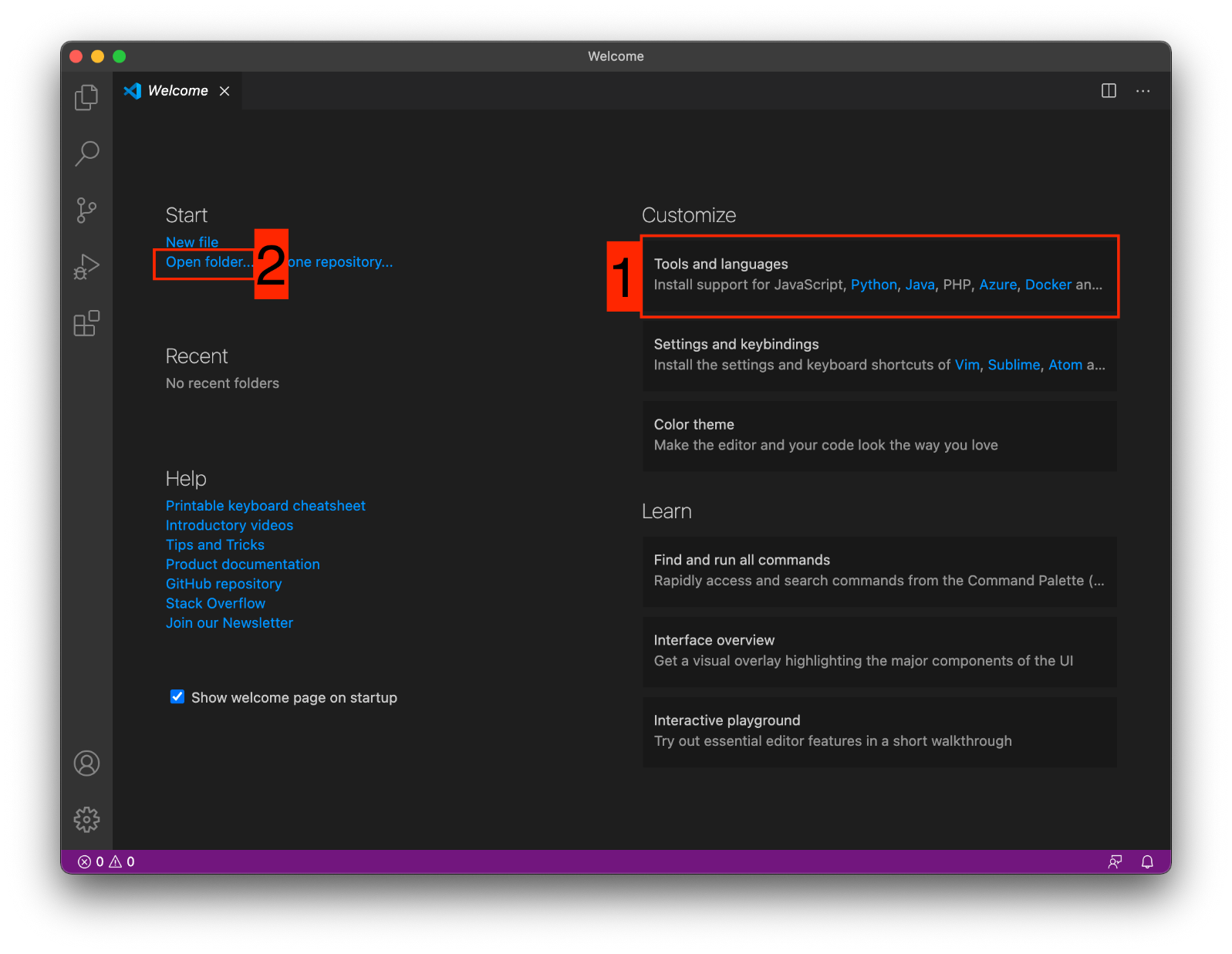
Создаем свой первый проект и выкладываем на GitHub

Давайте разберемся как это сделать, с помощью среды разработки Visual Studio Code (VS Code).

