Начинаем изучение Git’а.

Переходим на сайт <https://git-scm.com/>, где, по сути, мы можем установить Git и почитать что это вообще такое. То есть, на самом деле,

Git – это такая технология, которая служит для контроля версий.

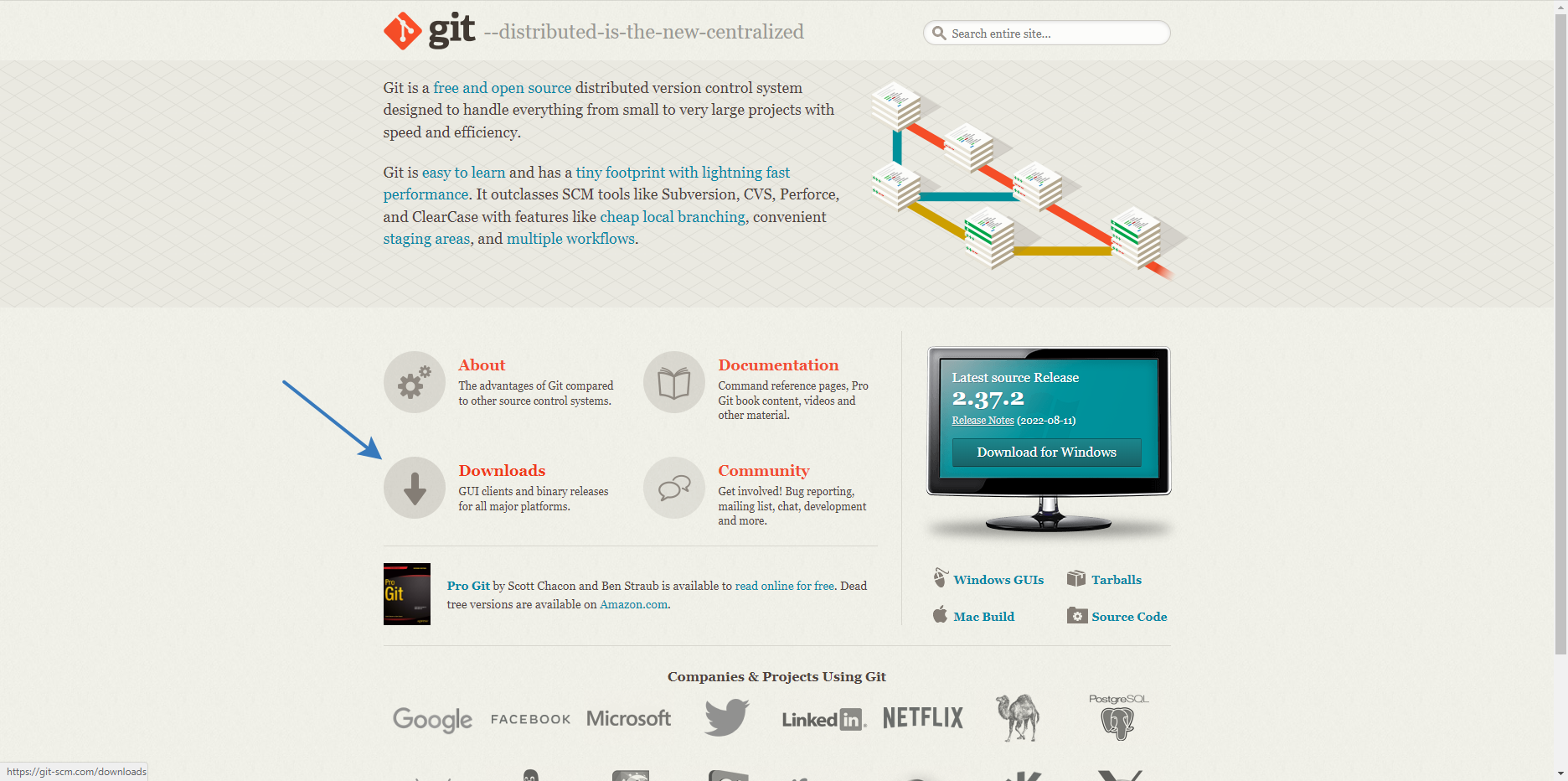
Для чего вообще это нужно? Безусловно, если вы работаете в одиночку над одним проектом, то, вероятнее всего, Git вам будет не так уж сильно и нужен. Но давайте представим какую-нибудь команду, которая работает над одним проектом, когда несколько разработчиков вносят различные изменения в проект, и существует какой-нибудь большой сайт, например VK или YouTube, где понятно, что если разработчик внесет какие-то изменения и вдруг они будут ошибочными, то тогда система рухнет. Нельзя такое давать пользователям, потому что сервис потеряет большое количество денег. Поэтому, как минимум, в больших проектах, да и вообще в любых проектах существует различная версионность, то есть имеется продакшн-приложение, которое видит финальный пользователь; то приложение, которое оттестированно; и то, которое находится в разработке. То приложение, которое находится в разработке, оно, вероятно, с какими-то багами, и множество разработчиков пытаются его исправить.

НО. Как раз таки, система контроля версий Git служит для того, чтобы несколько разработчиков могли работать над одним проектом и не мешать друг другу, то есть система позволяет «разруливать» количество конфликтов, которое возникает при разработке.

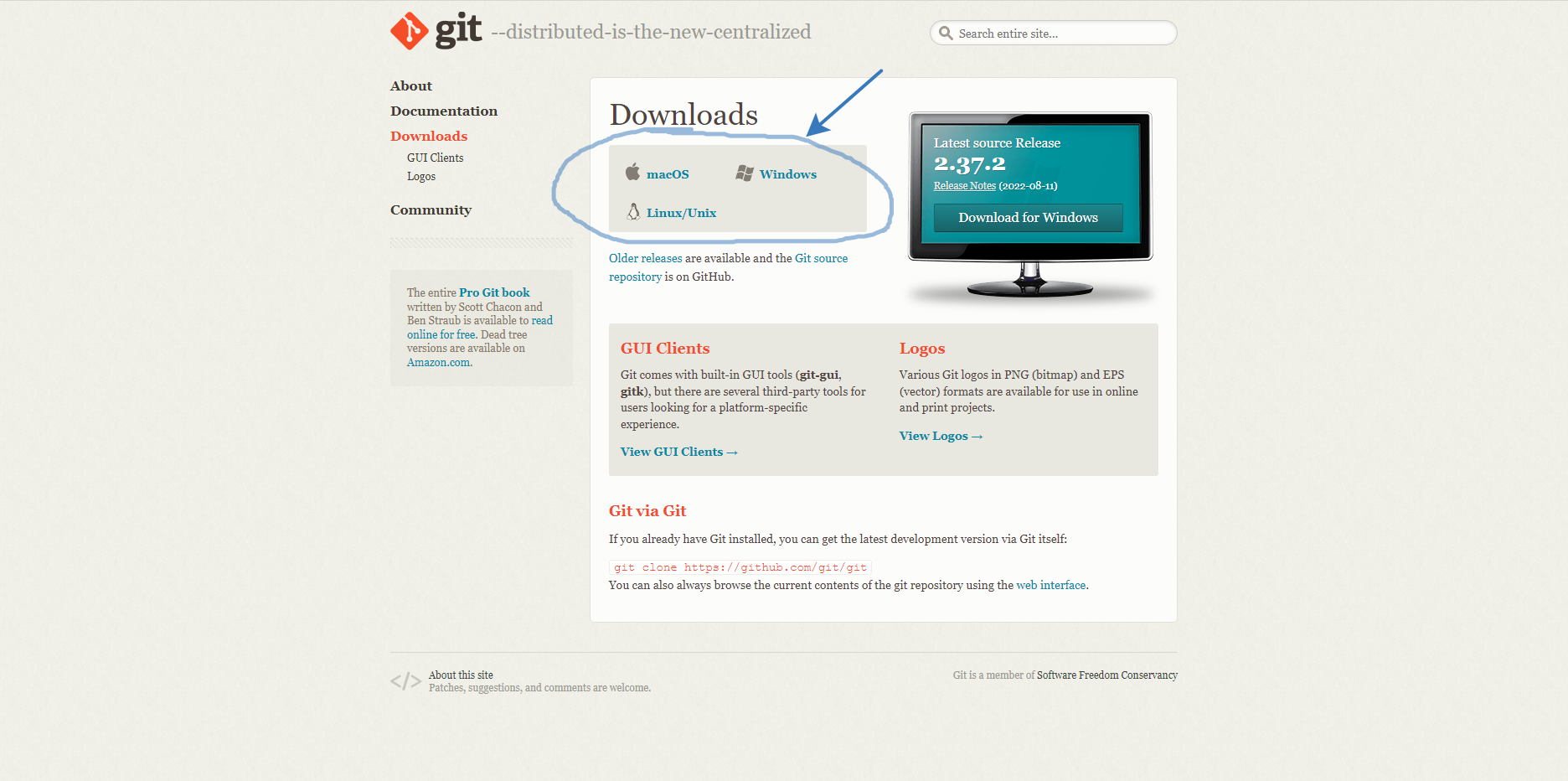
И на этой паре мы как раз посмотрим, как вообще можно этим пользоваться и как это выглядит.

Давайте разберем, как можно установить данную программу.

1. Переходим на сайт <https://git-scm.com/>.
2. Нажимаем на кнопку Downloads.



1. Здесь вы можете выбрать тот архив, который подходит под вашу операционную систему.



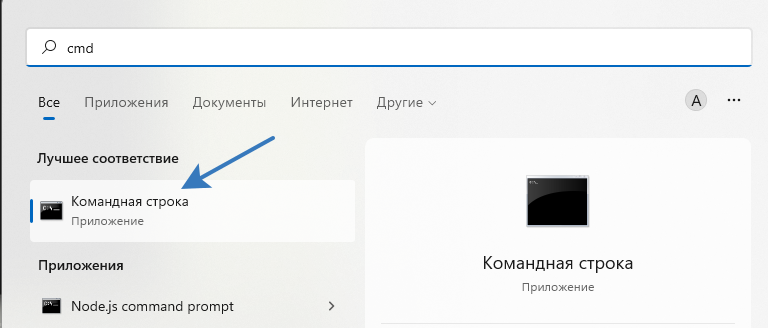
1. Далее установите программу на ваш компьютер. Все пункты по умолчанию.

Поздравляю! Вы установили систему контроля версий Git на свой компьютер.

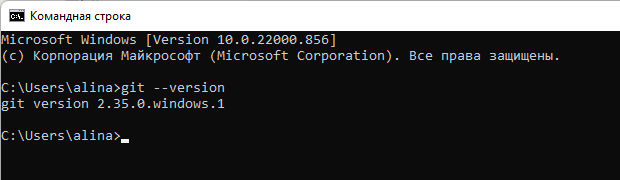
Приступаем к достоверности скачанной программы.

Для этого нам понадобится любой терминал. В моем случае, это Командная строка Windows.

1. Открываем терминал, введя в поисковую строку cmd или Командная строка.



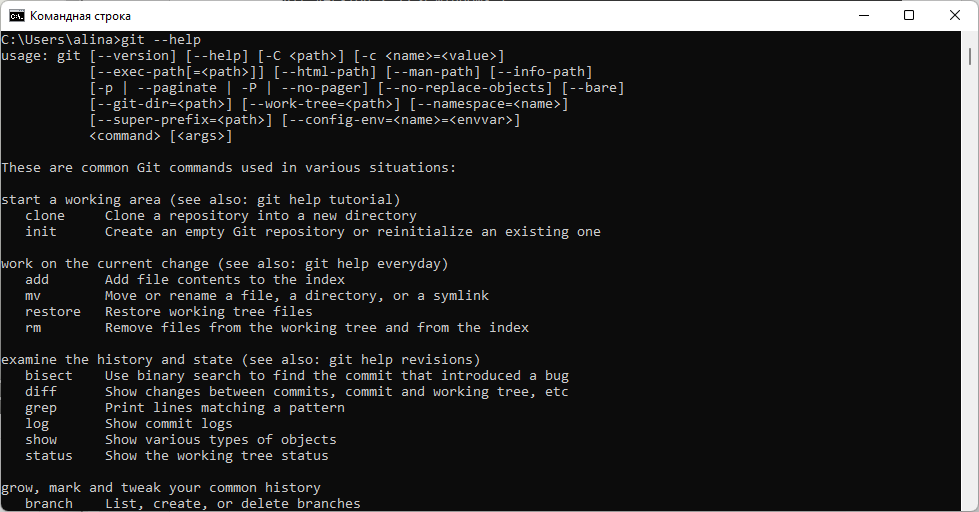
1. Далее мы можем проверить версию Git’а. Для этого необходимо ввести команду git --version и нажать на Enter.



И вот мы получаем текущую версию Git. То есть, если выдается какая-то информация о версии Git, это значит, что система контроля версий установлена и мы можем начинать работу😊

Для начала разберем базовые команды.

1. Команда git --help. Мы обращаемся к Git за предоставлением информации о базовых возможностях того, что есть в этой технологии.



От обилия команд можете пока не пугаться, потому что часть из них, которая используется в больших проектах, мы разобрали на предыдущей лекции.

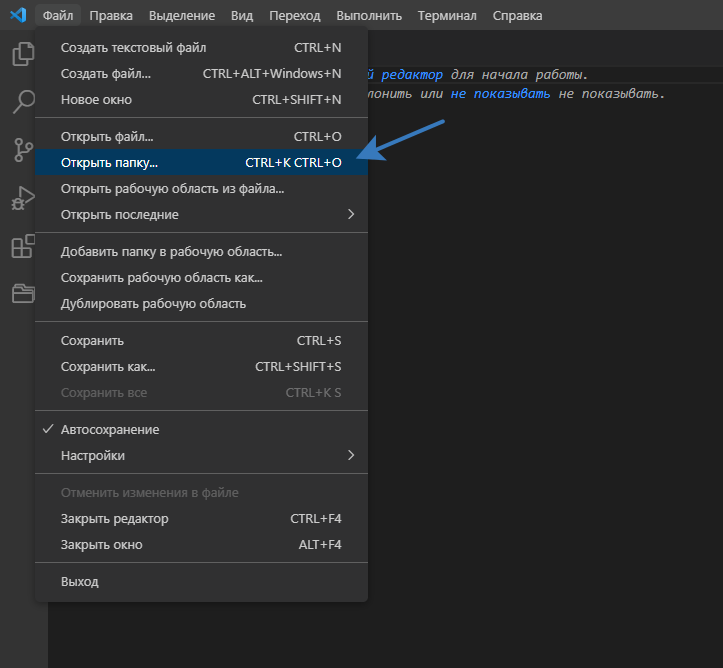
Но сегодня мы разберем, как именно использовать те команды.

Как начать пользоваться данной технологией?

