# Что такое, зачем.

Ситуация, в которой электронный документ за время своего существования претерпевает ряд изменений, достаточно типична. При этом часто бывает важно иметь не только последнюю версию, но и несколько предыдущих. В простейшем случае можно просто хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам, поэтому были разработаны средства для автоматизации этой работы.

Кроме того, часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, то один из них может случайно отменить изменения, сделанные другим. Системы управления версиями отслеживают такие конфликты и предлагают средства их решения. Большинство систем может автоматически объединить (слить) изменения, сделанные разными разработчиками. Однако такое автоматическое объединение изменений, обычно, возможно только для текстовых файлов и при условии, что изменялись разные (непересекающиеся) части этого файла. Такое ограничение связано с тем, что большинство систем управления версиями ориентированы на поддержку процесса разработки программного обеспечения, а исходные коды программ хранятся в текстовых файлах. Если автоматическое объединение выполнить не удалось, система может предложить решить проблему вручную.

Введем используемую в системах контроля версий терминологию. Набор файлов, с которым мы работаем в данный момент, называется рабочая копия (*working copy*). После того, как решено, что все нужные изменения на данный момент внесены, и об этом можно сообщить системе контроля версий, разработчик производит отправку изменений в репозиторий (*repository*). Репозиторий – это хранилище для нашего проекта, которое обслуживает система контроля версий. Сама операция отправки изменений называется *commit*, на русском языке ее так и называют – коммит. При необходимости синхронизации изменений с удаленным репозиторием необходимо использовать операции push (отправка изменений) и pull (вытягивание изменений).

Рассматриваемая далее система git представляет из себя децентрализованную систему контроля версий, общий вид которой показан на рисунке 1.