Перед началом предлагаю зарегистрироваться на [GitHub](https://github.com/).

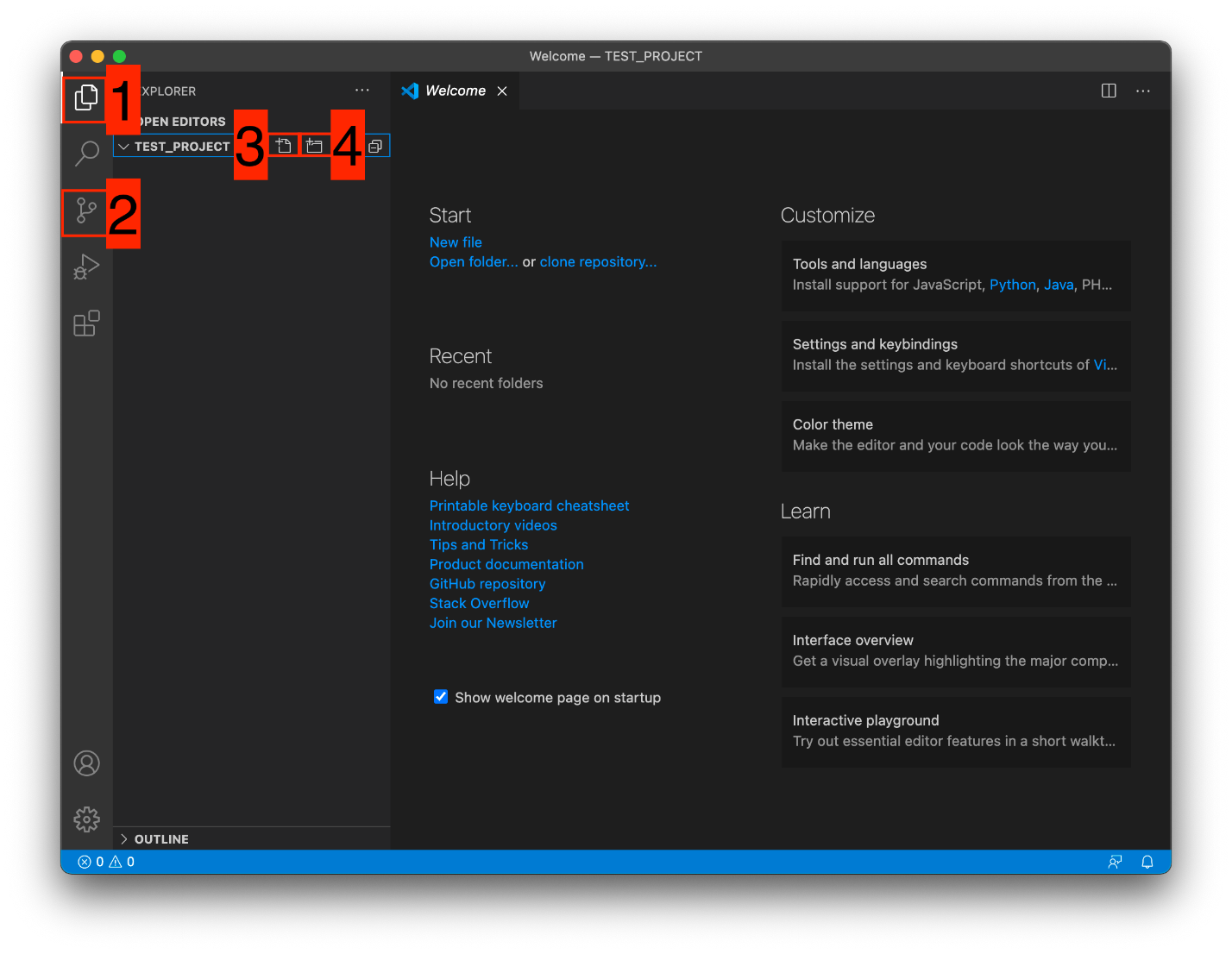
Создайте папку, где будет храниться ваш проект. Если такая папка уже есть, то создавать новую не надо.

После открываем VS Code .