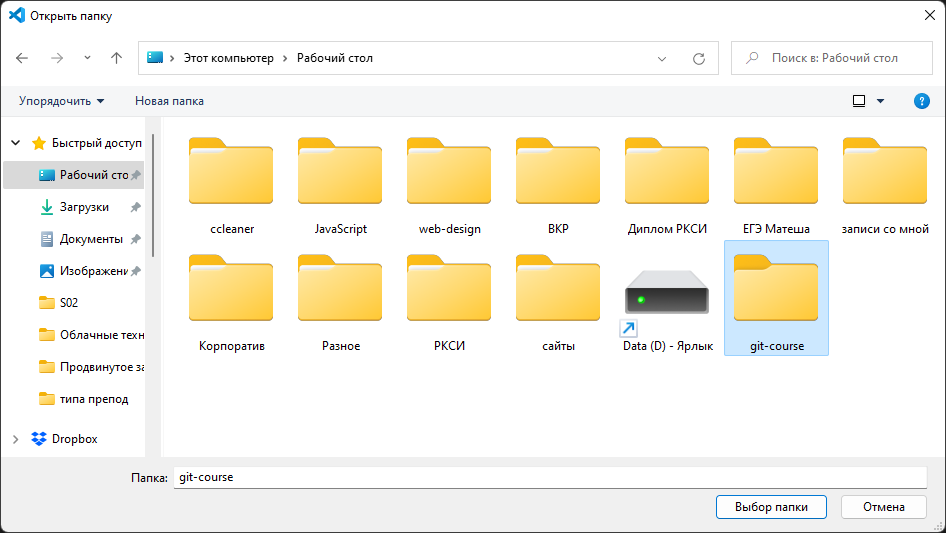
Для начала, нам пригодится какой-то проект. На протяжении данной практической работы мы будем создавать какой-нибудь простой проект, чтобы именно протестировать, как вообще Git работает и что он нам может предложить.

Мы будем пользоваться редактором кода Visual Studio Code (VS Code).

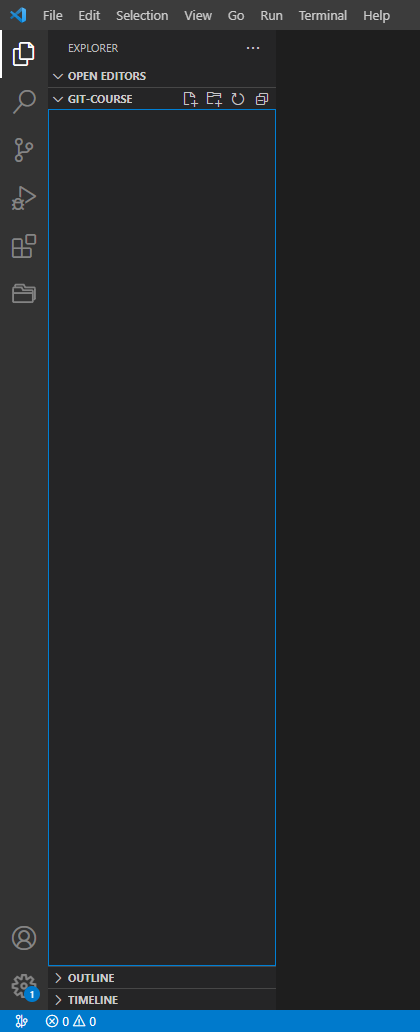
1. Открываем VS Code.
2. Нажимаем Файл -> Открыть папку



1. На рабочем столе создаем папку git-course. После двойным щелчком по папке открываем ее.



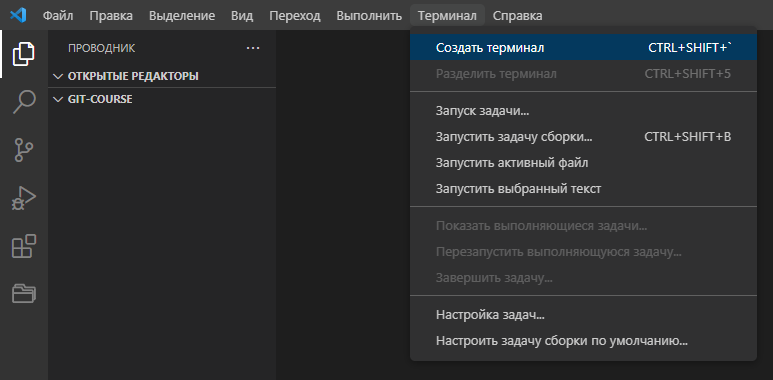
Сейчас у нас проект абсолютно пустой. В нем ничего нет.



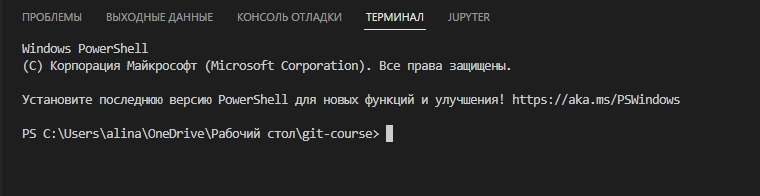
И для того, чтобы связать данный проект с системой контроля версий Git, нам необходимо в консоли перейти в данную папку. Сделать это можно двумя способами.

*1 способ.*

1. Открыть встроенный терминал в VS Code.



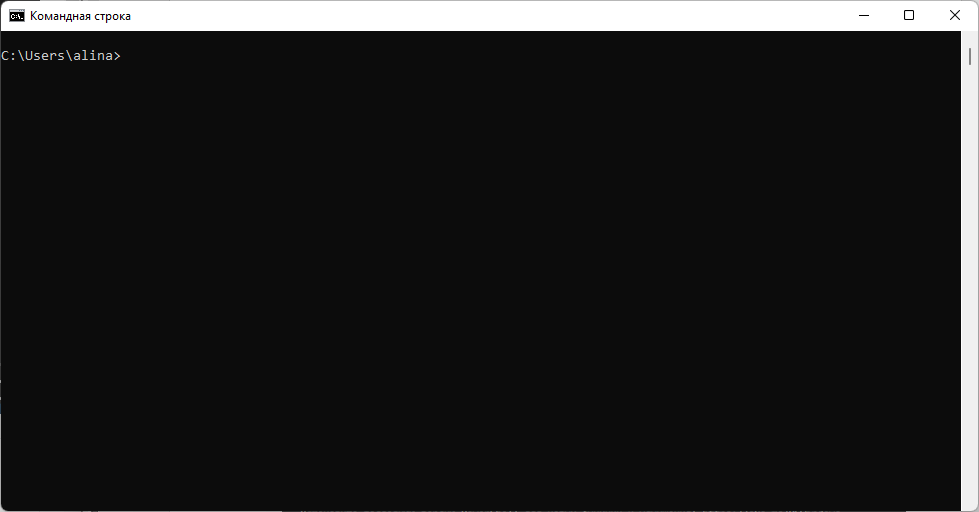
Мы видим, что находимся в папке, которую создали.



Но будем пользоваться 2 способом. С помощью обычного терминала.

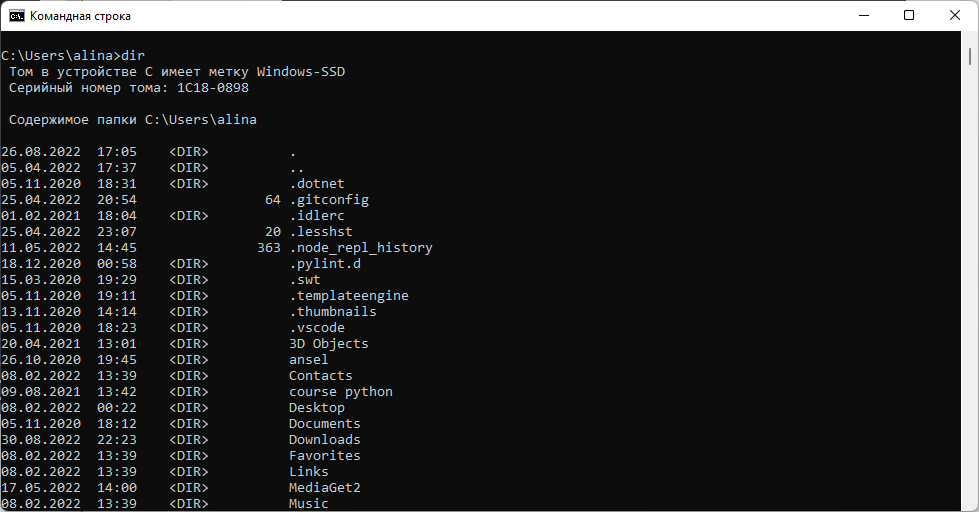
*2 способ.*

1. Очищаем консоль с помощью команды cls.



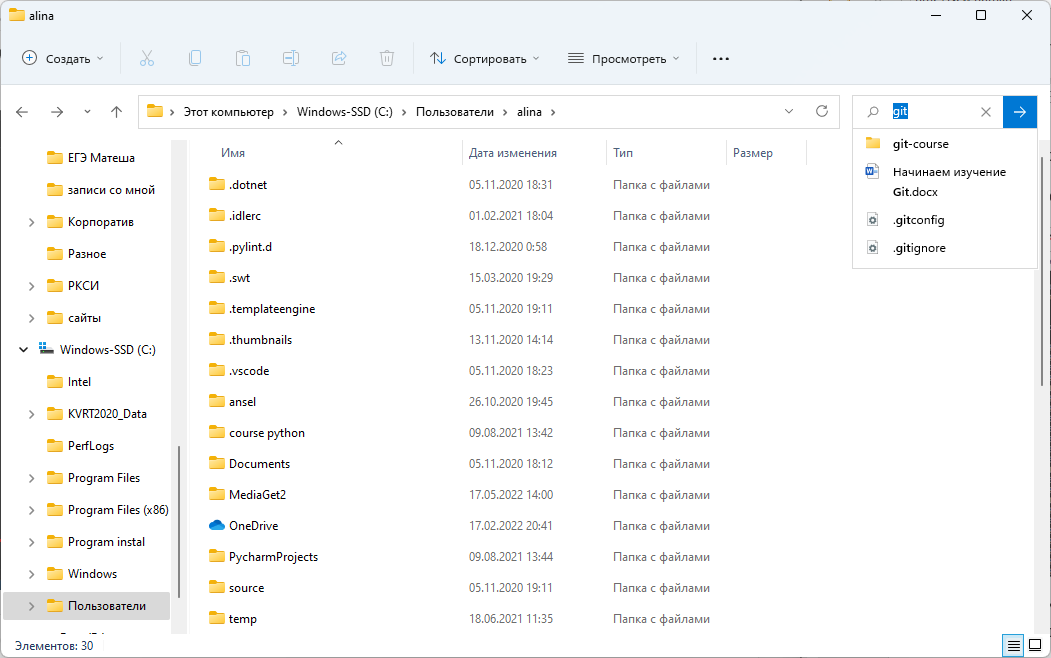
По итогу, у вас должна оказаться такая пустая консоль.

1. С помощью команды dir смотрим все папки, находящиеся в текущей дериктории.

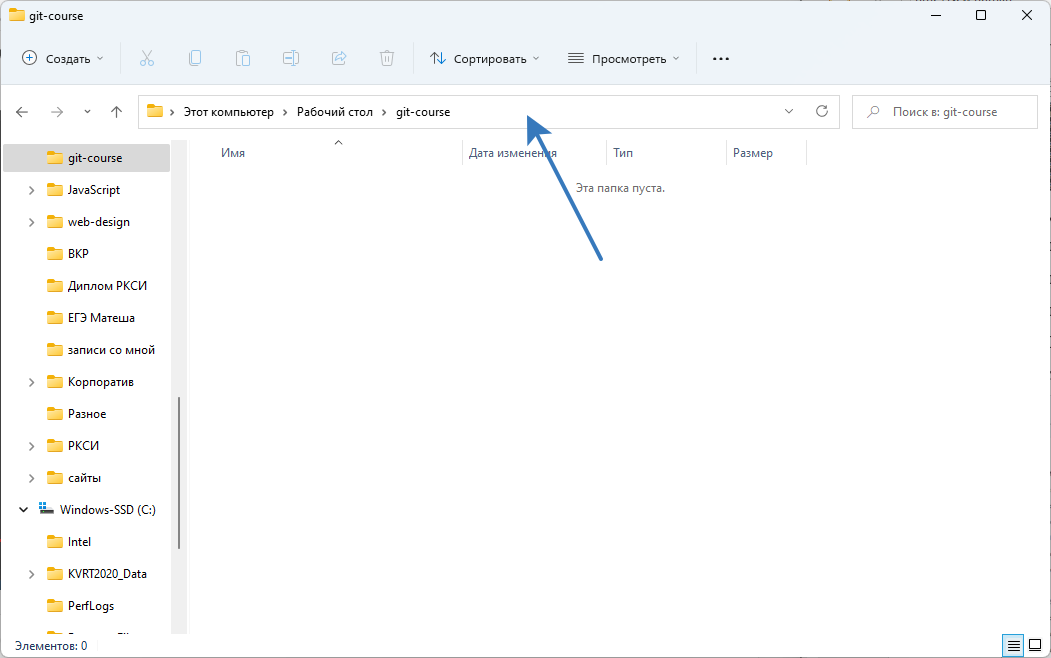


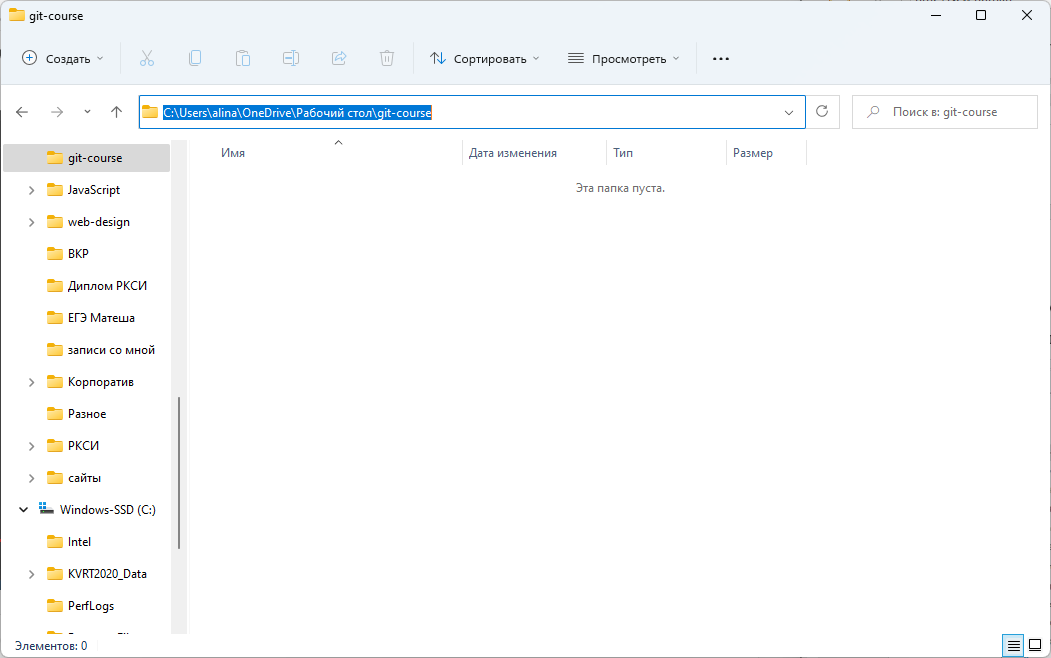
1. Далее нам необходимо перейти в папку git-course. По умолчанию, я нахожусь в системной папке со своим именем, которая в свою очередь находится на диске C. Для того, чтобы найти эту папку, надо открыть проводник -> зайти в диск C -> Пользователи -> alina.