1. Установите себе дополнительно анализаторы кода для JavaScript и PHP
2. Откройте вашу папку, которую создали ранее

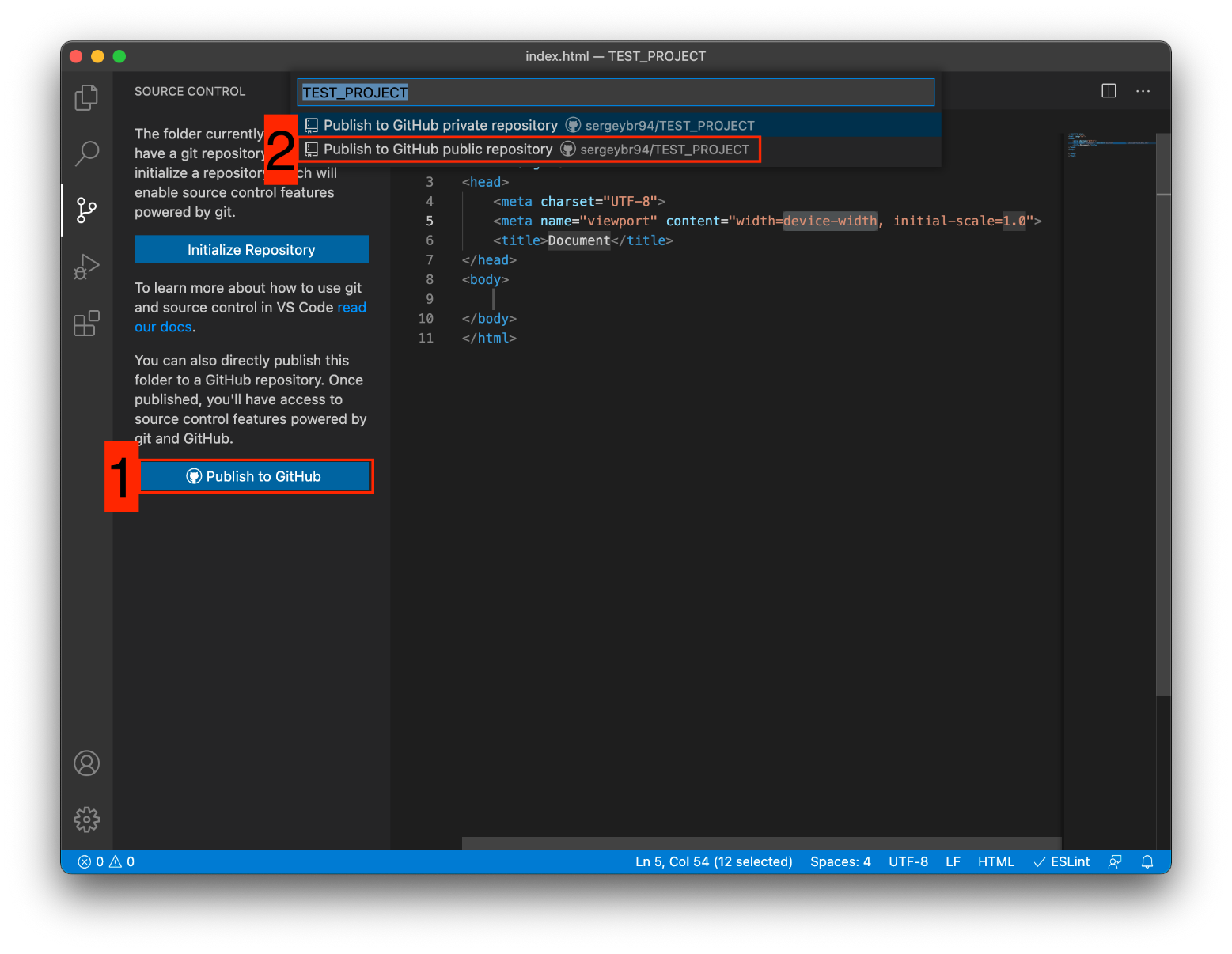
После этого у вас появится вот такой интерфейс



1. Здесь будут располагаться все файлы вашего проекта
2. Здесь можно работать с Git-ом
3. Кнопка для создания нового файла
4. Кнопка для создания новой папки

Если ваш проект пустой, как у меня, то создайте новый файл и назовите его index.html . После этого откроется окно редактирование этого файла. Напишите в нем ! и нажмите кнопку Tab . Автоматически должен сгенерироваться скелет пустой HTML страницы. Не забудьте нажать ctrl+s чтобы файл сохранился.