Пишем в поисковой строке git-course. Если появился значок папки с этим же названием, нажимаем на него.



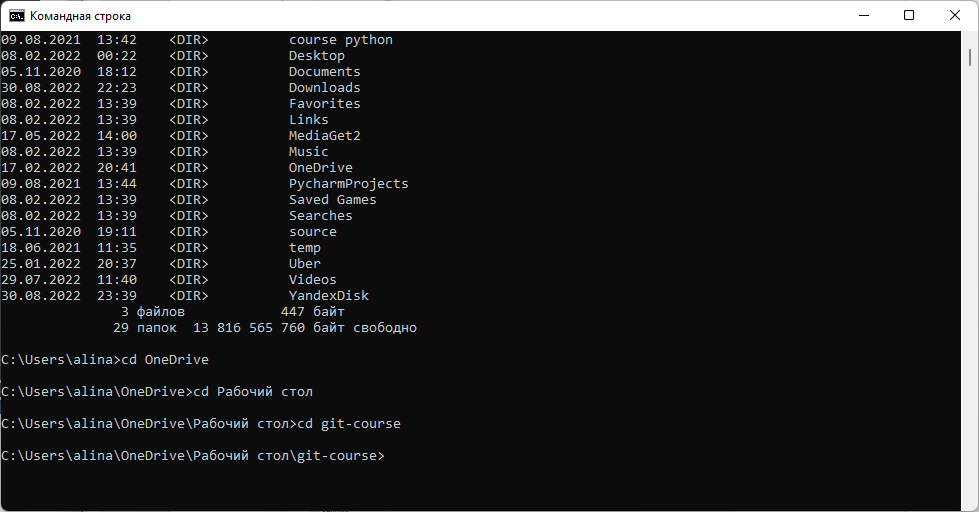
Нажимаем один раз в данное поле. И мы видим, что написан точный путь к нашей папке.



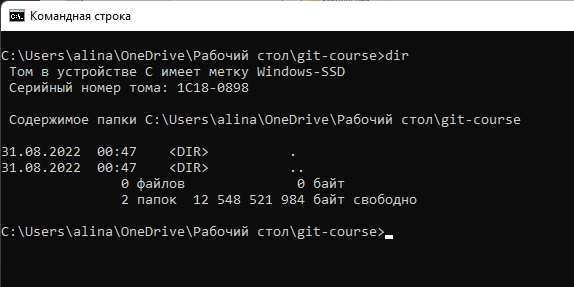


Отлично. Теперь нам надо обратно зайти в консоль.

1. Чтобы перейти в папку git-course, пропишите следующий цикл команд в терминале: cd OneDrive -> cd Рабочий стол -> cd git-course

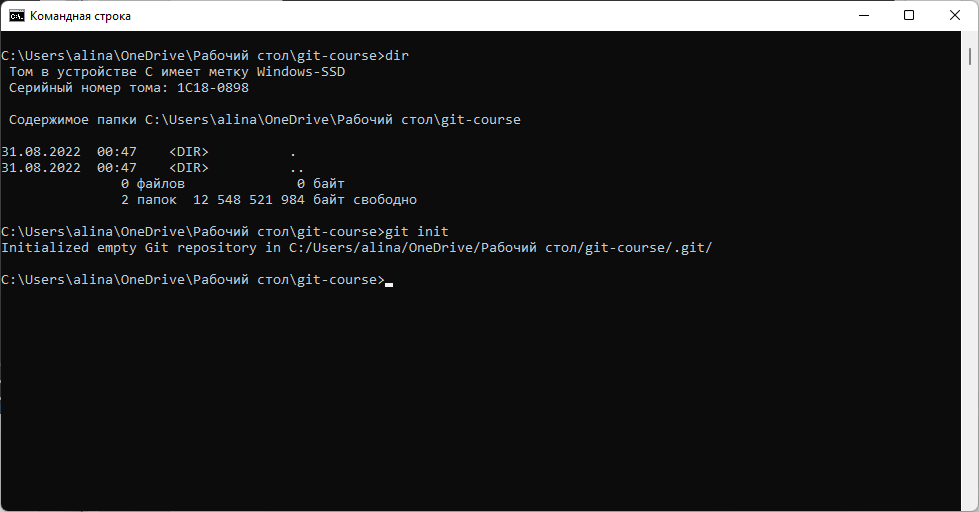


1. Очищаем консоль командой cls.
2. Проверяем папку на наличие каких-либо других папок и файлов.



Как мы видим, папка пуста.

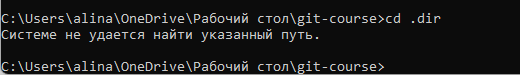
1. Теперь мы можем продолжить работу с Git. И первое, что нам необходимо сделать: проинициализировать (активировать, подготовить к работе) репозиторий (папку). Для этого мы обращаемся к Git командой git init.



После чего мы получаем сообщение о том, что был проинициализирован пустой репозиторий и показан путь до папки с нашим проектом.

По факту, в нашем проекте ничего не изменилось. Как ничего не было, так ничего и нет.

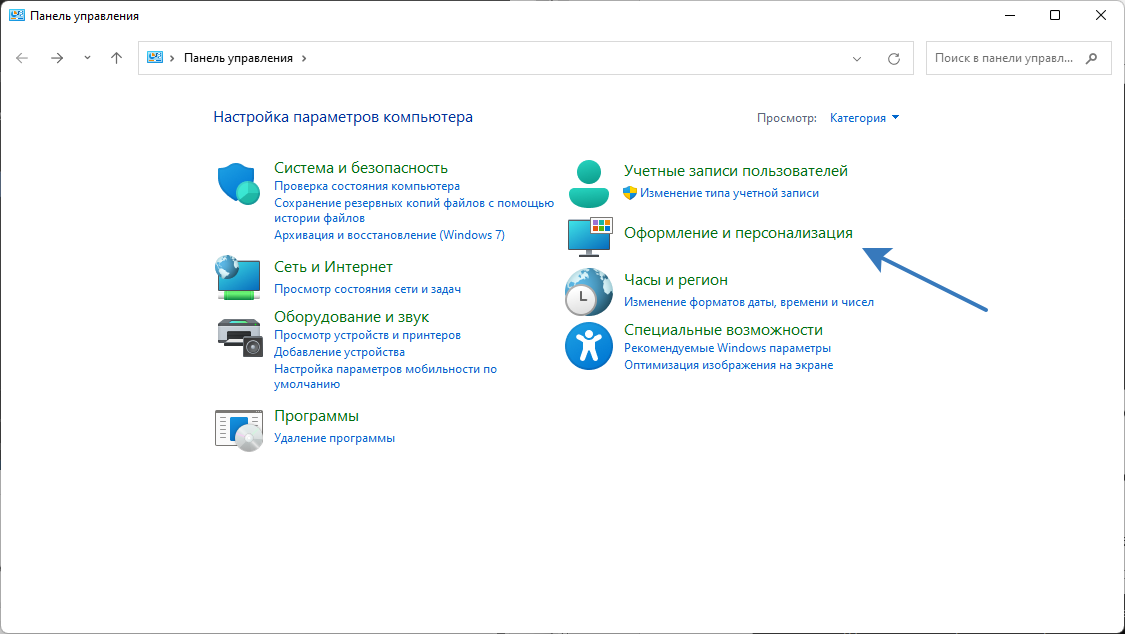
Что же произошло на самом деле? На самом деле, создалась системная папка, которая называется .git. На Винде она может быть скрыта.

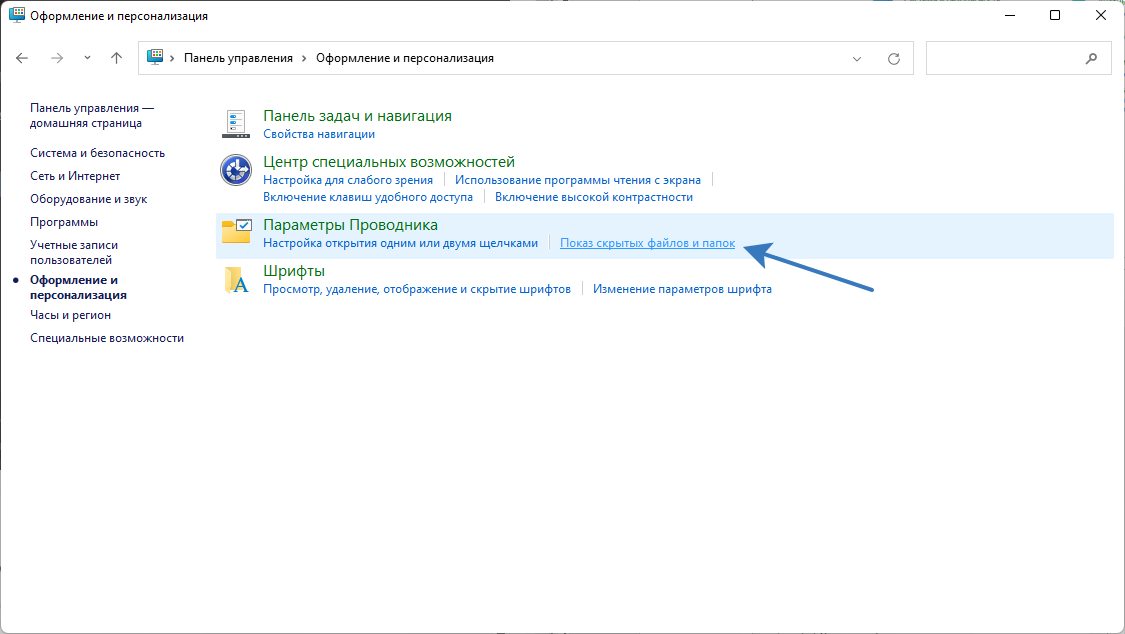


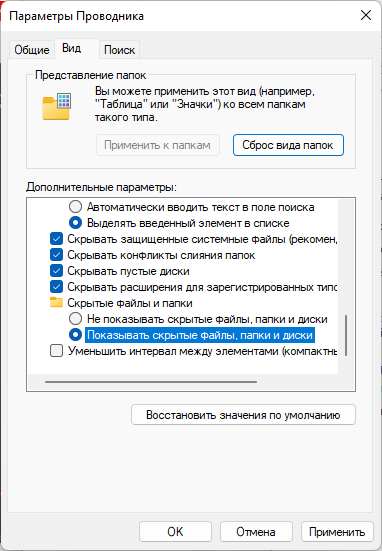
Чтобы папку увидеть, надо настроить видимость папок таким образом, чтобы были показаны и скрытые папки.

Для этого перейдем в Панель управления.

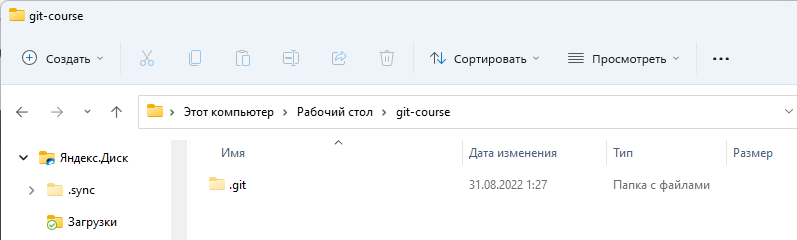
Выберем пункт «Оформление и персонализация». В нем есть «Параметры Проводника» и пункт «Показ скрытых файлов и папок». Нажимаем на него. Находим радиокнопку «Показывать скрытые файлы, папки и диски». Ставим галочку. Нажимаем ОК.



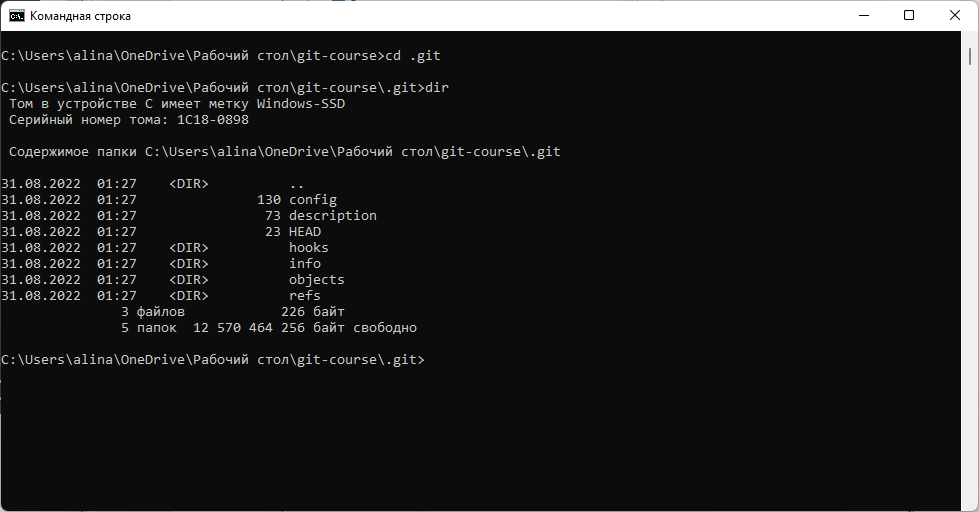




Теперь проверим, действительно ли папка видна.

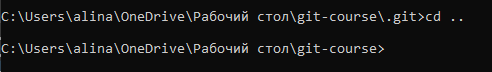


Папка видна. Супер. Теперь попробуем войти в нее через консоль и посмотреть, что же она в себе хранит. Cd .git -> dir.

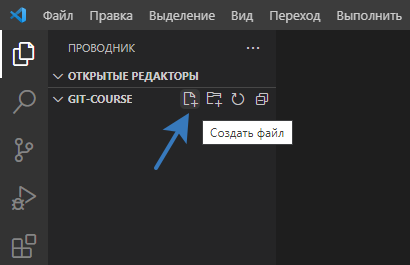


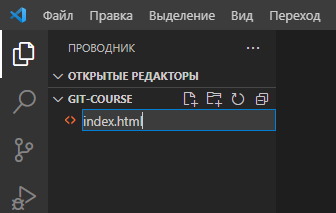
Внутри лежат другие папки, которые относятся как раз таки к Git. В дальнейшем в нем будет храниться различная информация. Нам необязательно знать, что находится внутри. Важно понимать, что такая папка присутствует и что в ней записывается техническая информация, нужная для данной системы контроля версий.

1. Пока выйдем на уровень проекта. Для этого нам надо прописать команду cd ..

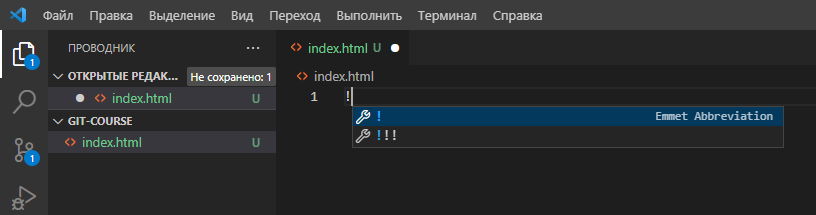


1. Очистим консоль.
2. Теперь давайте создадим какой-нибудь первый файл, который будет служить нам в качестве нашего проекта. Для этого необходимо нажать кнопку «Создать файл». И создадим базовый файл index.html.

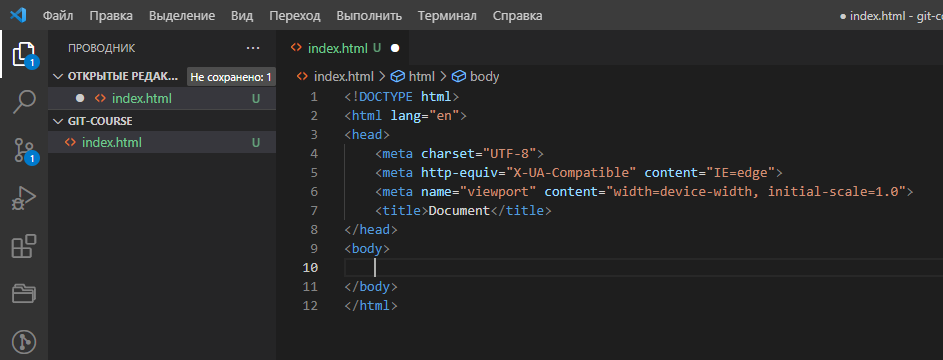




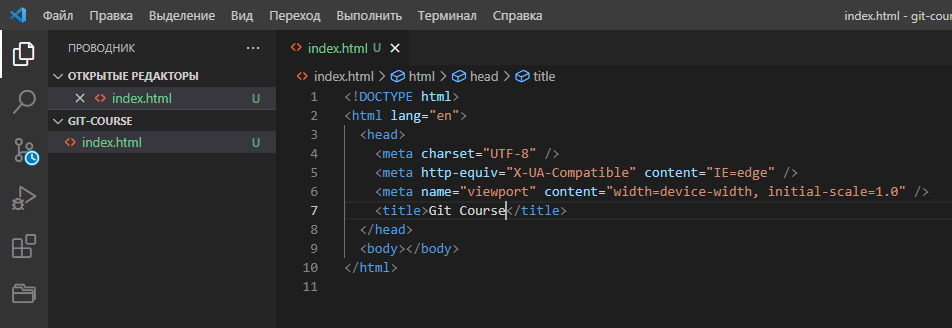
1. Далее с помощью комбинации !+Tab создадим документ.



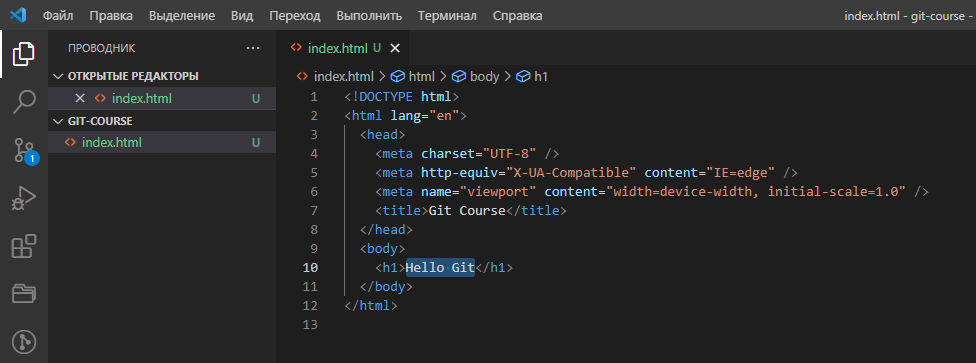
Должно в итоге получится следующее:



1. Напишем базовую разметку в title. Например, Git Course.

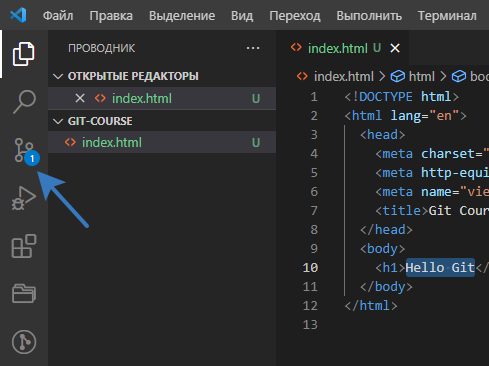


1. Зададим в теге body заголовок h1. Например, Hello Git.



Контент файла в данном случае нам вообще не принципиален, потому что мы будем работать с системой контроля версий, а не создавать сайт.

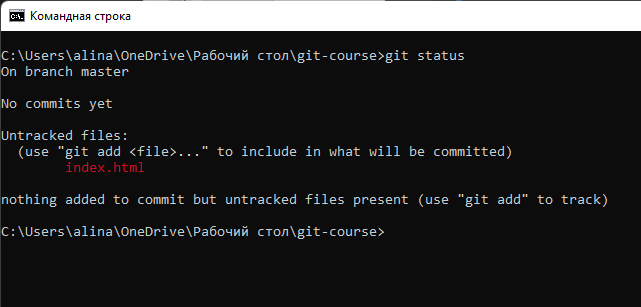
1. Вы можете обратить внимание, что слева в панели загорелась синенькая иконочка. Это как раз таки вещь, служащая для отслеживания системы контроля версий. Но в VS Code мы с этим работать не будем. Всё только через консоль)))))



Итак, что же мы сейчас сделали? Мы создали первый файл. И нам нужно сообщить Git о том, что мы вообще создали такой файл и его нужно зафиксировать. Потому что Git и построен на таких определенных моментах фиксации, т.е. когда мы фиксируем проект на определенной стадии и дальше уже с ним взаимодействуем.

Как нам вообще понять, что происходит сейчас с Git?

1. Мы можем обратиться к Git. Базовой командой, которая постоянно используется для выяснения того, что сейчас вообще происходит, является git status. То есть мы сейчас выясняем текущий статус.



Здесь есть интересная информация. Мы находимся в master ветке.

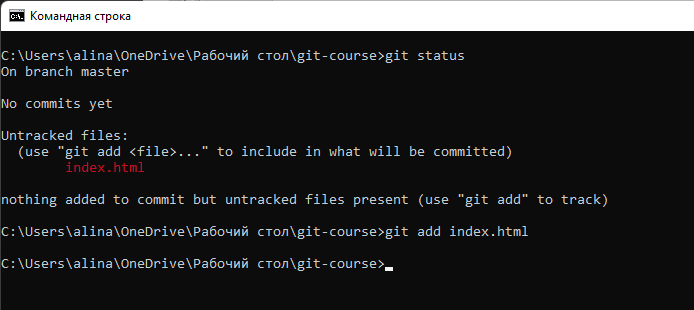
Что же такое ветка? Git имеет древовидную структуру. Ветка представляет из себя версию вашего проекта. И понятно, что, если версий может быть много, соответственно, и веток может быть много. А также эти ветки можно совмещать друг с другом, совмещать разные версии проекта и получать какой-то итог.

Следующая информация о том, что у нас нет никаких коммитов. Мы не зафиксировали никаких версий, никаких изменений. И про это Git нам сообщает.

Еще есть информация о файле, за которым Git не следит (index.html). И в скобочках как раз прописано сообщение, что, написав команду git add <file>…, в нашем случае это index.html, Git может начать следить за ним.

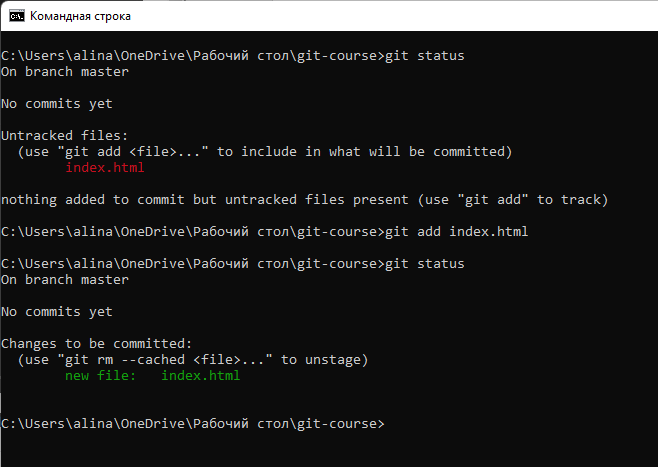
Что же изменится, если мы напишем данную команду?

1. Напишем в терминале команду git add index.html.



Мы ее прописали, но сообщений никаких нет.

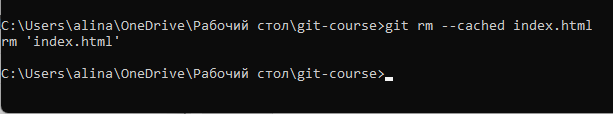
1. Вернемся обратно к git status.



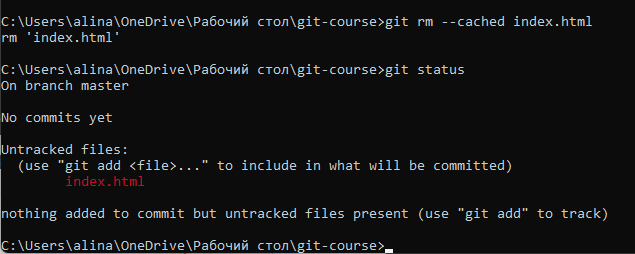
Видим базовую информацию. И новую тоже. Git нам пишет о том, что теперь у нас есть изменения, за которыми Git следит, и мы добавили новый файл. Изначально репозиторий был пустой, потому что мы создали пустой проект. А сейчас появился первый файл index.html.

Здесь также написано, что мы можем перестать следить за файлом при помощи команды git rm --cached <file>… Давайте немного поиграемся с ней.

1. Напишем команду git rm –cached index.html.

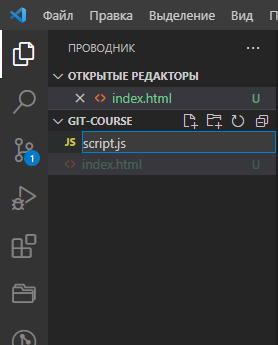


1. Проверим статус проекта.

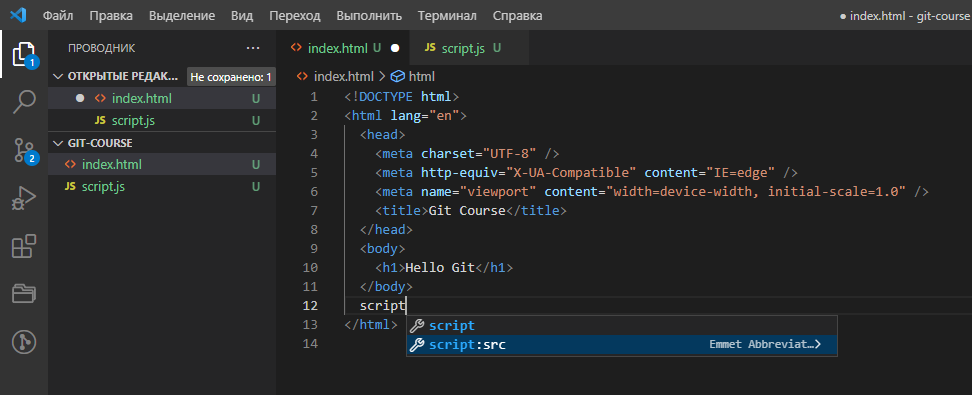


Как мы видим, всё вернулось в первоначальное положение. Надо бы его добавить обратно.

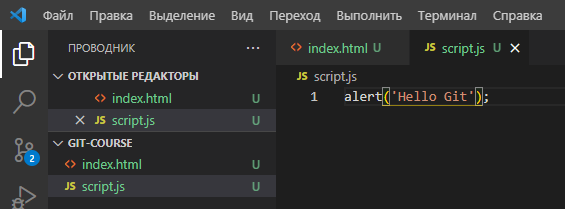
1. Но перед этим мы создадим еще один файл. Например, script.js.



1. Подключим его к index.html. Для этого выбираем script:src -> Enter. Пишем в кавычках имя файла, который надо подключить. В нашем случае, это script.js.

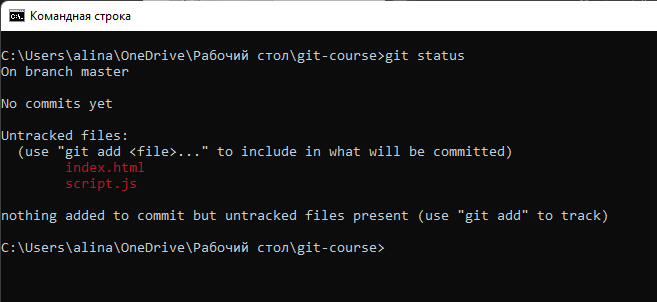


1. Давайте напишем в файле script.js следующее: alert(‘Hello Git’);.



Вот мы и добавили еще один файл. Посмотрим его git status.

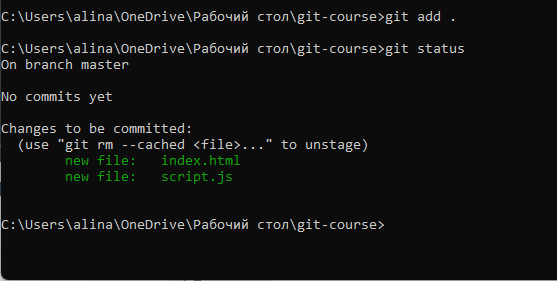
1. В терминале пишем git status.



Как мы видим, у нас есть теперь два файла в нашем проекте, за которыми Git не следит.

Мы можем добавлять их по отдельности каждый. Но мы пойдем хитрым путем.

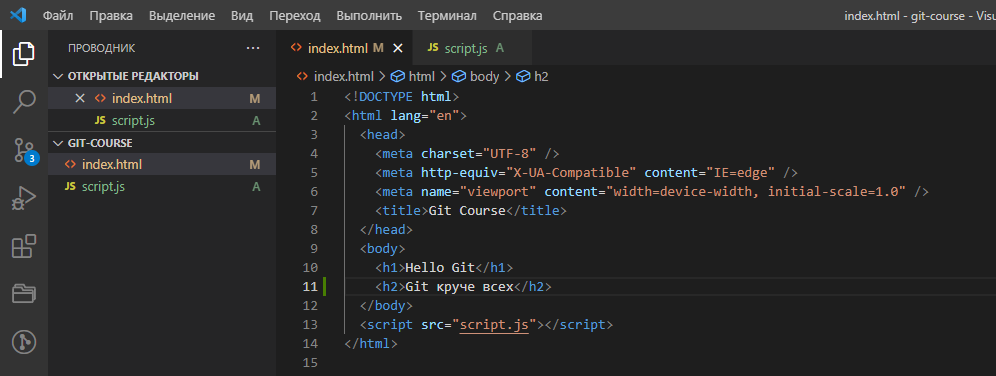
1. Пишем команду git add . -> git status.