Давайте теперь перейдем во вкладу для работы с Git-ом.



Откроется вот такое окно:

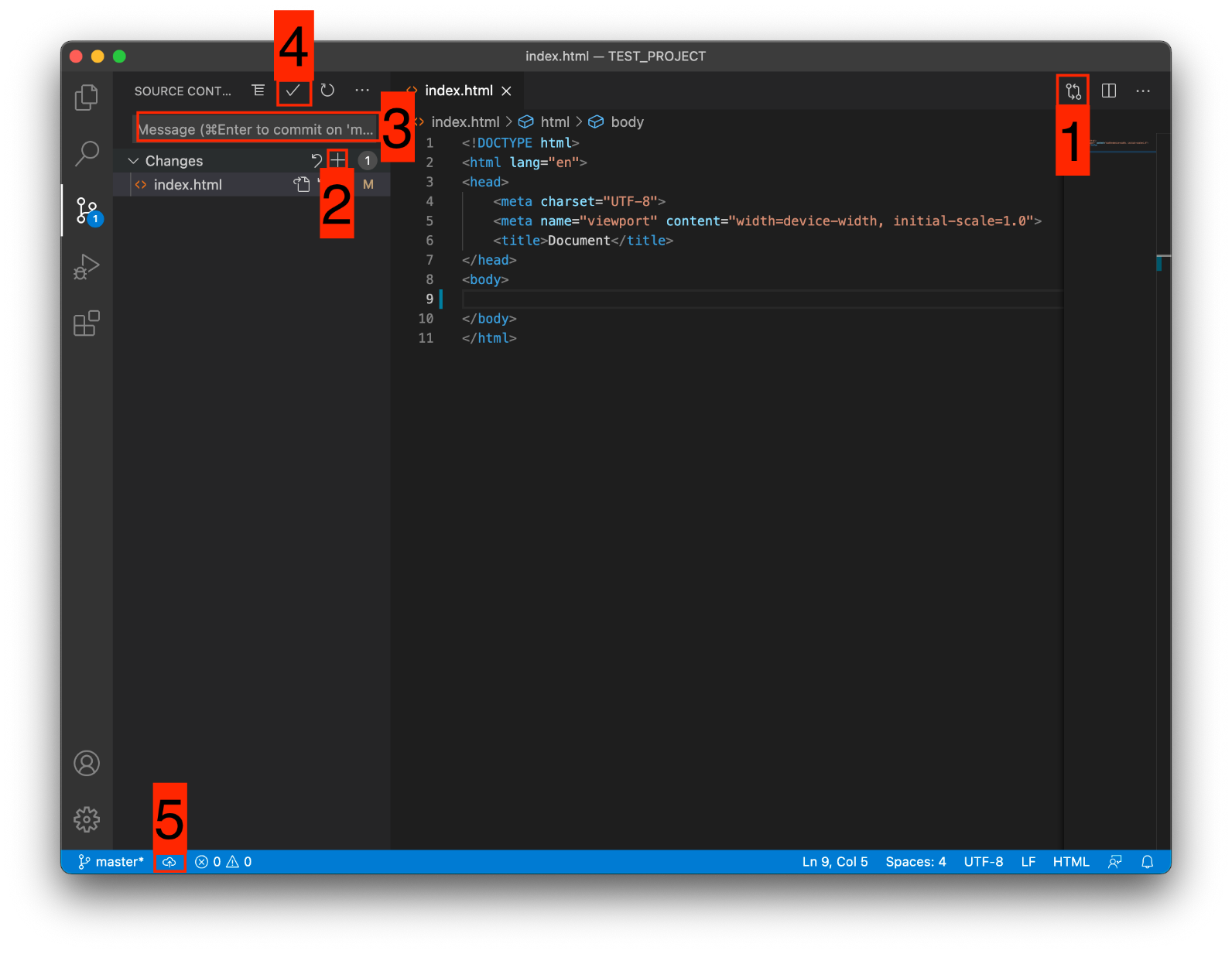
1. Кнопка для публикации нашего проекта на GitHub
2. После нажатия на кнопку 1 , появится всплывающее окно. Нужно выбрать второй вариант или там где присутствует фраза ...public repository

Если вы хотите создать локальный репозиторий и опубликовать код в другой сервис, то необходимо нажать на кнопку Initialize Repository . После этого, вручную выбрать сервис куда публиковать.

После того, как выбрали "Опубликовать на GitHub публичный репозиторий" (пункт 2), программа предложит вам выбрать файлы, которые будут входить в первый commit. Проставляем галочки у всех файлов, если не проставлены и жмем ОК . Вас перекинет на сайт GitHub, где нужно будет подтвердить вход в аккаунт.

Вы создали и опубликовали репозиторий на GitHub.

Теперь сделаем изменения в коде и попробуем их снова опубликовать. Перейдите во вкладку с файлами, отредактируйте какой-нибудь файл, не забудьте нажать crtl+s (Windows) или cmd+s (MacOS), чтобы сохранить файл. Вернитесь обратно во вкладу управления Git.



Если посмотреть на значок вкладки Git, то можно увидеть цифру 1 в синем кружке. Она означает, сколько файлов у нас изменено и незакоммичено. Давайте его закоммитим и опубликуем:

1. Кнопка для просмотра изменений в файле. Необязательно нажимать, указал для справки
2. Добавляем наш файл для будущего commit
3. Пишем комментарий
4. Создаем commit
5. Отправляем наш commit в GitHub

Поздравляю, вы научились создавать commit и отправлять его в GitHub!

Итог

Это первая вводная статья по утилите Git. Здесь мы рассмотрели:

* Как его устанавливать
* Как его настраивать
* Как инициализировать репозиторий и создать commit через консоль
* Как на примере VS Code, опубликовать свой код на GitHub

Забегая вперед, советую вам погуглить, как работают следующие команды:

git help # справка по всем командам

git clone

git status

git branch

git checkout

git merge

git remote

git fetch

git push

git pull

P.S. Для облегчения обучения, оставлю вам ссылку на бесплатный тренажер по Git.

<https://learngitbranching.js.org/>

В телеграмм канале[Step by Step](https://t.me/stepByStep_dev), я публикую еще больше материала и провожу обучающие стримы, для всех желающих.

**Теги:**

* [git](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bgit%5D)
* [github](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bgithub%5D)
* [vscode](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bvscode%5D)
* [новичкам](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5B%D0%BD%D0%BE%D0%B2%D0%B8%D1%87%D0%BA%D0%B0%D0%BC%5D)

**Хабы:**

* [Git](https://habr.com/ru/hub/git/)
* [GitHub](https://habr.com/ru/hub/github/)