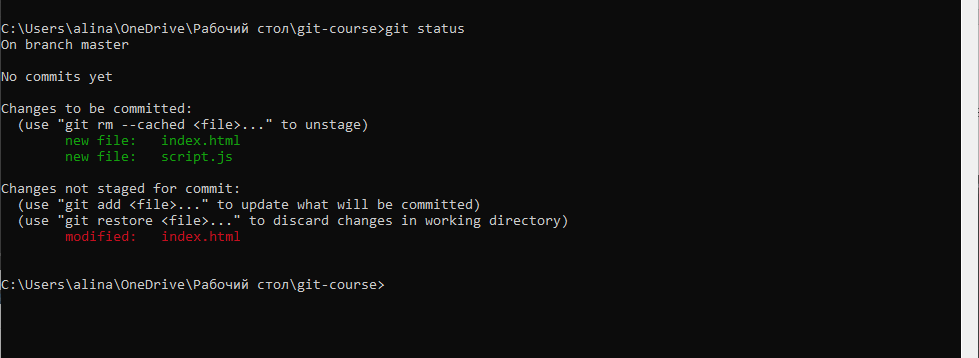


Добавилось сразу два файла. Замечательно.

Теперь важный момент. Мы добавили два файла в Git. И в друг прилетает задача, что нужно что-то изменить в файле index.html. Например, добавить заголовок 2 уровня с текстом «Git круче всех». Внесем данные изменения и посмотрим, как отреагирует на это Git.

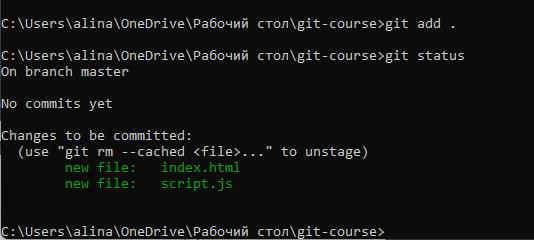
1. Напишем следующее: <h2>Git круче всех</h2>. Далее в терминале напишем команду git status.





Здесь видно, что мы добавили новые файлы. Однако, были внесены определенные изменения в index.html несмотря на то, что мы его добавили. Git нам сообщает, что мы должны снова добавить index.html, перед тем как зафиксировать.

1. Напишем git add . и проверим статус.

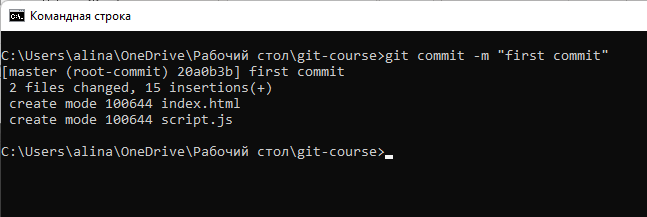


**Запомните!!!!! Если мы вносим какие-то изменения, то перед тем как продолжать работу с Git, нам необходимо всё зафиксировать.**

Следующий этап: Коммит.

Коммит (commit) – это такой фикс нашей версии, для того чтобы с этим проектом можно было работать.

Выглядит это следующим образом: пишем git commit -m “first commit” (в кавычках написано название коммита. Главное, чтобы оно было «говорящее».)



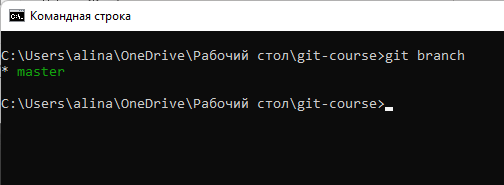
Мы видим, что по умолчанию находимся в ветке master. Также создано 2 файла – index.html и script.js, изменены 2 файла, добавлено 15 вставок.

Еще мы наблюдаем с вами хэш  - уникальный код этого коммита, то есть в дальнейшем к этим коммитам можно будет возвращаться.

**ВАЖНО!!!!! Чем больше мы коммитим, тем больше у нас есть шансов вернуться на определенный этап проектирования!!!!**

Теперь поговорим про ветки.

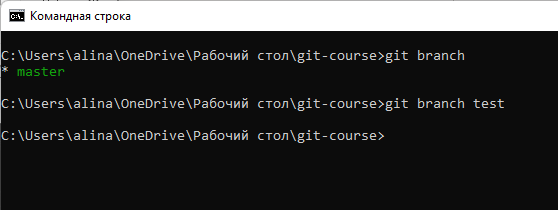
1. В данный момент мы находимся на основной ветке нашего репозитория – master. Чтобы убедиться в этом напишем в консоли следующую команду: git branch



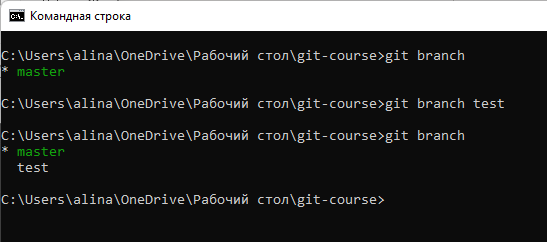
Смоделируем следующую задачу: предположим, есть другой разработчик, которому поручили добавить файл описания для данного проекта. Другой разработчик, в идеале, должен создать другую ветку, чтобы не мешать нашему процессу разработки.

Сейчас мы разберем, как это сделать.

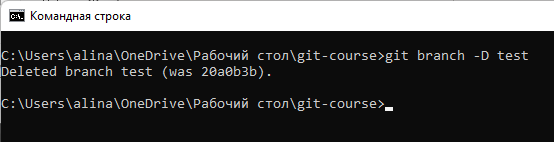
1. Для того, чтобы создать новую ветку, мы обращаемся к команде git, далее к его команде branch, дальше говорим, какую ветку мы хотим создать.



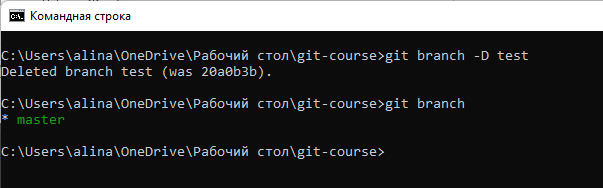
Мы не видим сейчас никаких логов в консоли. Но если мы снова обратимся к команде git branch, то мы увидим список всех веток в репозитории.



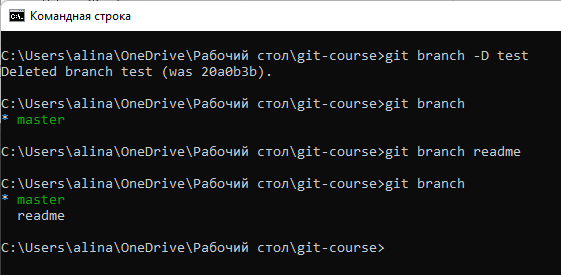
1. Также мы можем удалить ветку test с помощью команды git branch -D test



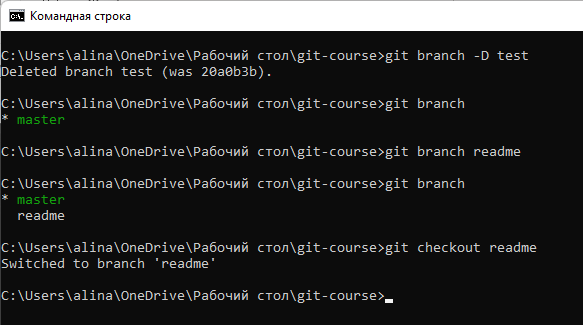
Ветка удалена. Можно убедиться в том, что ее действительно больше нет, с помощью команды git branch



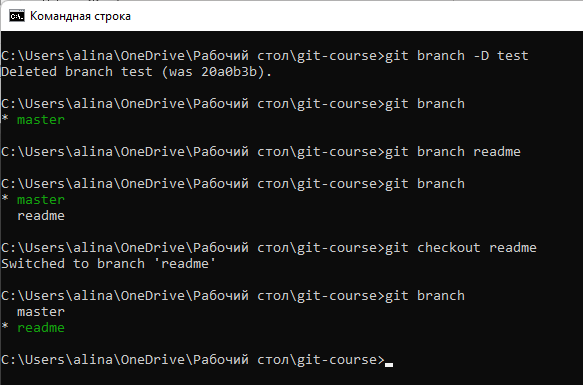
1. Давайте, всё-таки, создадим другую ветку readme.



Заметьте, несмотря на то что мы создали другую ветку, мы всё еще находимся в master. Чтобы переключиться на другую ветку, нам необходимо вызвать команду git checkout readme

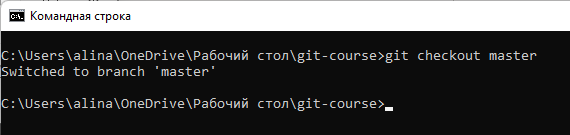


Git пишет, что мы переключились на ветку readme. Давайте убедимся в этом.



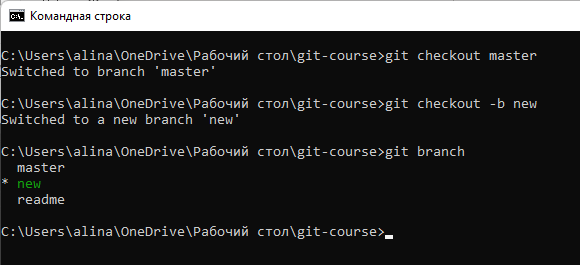
Если надо переключиться на другую ветку, то мы пишем git checkout “название ветки”.

1. Перейдем в главную ветку.



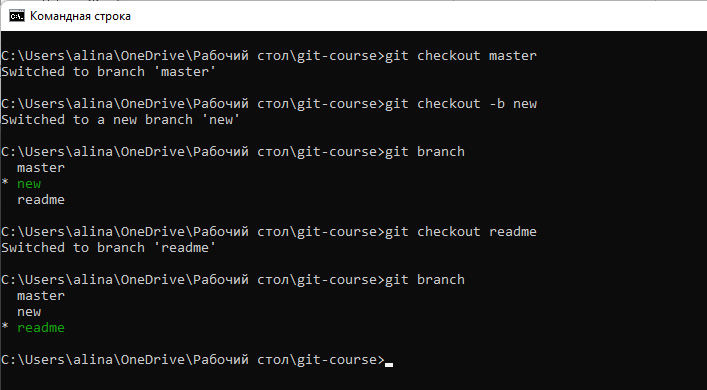
Теперь попробуем создать ветку другим способом.

1. Если нам надо создать ветку и переключиться на нее сразу же, в этом нам поможет команда git checkout -b “название ветки”.

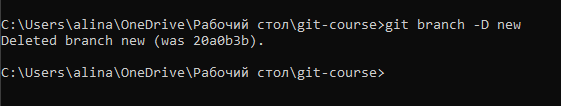


Теперь в нашем репозитории есть три ветки, и мы находимся на ветке new.

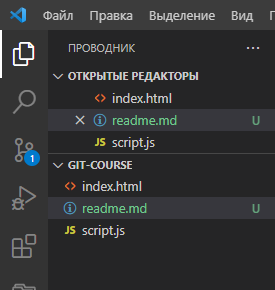
1. Но мы будем работать в ветке readme. Поэтому переключимся на нее обратно.



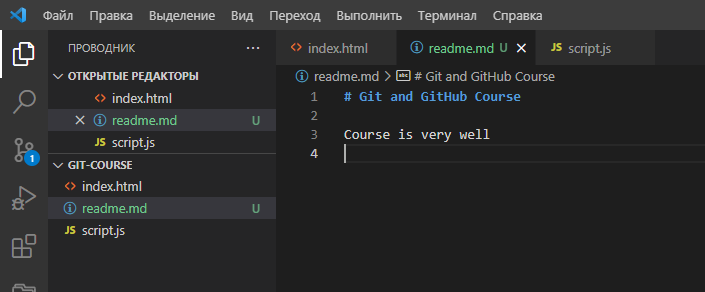
1. Удалим теперь ветку new. Для этого мы напишем git branch -D new



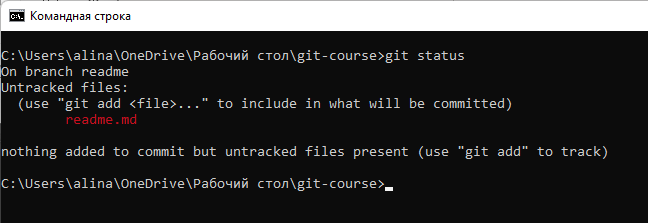
1. Так как мы сейчас на ветке readme, в которой нет файлов, создадим новый файл readme.md.



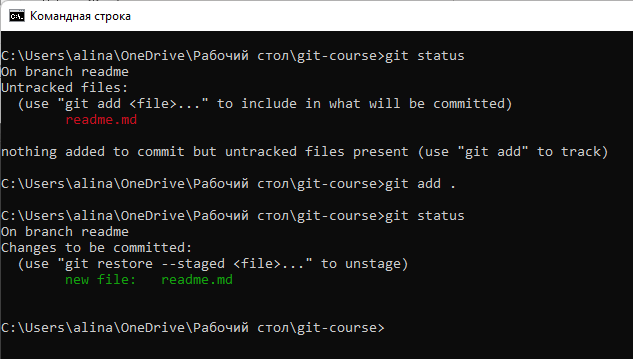
1. Внесем следующий текст.



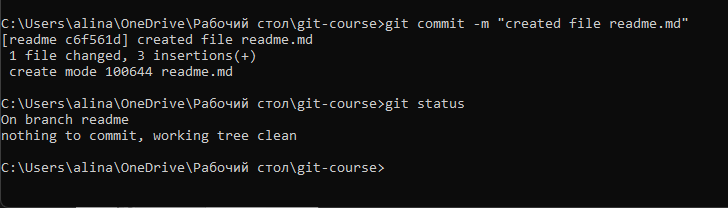
1. Проверим изменения в консоли.



1. Добавим данный файл в наш проект.

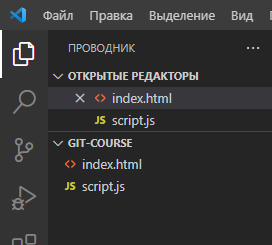


1. Далее необходимо сделать коммит.

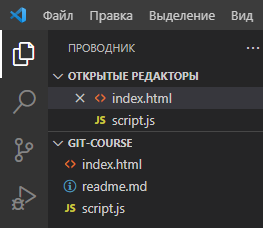


Мы видим, что находимся в ветке readme, коммитить нам нечего. Значит, всё сделали правильно.

ВАЖНЫЙ МОМЕНТ!!!!! В ветке readme мы создали файл readme.md, но в ветке master его нет. Если мы переключимся на ветку master, мы подробно это увидим.

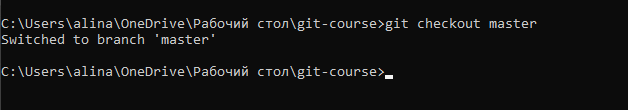


Если мы переключимся снова на readme, то файл появится.