# Git для новичков (часть 2)

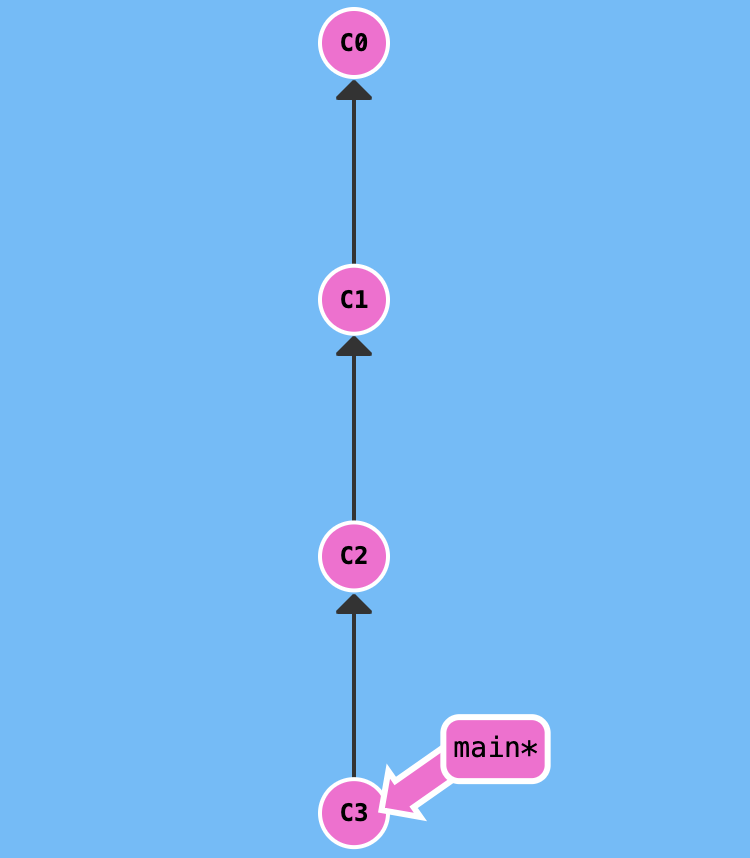
[Git\*](https://habr.com/ru/hub/git/)[GitHub\*](https://habr.com/ru/hub/github/)

В [прошлой статье](https://habr.com/ru/post/541258/), я рассказал, что такое Git, как его установить и выложить свой код на GitHub. Сегодня мы поговорим про работу в команде над одним проектом. И как это устроено в Git.

В данной статье, вся работа с Git будет через командную строку.

### Совместная работа

Представим, что вы с друзьями придумали проект, с "блэкджеком" и ... Вы разделили обязанности. Кто-то будет делать авторизацию и регистрацию, а кто-то функционал вывода новостей. Для этого вам пригодится ветвление.



**Ветка** - это набор commit (кружок), которые идут друг за другом. У ветки есть название, основную ветку чаще всего называют master (на картинках будет называться main) . Если говорить простыми словами, то ветка master - это наш проект.

Другие ветки - это отдельное место для реализации нового функционала или исправление багов (ошибок) нашего проекта. То есть, с отдельной веткой вы делаете что угодно, а затем сливаете эти изменения в основную ветку master.

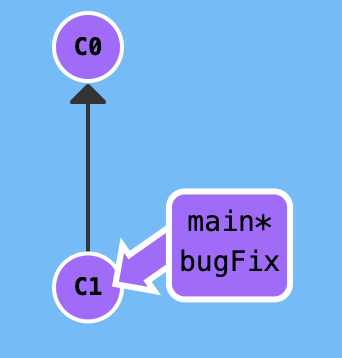
? Не рекомендую создавать commit напрямую в master . Лучше для этого заводить новую ветку и все изменения писать там.

Для того, чтобы создать новую ветку вводим:

git branch <название\_ветки>

#или вот так

git checkout -b <название\_ветки>



Эти команды делают тоже самое, только второй вариант позволяет сразу переключиться в новую ветку. Вносить изменения в новую ветку можно сразу после ее создания.

При создании новой ветки, старайтесь называть ее кратким и ёмким именем. Чтобы сразу было понятно, что именно изменялось по проекту. Если вы используете, какую-нибудь систему для ведения задач, то можете в начале названия ветки указывать ID задачи, чтобы можно было легко найти, на основе какой задачи была создана ветка. Например вот так:

3424\_fix\_catalog\_ajax

В каждом новом commit следует оставлять коммент и в нем описывать суть изменений.