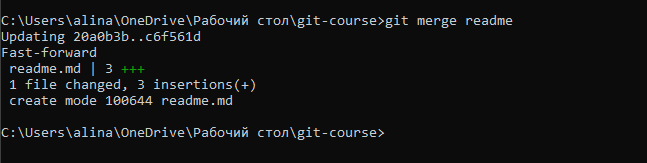


Тем самым мы видим, что у нас есть 2 разных версии нашего проекта.

1. Переключимся обратно на ветку master

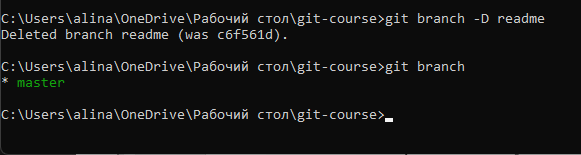


1. Теперь мы хотим совместить две эти ветки. Для этого присутствует команда git merge



Мы видим добавленный файл readme.md.

1. Теперь мы можем удалить ветку readme.md



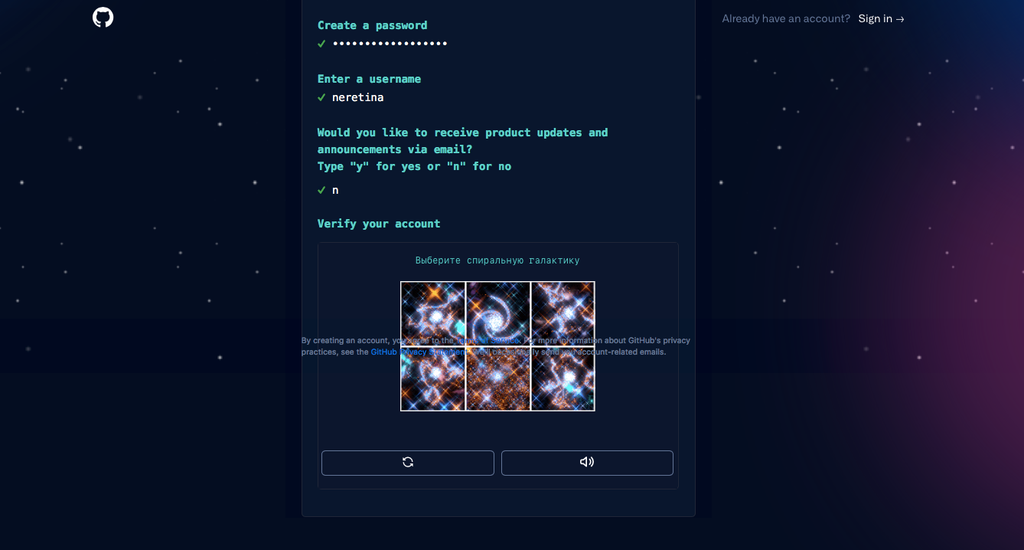
Теперь у нас осталась одна ветка master.

Работа с GitHub.

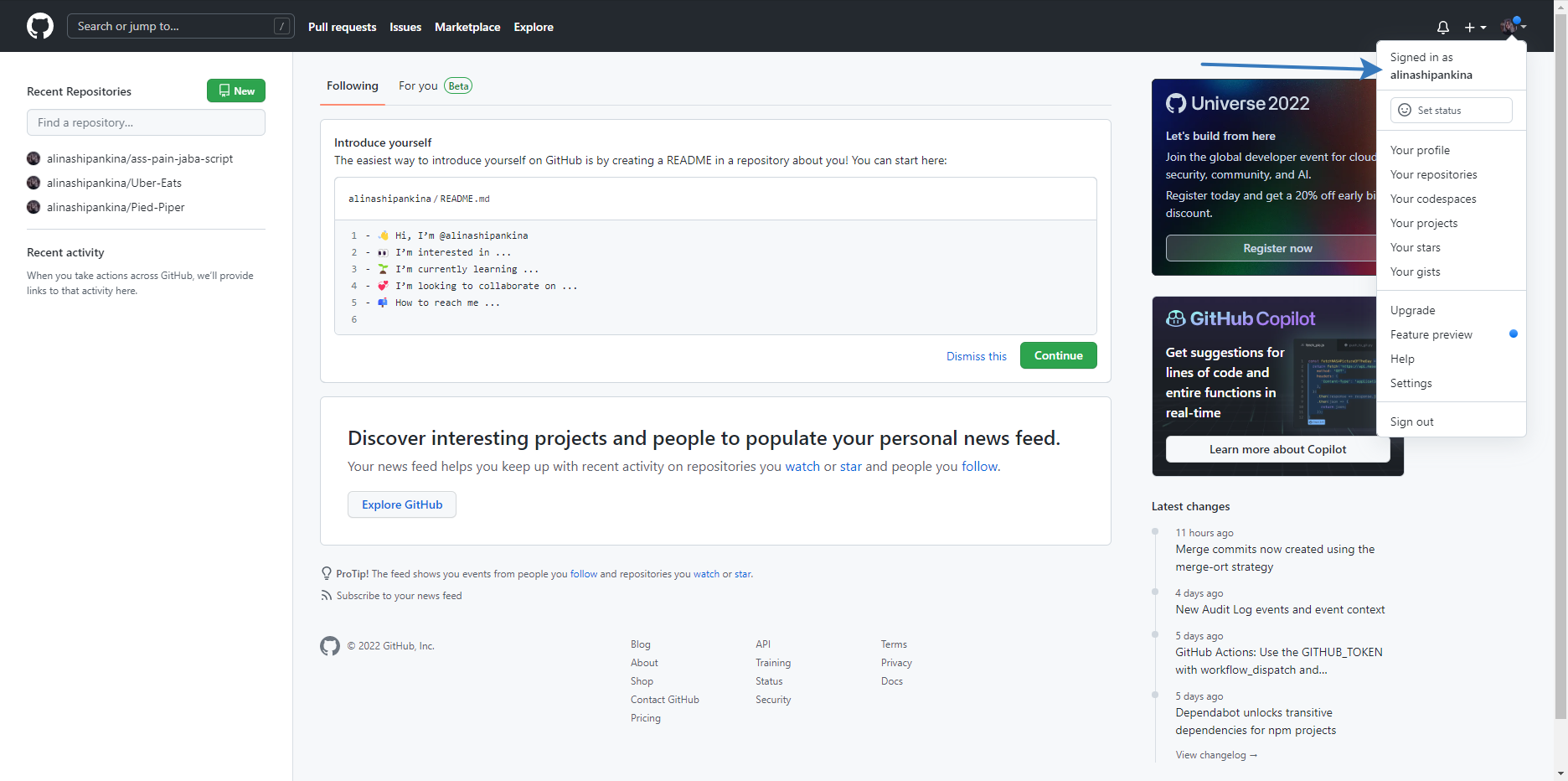
GitHub – это удаленная система, которая позволяет отображать визуально, что у нас происходит локально в Git.

1. Заходим в GitHub и заводим свой аккаунт.

Чтобы выложить на GitHub проект, нужна регистрация на сайте сервиса. Первоначально необходимо указать адрес электронной почты, который станет логином для учётной записи. Сюда будут приходить все уведомления, и с помощью электронной почты можно будет восстановить доступ к GitHub в случае необходимости.

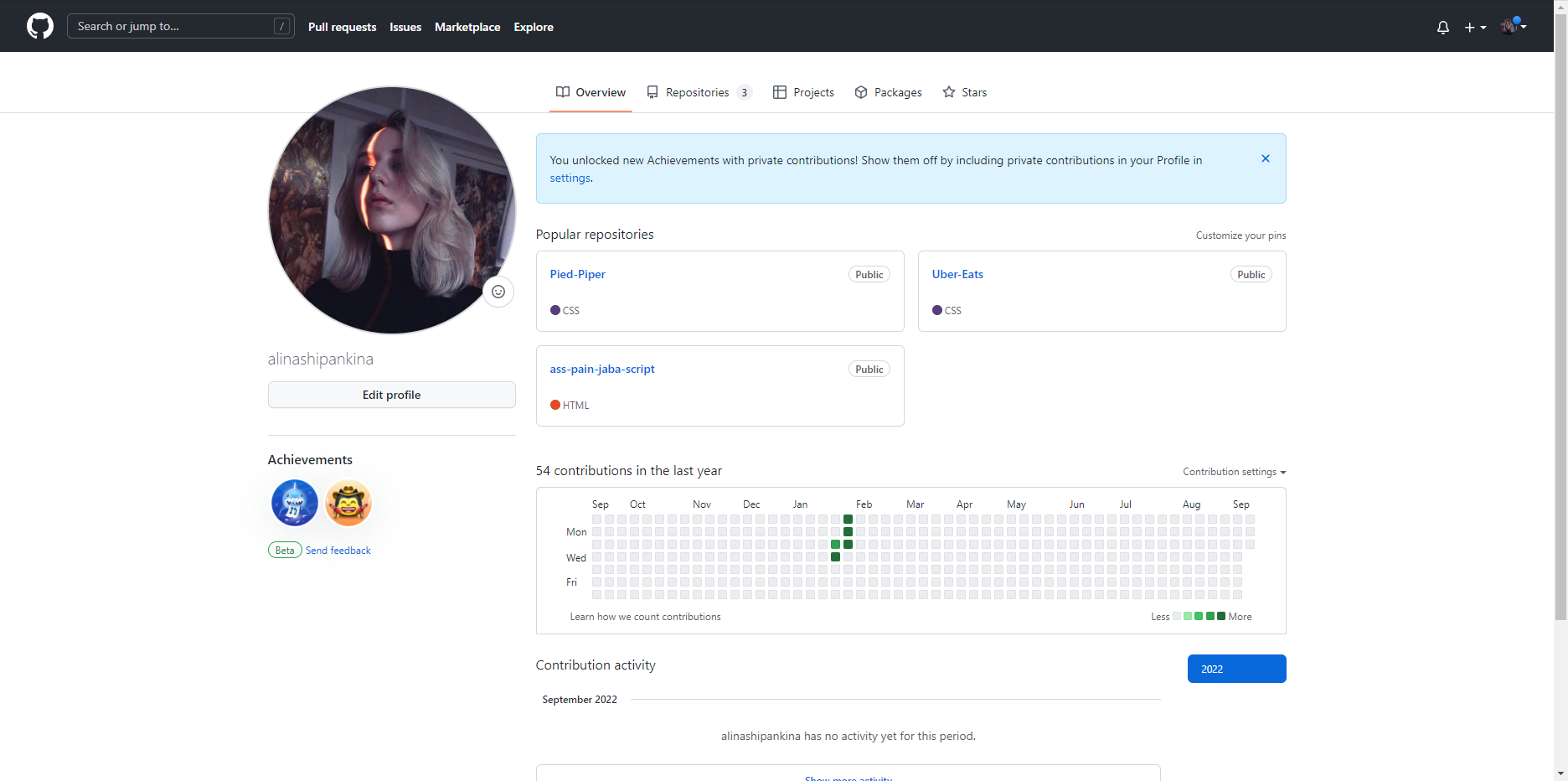


При регистрации попросят указать email, имя пользователя, согласиться на рассылку или отказаться



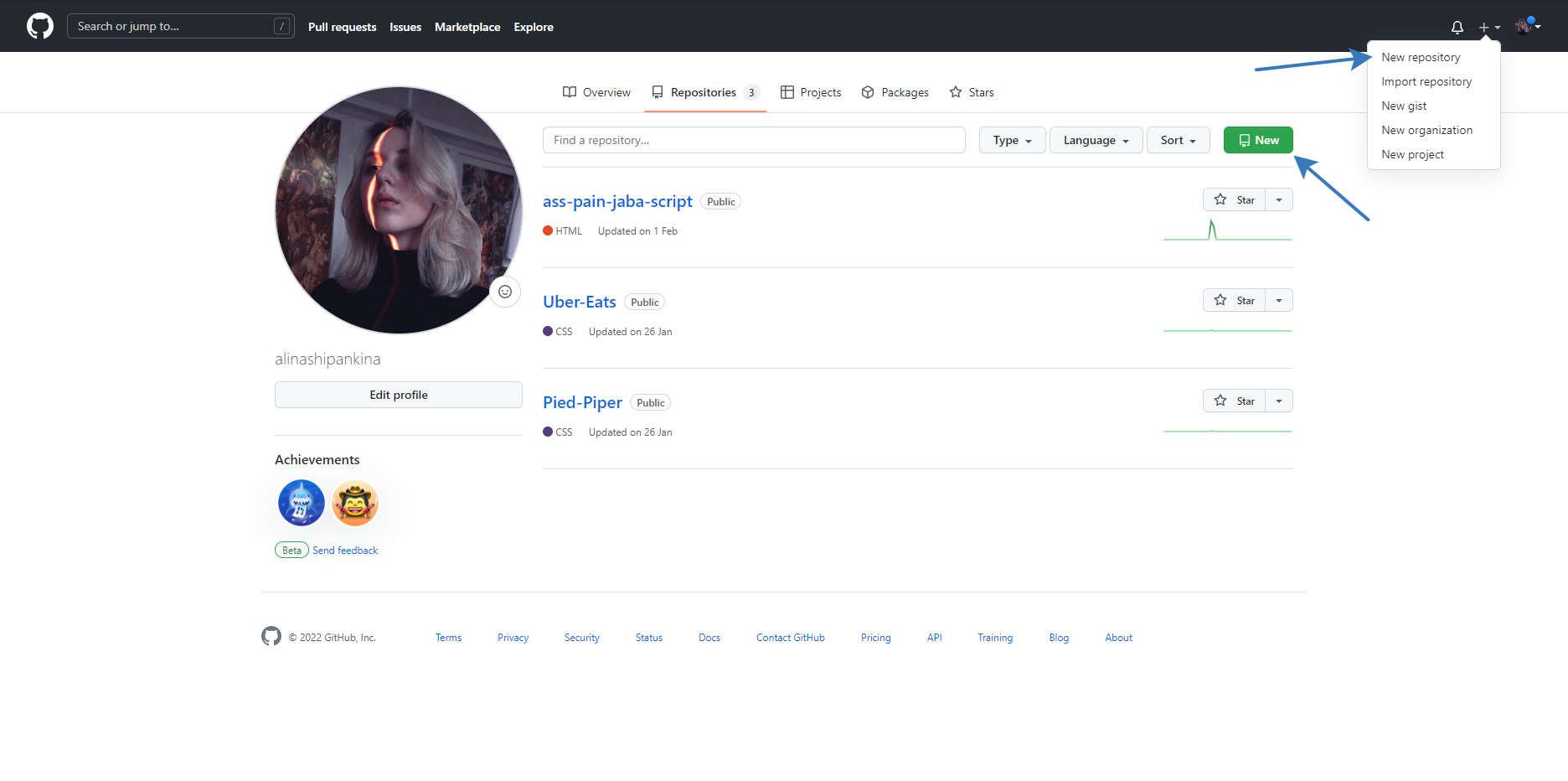
Справа вы увидите свой профиль.

1. Нажимаем на кнопку Your profile. Здесь мы видим различную информацию о своем профиле (у вас пока ее нет).

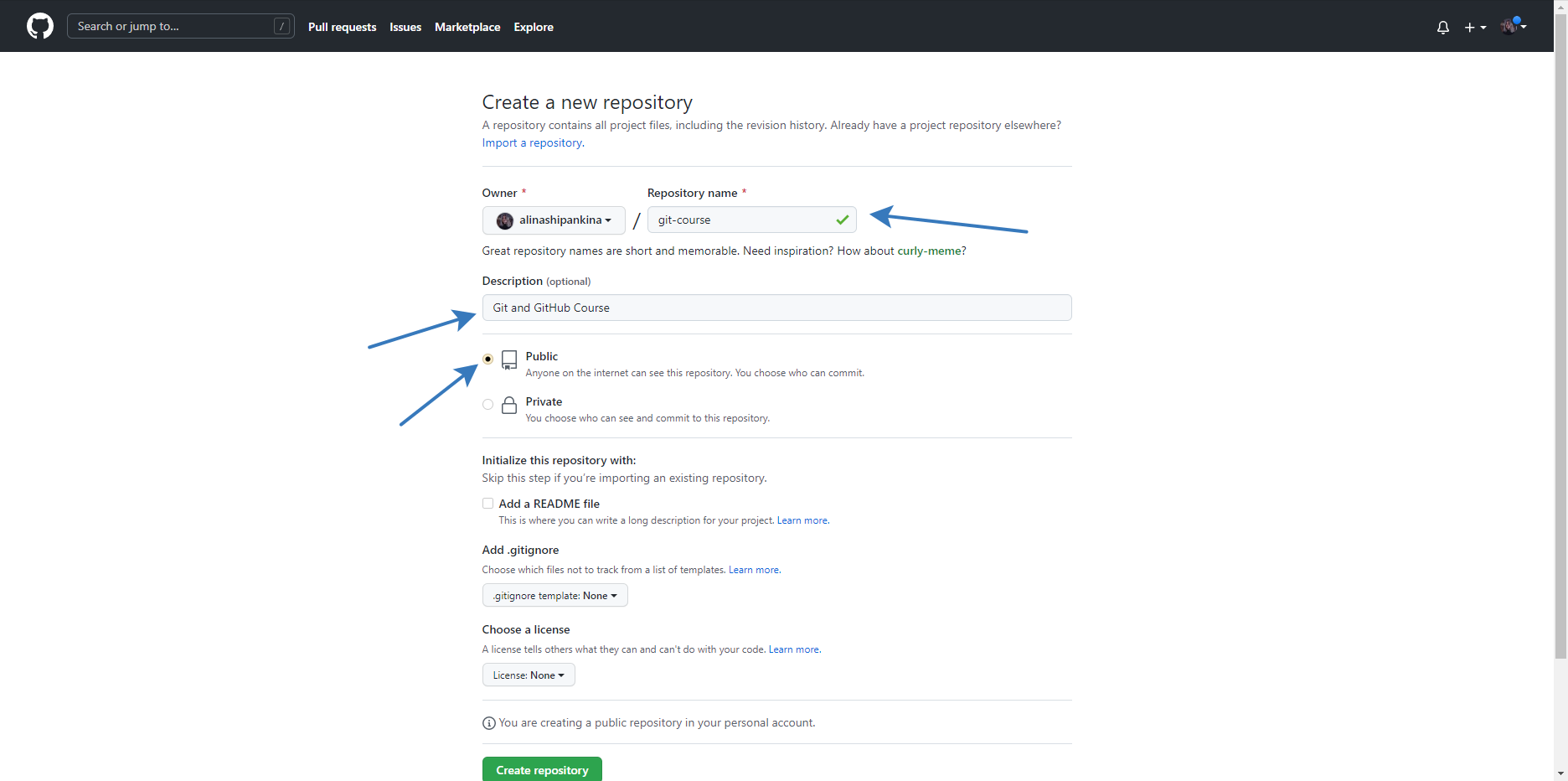


При регистрации в GitHub необходимо запомнить свои логин и пароль, так как при подключении Git к нашему репозиторию удаленному это пригодится.

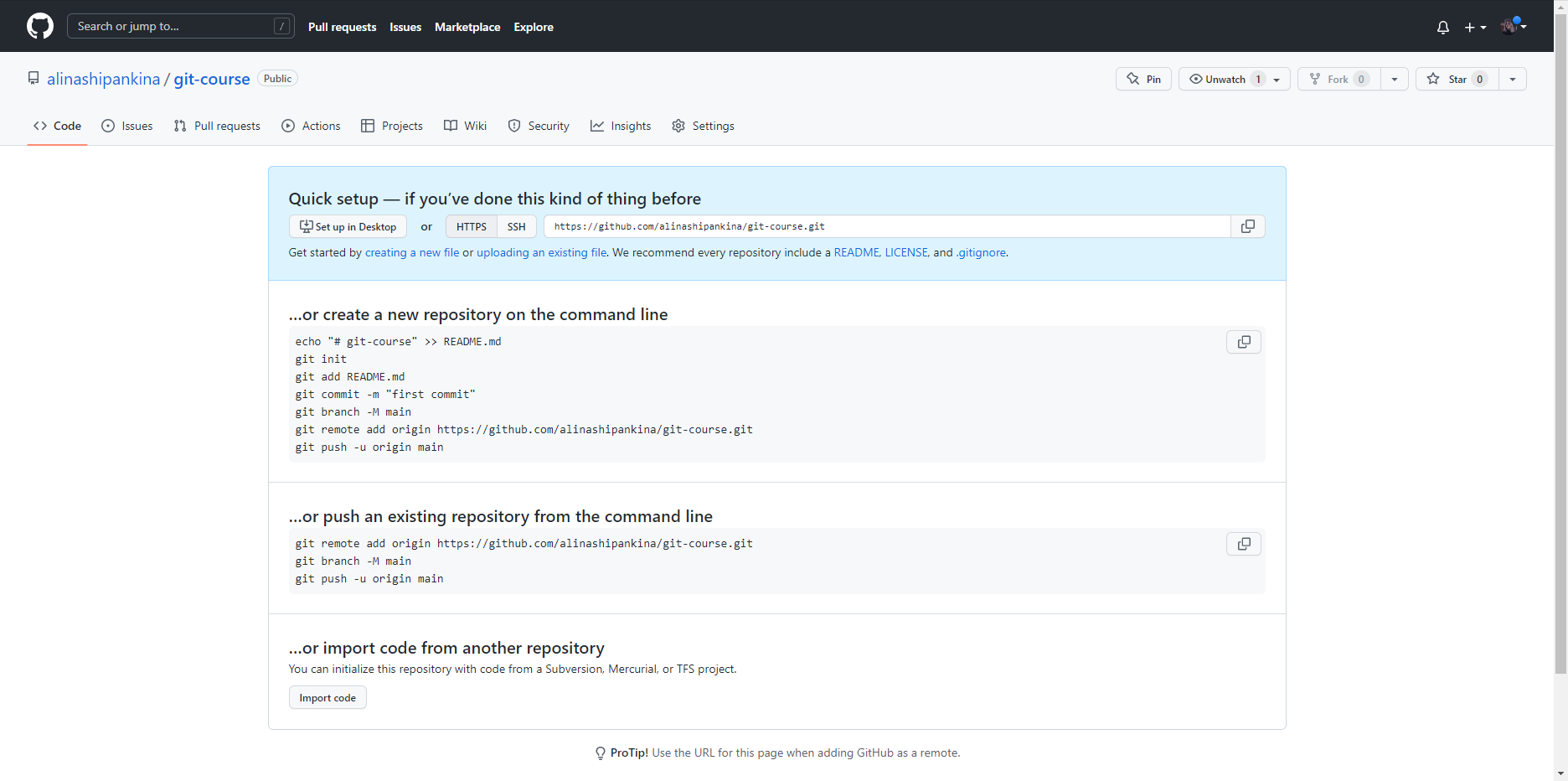
1. Перейдем в список репозиториев. Кнопочка Your Repositories. Создадим по одному из предложенных способов репозиторий.



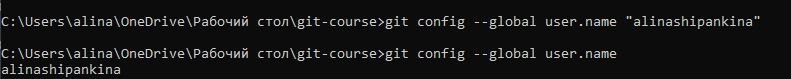
1. Вносим данные в соответствии с картинкой



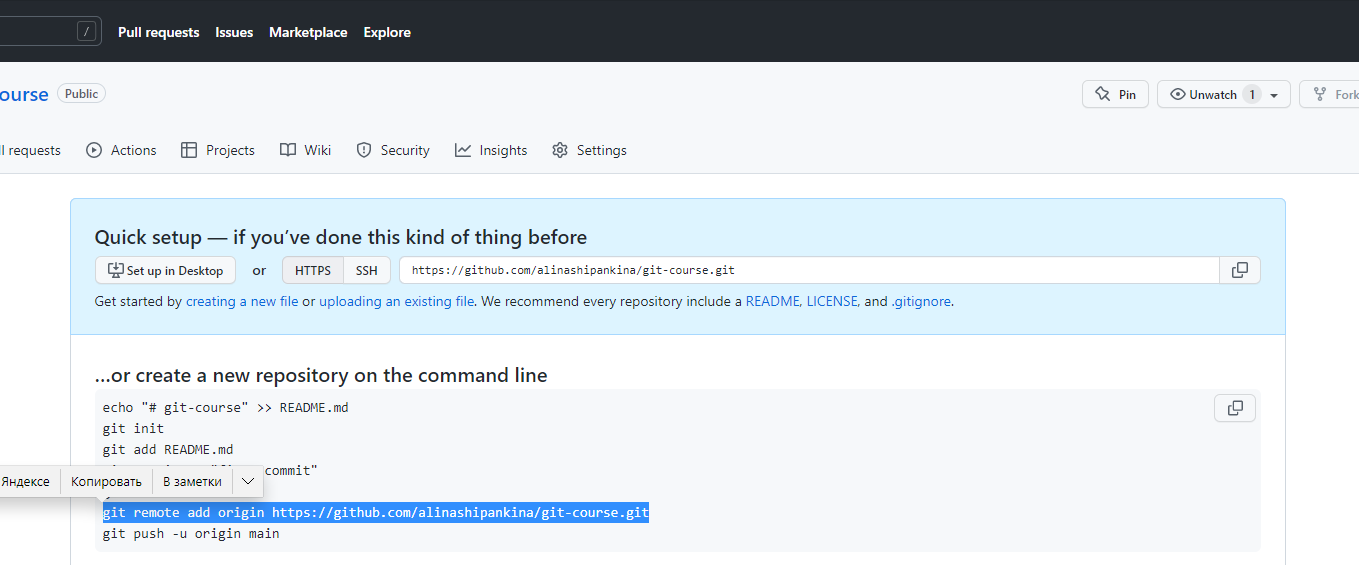
1. Нажимаем Create repository.
2. Далее мы попадаем на такую страницу. Но разбирать мы ее не будем. Если интересно, можете в дальнейшем изучить ее в индивидуальном порядке.

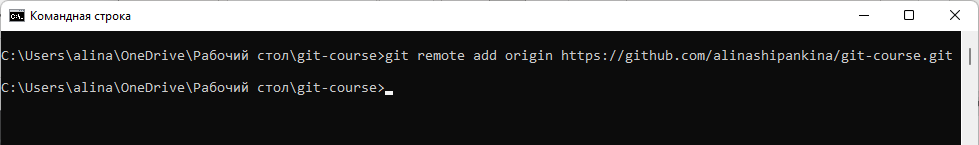


1. Возвращаемся локально к Git’у. И здесь нам необходимо ввести наши данные, чтобы g\Git смог связаться с нашим репозиторием в GitHub. Для этого есть команда git config --global user.name “…” и git config --global user.email “…”. ГЛАВНОЕ, чтобы имя и e-mail совпадали с данными в GitHub.



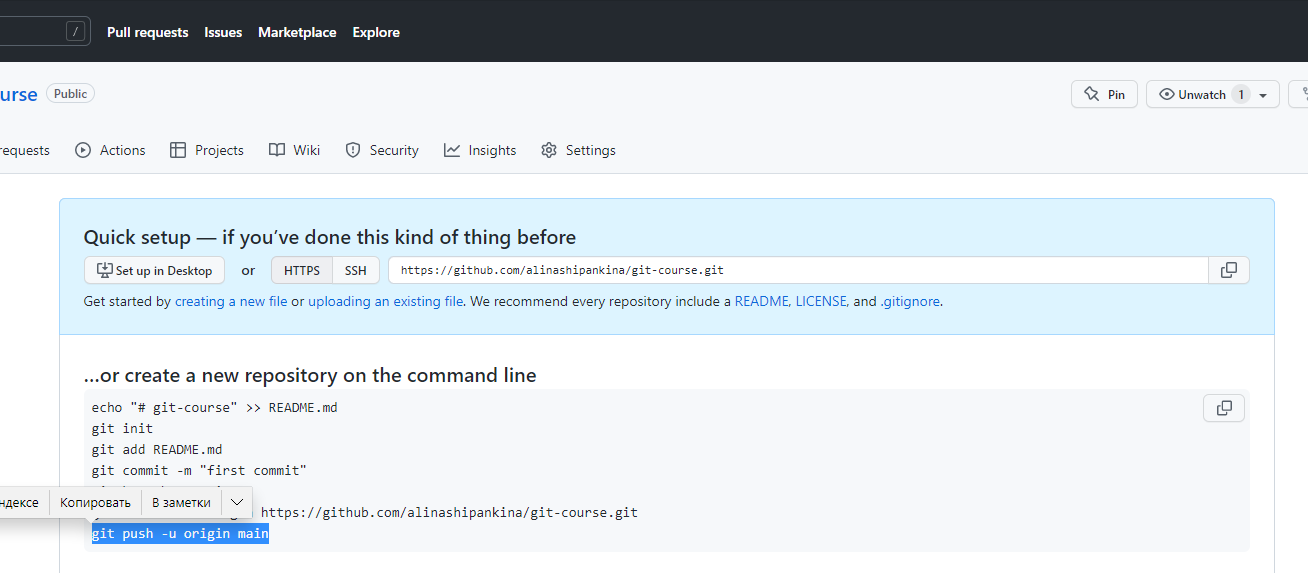
1. Теперь мы свяжем локальный репозиторий с внешним. Копируем выделенную строчку. Вставляем ее в консоль и нажимаем Enter.