Переключаться между ветками можно такой командой:

git checkout <название\_ветки>

После того, как вы завершили работу над своей задачей, ветку можно слить в master . Для этого нужно переключиться в ветку master и выполнить следующую команду:

# Переключаемся в master

git checkout master

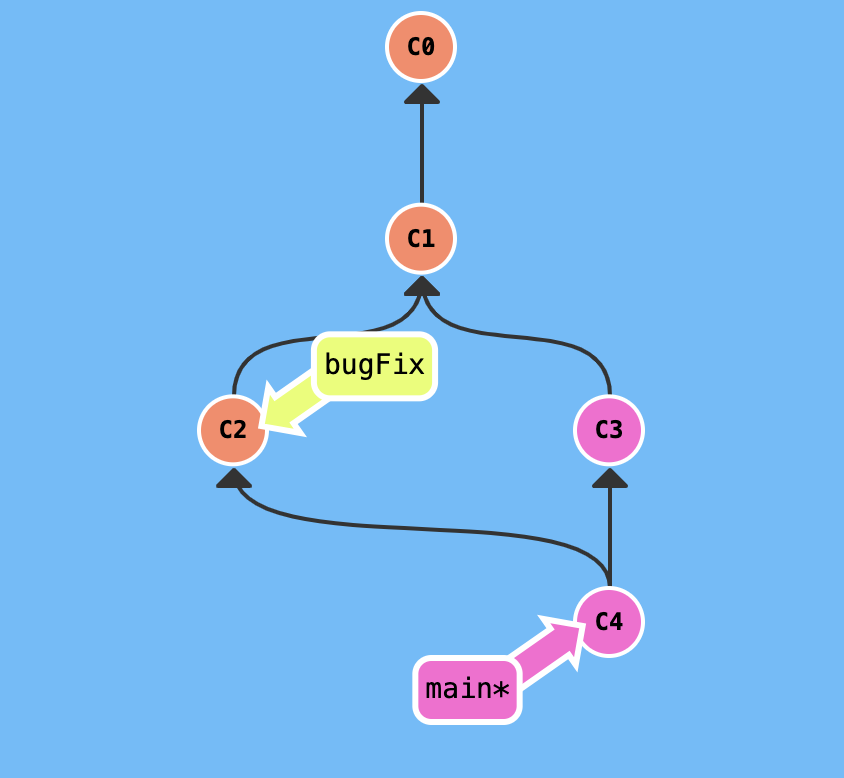
# Обновляем локальную ветку с сервера

git pull origin master

# Делаем merge вашей ветки, в ветку в которой вы находитесь

# В данном примере это master

git merge <название\_ветки>



❗️ Перед тем как сливать новый merge , стоит обновить локальную ветку master , во избежания дальнейших проблем.

Команда merge берет все изменения из ветки (например bugFix) и добавляет их в ветку master.

Для того чтобы посмотреть текущее состояние ветки, например, какие файлы добавлены или не добавлены для создания commit, можно выполнить команду:

git status

Другие пользователи не увидят вашу ветку, пока она не будет отправлена на удаленный репозиторий. Поэтому, после того как вы слили все изменения в master , нужно отправить их в GitHub. Для этого **обязательно** нужно находиться в ветке master :

git checkout master

# Отправляем наши изменения в GitHub

git push origin master

Теперь все ваши изменения, в ветке master улетели в GitHub. Таким же образом, можно отправить любую другую ветку:

git checkout <название\_ветки>

git push origin <название\_ветки>

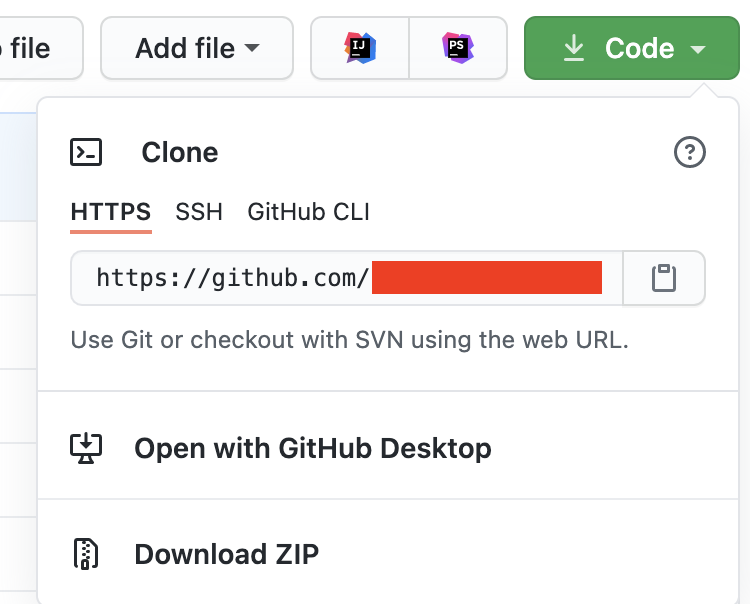
?Совет. Каждый коммит, лучше заливать сразу в удаленный репозиторий. Никто не застрахован, поломки собственного ПК. Поэтому чтобы не потерять все наработки, не забывайте сливать ваши изменения на GitHub.