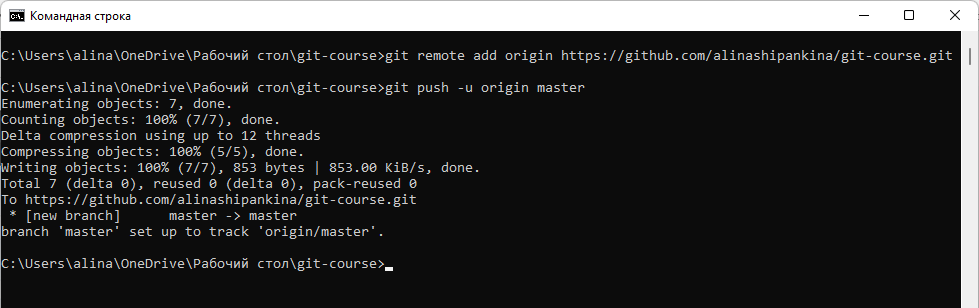




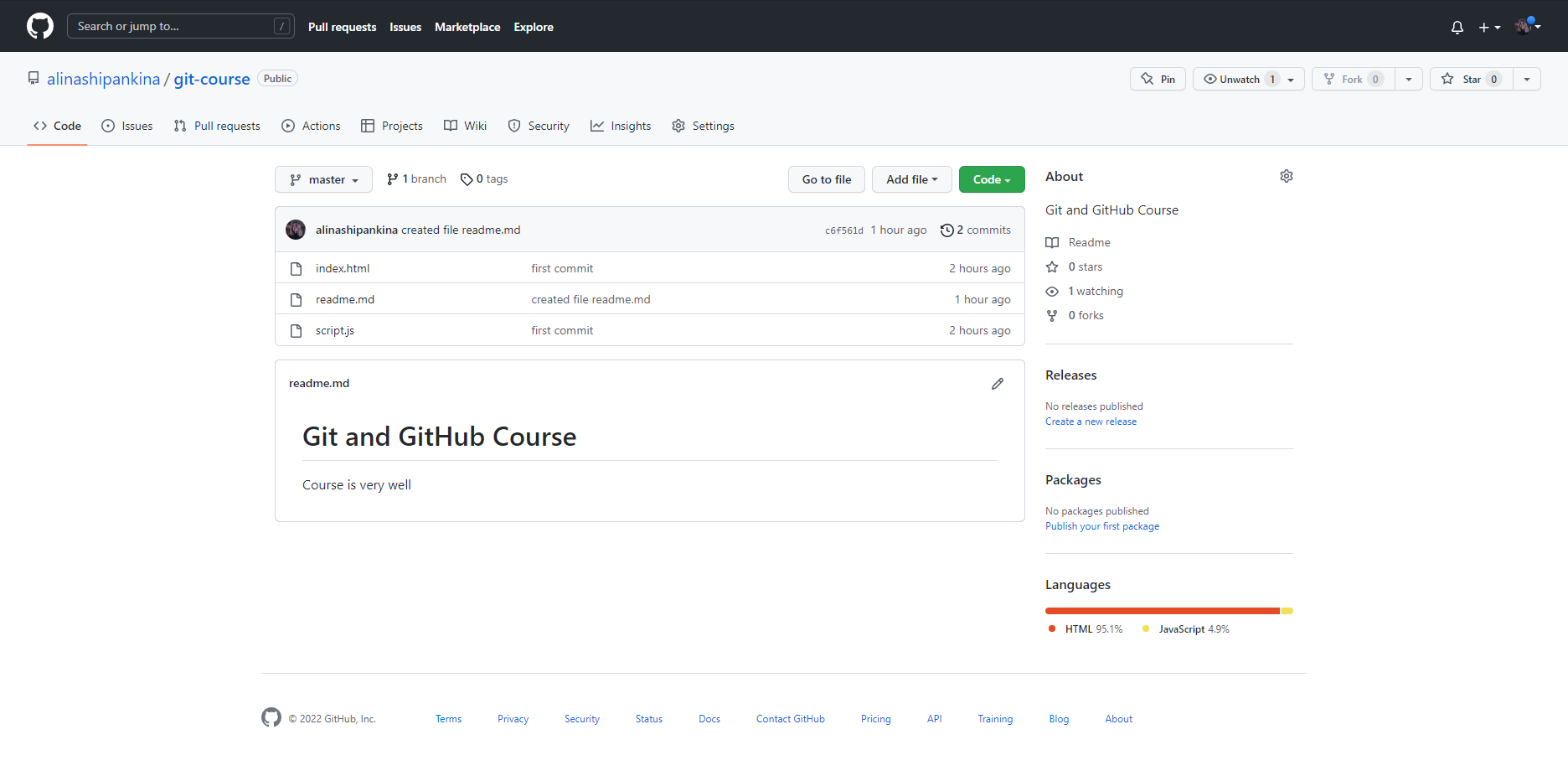
Все, аккаунт привязан.

1. Теперь нам надо залить все, что есть локально, во внешний репозиторий. Для этого копируем строчку ниже, но вместо main пишем master.





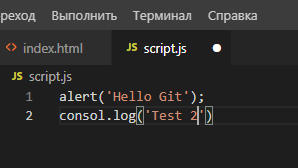
1. Обновим нашу страницу на GitHub. И мы видим вот такую красоту.



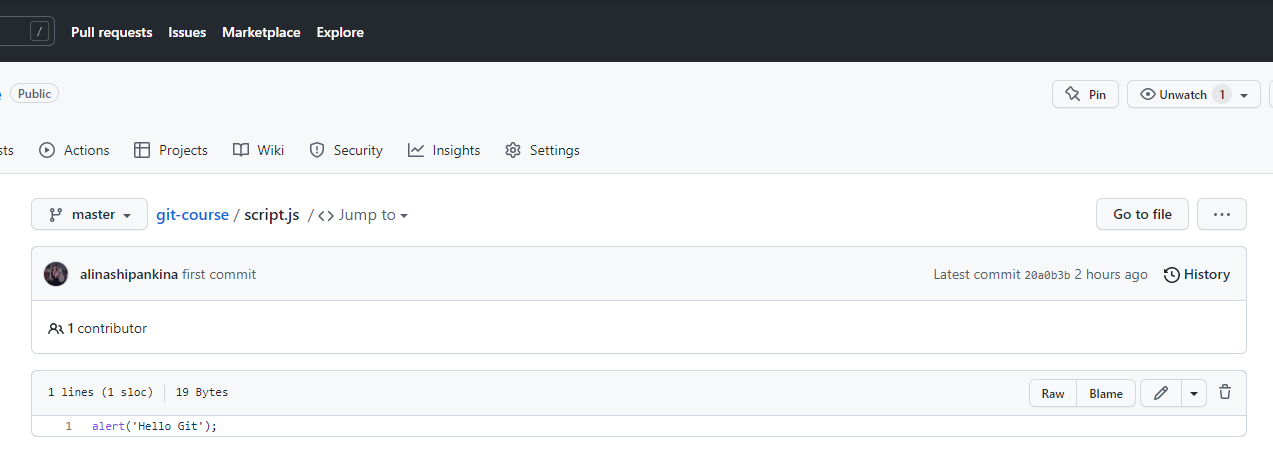
Здесь присутствуют различные файлы, которые мы создавали, причем написано, когда они были созданы.

Но на этом наша практическая работа не заканчивается)))))

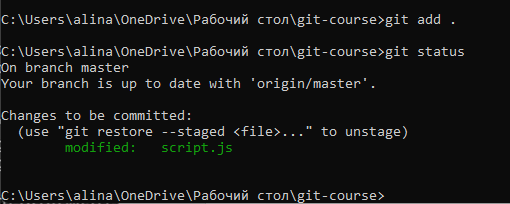
1. Изменим немного файл script.js локально. Напишем console.log(‘Test 2’)



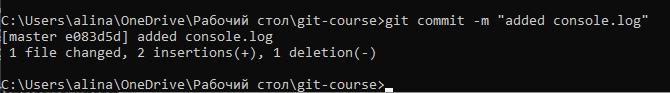
Но в GitHub эти изменений нет.



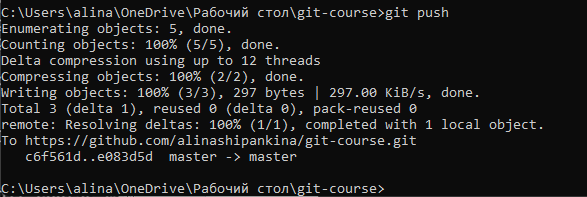
1. Чтобы зафиксировать наши изменения и опубликовать нам надо добавить изменения командой git add .



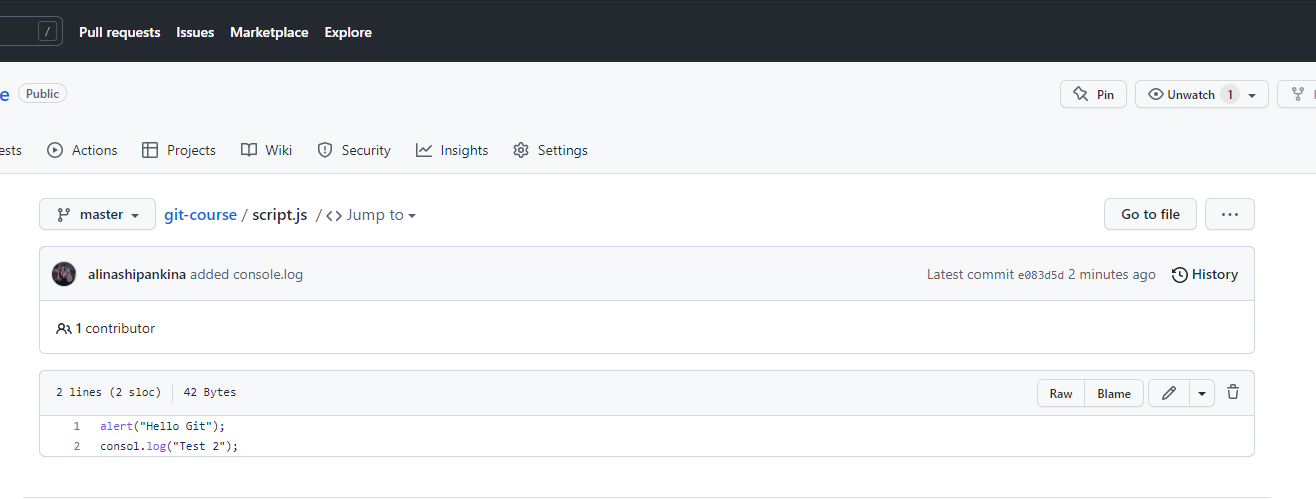
1. Дальше мы делаем git commit -m “added console.log”

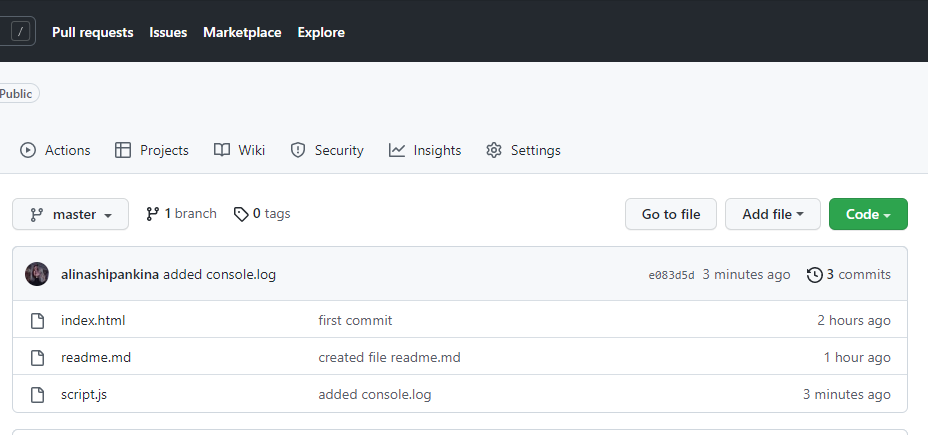


1. Чтобы залить это всё на GitHub, мы пишем git push



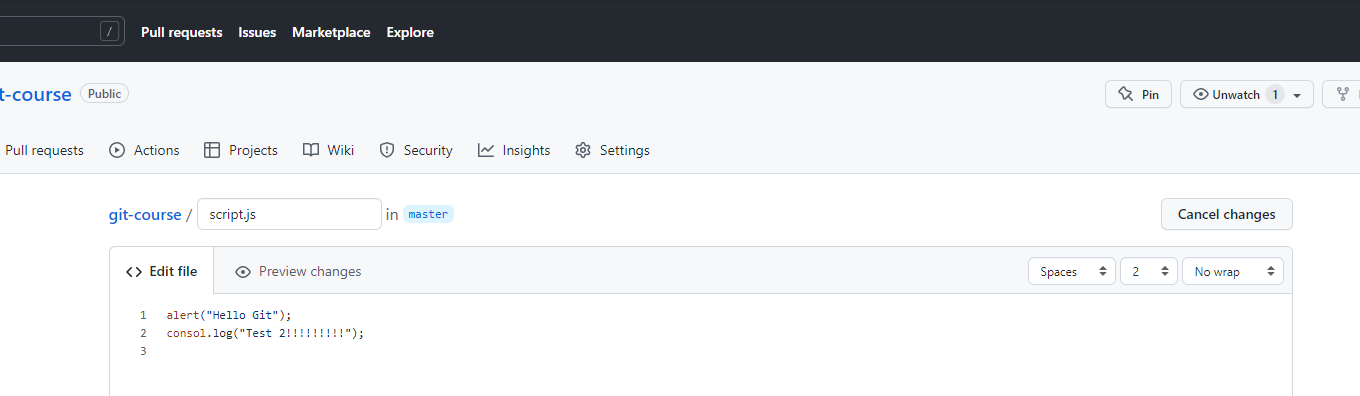
1. Проверим изменения в удаленном репозитории





Файл был изменен 3 минуты назад. Всё есть. Супер.

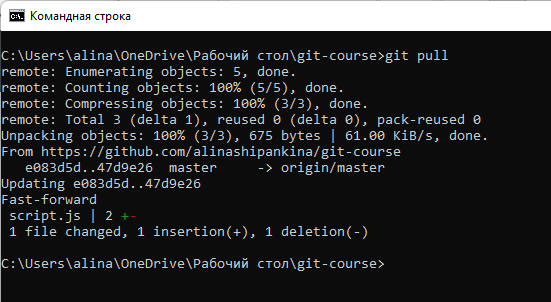
1. Добавим на удаленном репозитории некоторую информацию.



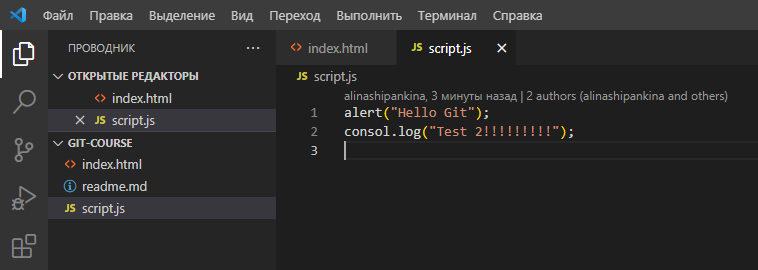
Пишем коммит на основной ветке. Нажимаем Commit changes.

Но при этом в локальном репозитории ничего не изменилось.

1. Чтобы добавить на локальный репозиторий изменения со внешнего напишем git pull в консоли.



Зайдем в VS Code. Изменения внесены.



Поздравляю!!!!! Практическое задание выполнено!!!!! Вы проделали большой путь.

Данная практическая работа содержит в себе основы основ, которые вы сможете встретить при дальнейшем трудоустройстве в компании.

Используйте этот гайд, как шпаргалку, смотрите гайды от ведущих программистов. Уверена, вы станете крутыми разработчиками😊