### Как же теперь другой человек получит все ваши изменения?

Для этого вам понадобиться GitHub или любой другой сервис для хранения кода. В прошлой статье я рассказывал как отправить код в GitHub. Сейчас я покажу, как его скопировать обратно себе на компьютер.

Если у вашего друга раньше не было проекта, то ему придется его "клонировать" себе:

git clone <адрес\_репозитория>



? Адрес репозитория на GitHub можно получить, нажав на зеленую кнопку Code

После выполнения команды, в папке где появиться проект и ваш друг сможет с ним работать. Все ветки и их история также подтянуться.

### Теперь самое главное

Перед тем, как создавать новый функционал и новую ветку, стоит обновить master на вашем устройстве. Для этого нужно находиться в этой ветке и выполнить следующую команду:

# Переключаемся в master

git checkout master

# Подтягиваем изменения из репозитория GitHub

git pull origin master

Таким же образом можно актуализировать любую другую ветку, заменив название ветки master на вашу.

Для обновления всех веток сразу, можно использовать такую, команду, но **не рекомендую**:

git pull

Теперь можно создавать новую ветку и кодить.

### Какие проблемы могут возникнуть?

Git старается автоматически сливать изменения, однако это не всегда возможно. Иногда возникают конфликты. Например, когда в двух ветках были изменения в одной и той же строчке кода. Если такое произошло, то необходимо разрешить конфликт вручную. Для этого откройте файл там, где этого произошло. Например, вы можете увидеть что-то подобное:

# Вы пытались выполнить это

git checkout new\_styles

git merge master

# В конце увидели это

Auto-merging index.html

CONFLICT (content): Merge conflict **in** index.html

Automatic merge failed; fix conflicts and **then** commit the result.

<**html**>

<**head**>

<<<<<<< HEAD

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

=======

<!-- подключение стилей отсутствуют, по какой-то причине -->

>>>>>>> master

</**head**>

<**body**>

<**h1**>Hello,World!</**h1**>

</**body**>

</**html**>

Первый раздел (HEAD) - это то, что находиться в текущей ветке, куда вы пытались слить код. Второй раздел (между ==== и >>>>master) - версия кода в ветке, откуда вы пытались слить код (в данном случае master). Для того, чтобы разрешить конфликт, стоит оставить стили и привести файл в такой вид:

<**html**>

<**head**>

<**link** type="text/css" rel="stylesheet" media="all" href="style.css" />

</**head**>

<**body**>

<**h1**>Hello,World!</**h1**>

</**body**>

</**html**>

После внесения нужных изменений добавьте ваш файл через git add <имя\_файла> как измененный и создайте новый commit:

git add index.html

git commit -m "Merged master fixed conflict."

### Вспомогательные команды

Просмотреть изменения относительно двух веток можно командой:

git diff <исходная\_ветка> <целевая\_ветка>

Удалить ненужную ветку:

git branch -d <название\_ветки>

Просмотр историю ветки:

git log

Подсказки по популярным командам:

# По популярным командам

git help

# Или по конкретной команде

git help <название\_команды>

# Например clone

git help clone

### Практика и вспомогательные инструменты

Для улучшения ваших навыков, в очередной раз оставлю ссылку на [полезный тренажер с заданиями](https://learngitbranching.js.org/).

Так же, для удобства использования в Visual Studio Code, советую поставить [это расширение](https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph), которое визуализирует ваши ветки и commit, и помогает с ними работать.