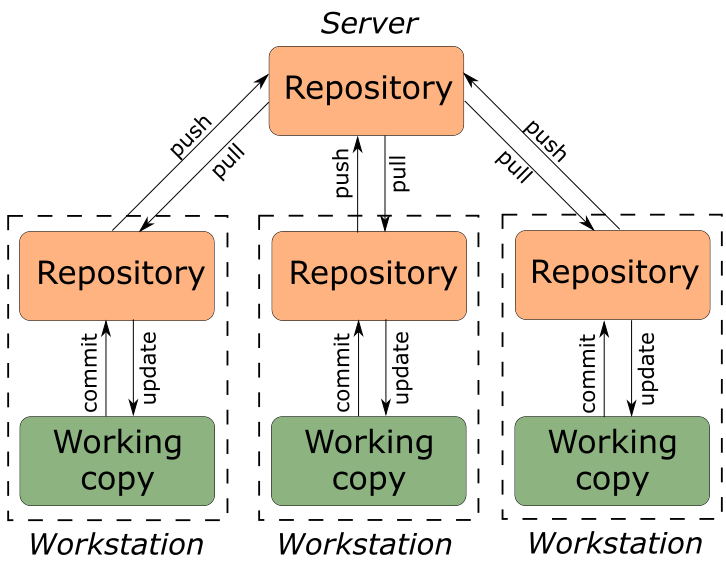


Рисунок 1- Архитектура распределенных систем контроля версий

# Порядок работы с Git

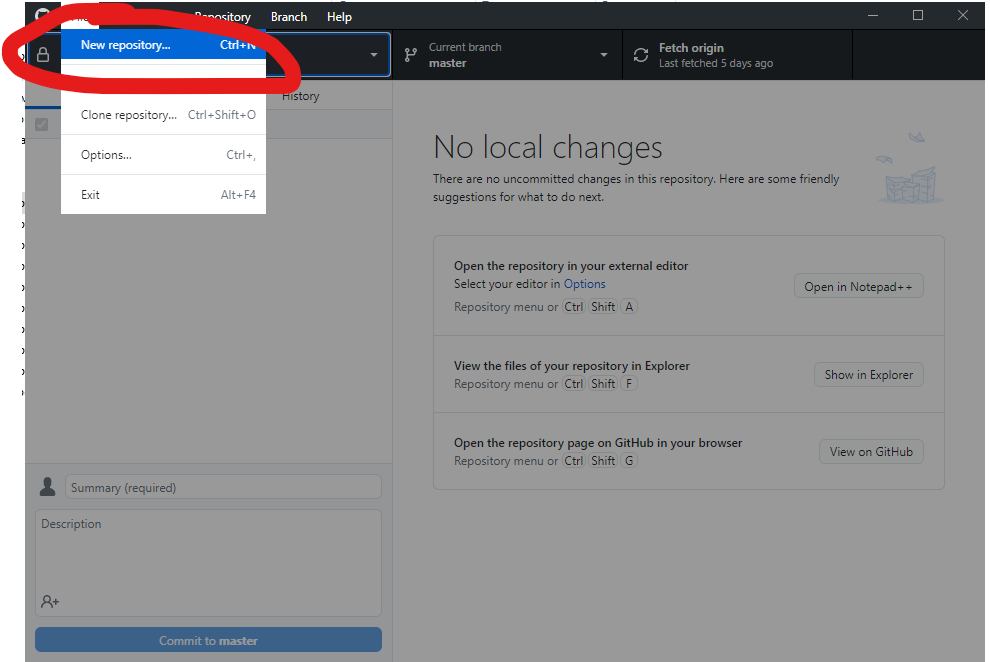
Для начала работы скачайте GitHub Desktop и зарегистрируйтесь на сайте github.com. Дальнейшие шаги будут описаны с учетом использования этой системы.

При запуске программы авторизуетесь (sign in) в Github Desktop с использованием учетных данных github.com.

Далее рассмотрим несколько базовых сценариев работы с данной системой контроля версий.

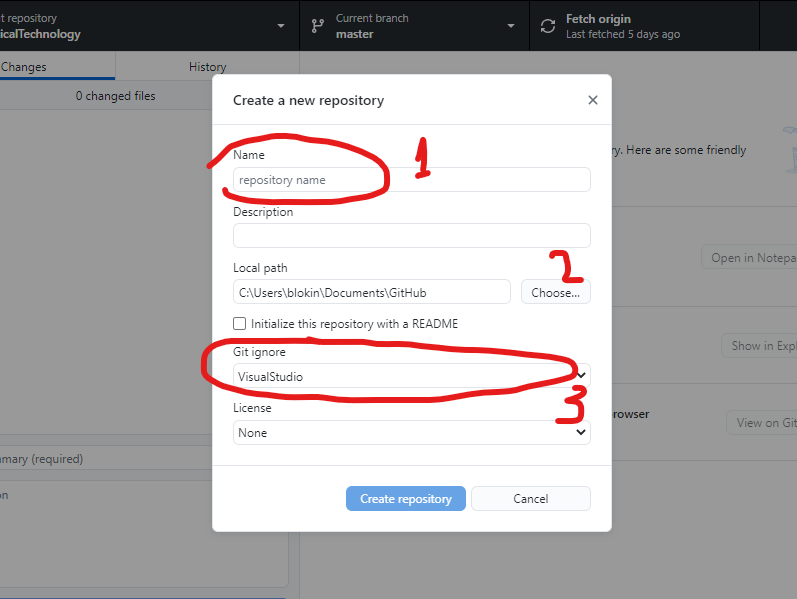
## Сценарий 1: есть новый проект, его необходимо выгрузить в GitHub

Запускаем GitHub Desktop (далее GD), выбираем пункт меню создания репозитория:

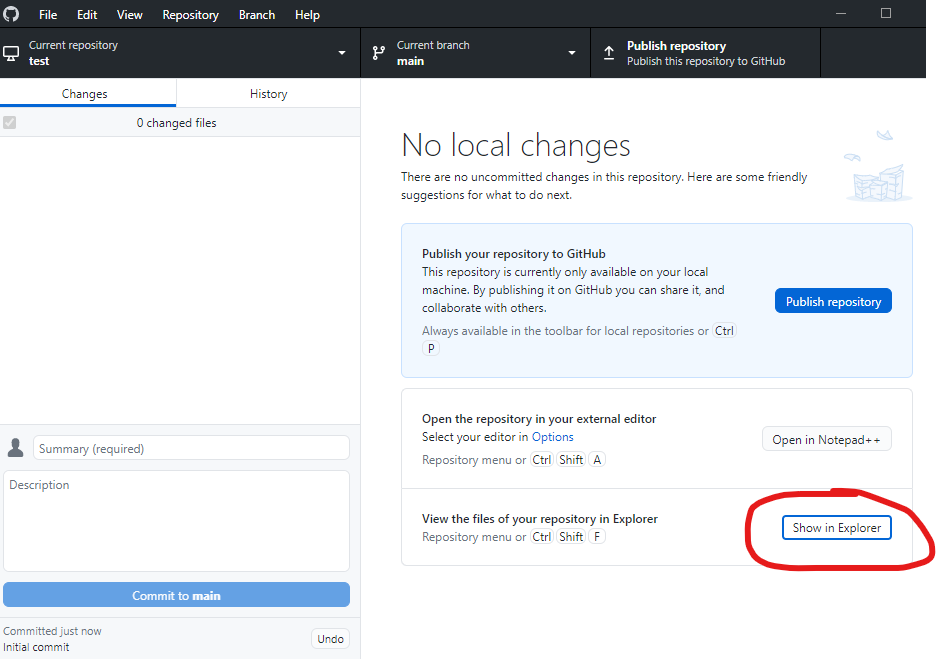


Заполняем параметры репозитория:

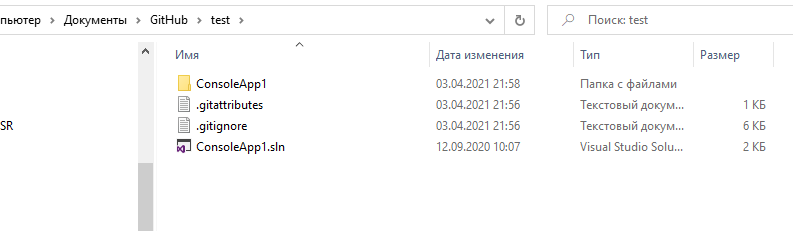
1. Его название (обычно название проекта)
2. Расположение на локальном компьютере
3. В выпадающем списке Git ignore выбираем Visual Studio (важно, чтобы не хранились изменения промежуточных файлов, тем самым засоряя репозиторий)



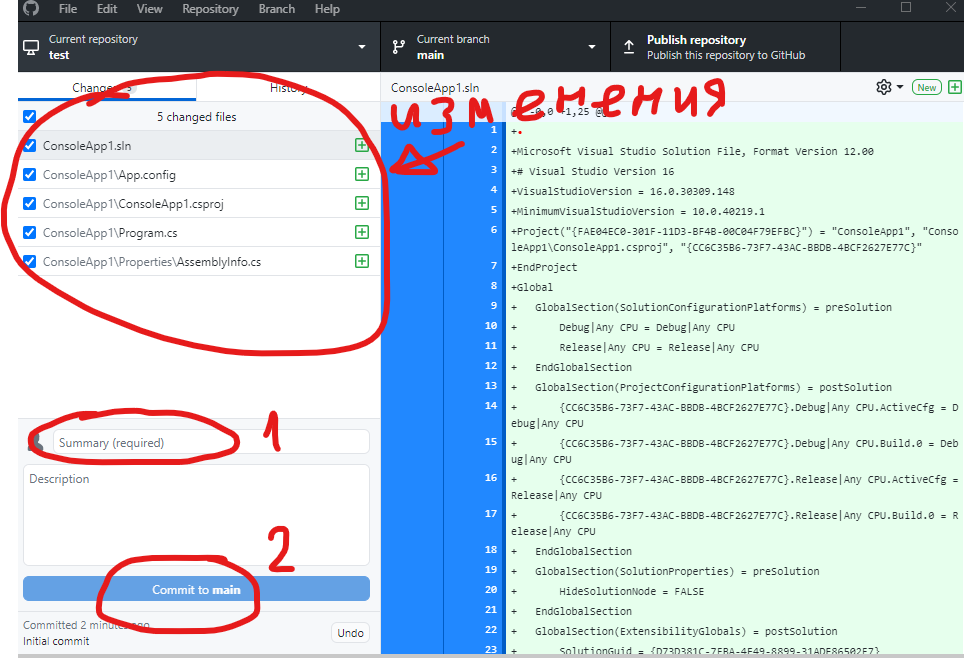
Открываем папку репозитория:



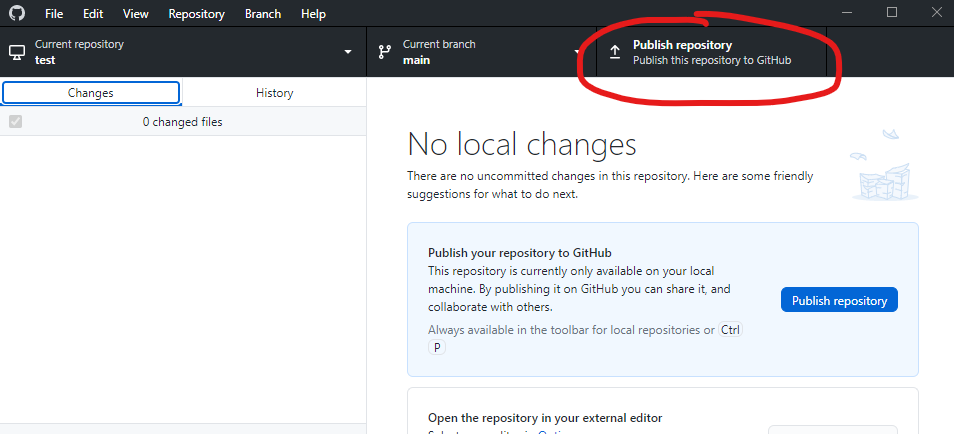
Копируем в эту папку файлы проекта:



Возвращаемся в Github Desktop, если все сделано правильно, на вкладке Changes должны появиться новые файлы. Чтобы их зафиксировать, необходимо ввести краткое описание в поле Summary, после чего нажать Commit:

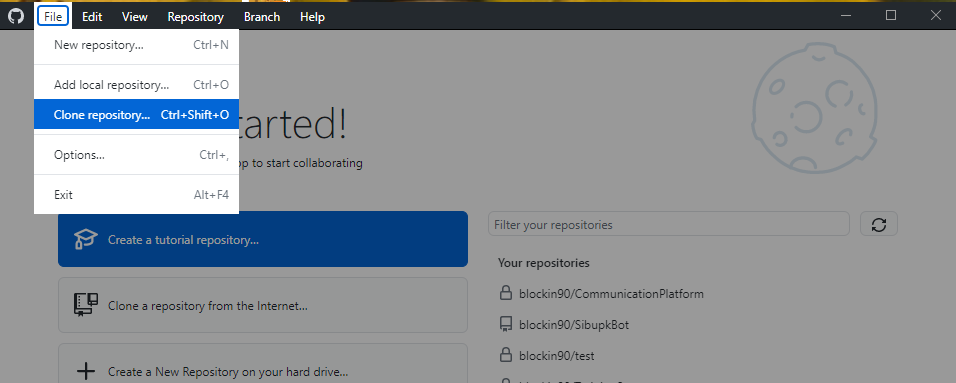


После этого выгружаем файлы в удаленный репозиторий на Github:

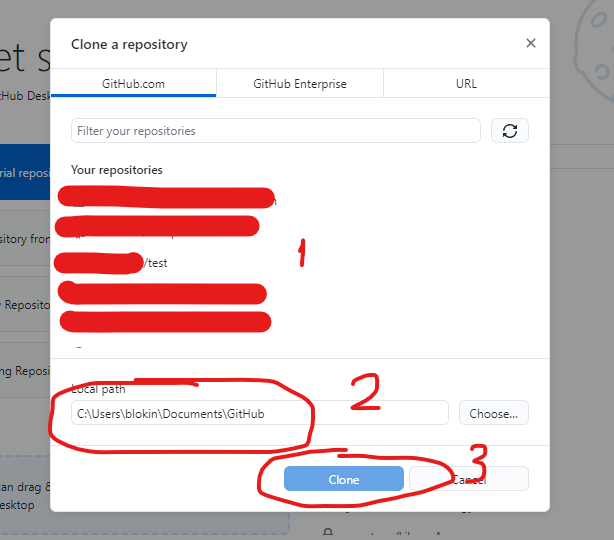


## Сценарий 2: проект уже на GitHub’е, нужно его скопировать на **другой** компьютер, чтобы продолжить разработку

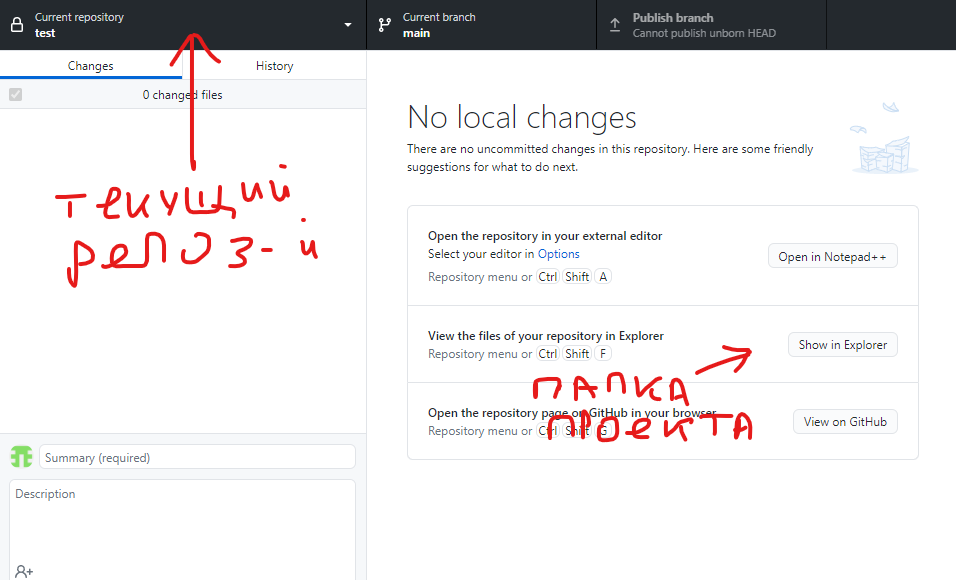
Запускаем GD, выбираем пункт меню “Clone repository”:

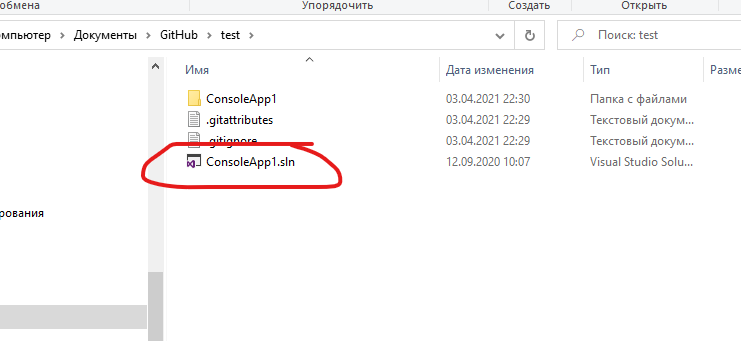


В следующем окне выбираем нужный репозиторий (1) и папку, куда мы хотим его скопировать (2) и нажимаем «Clone» (3):



После этого переходим в папку проекта, и запускаем .sln файл:

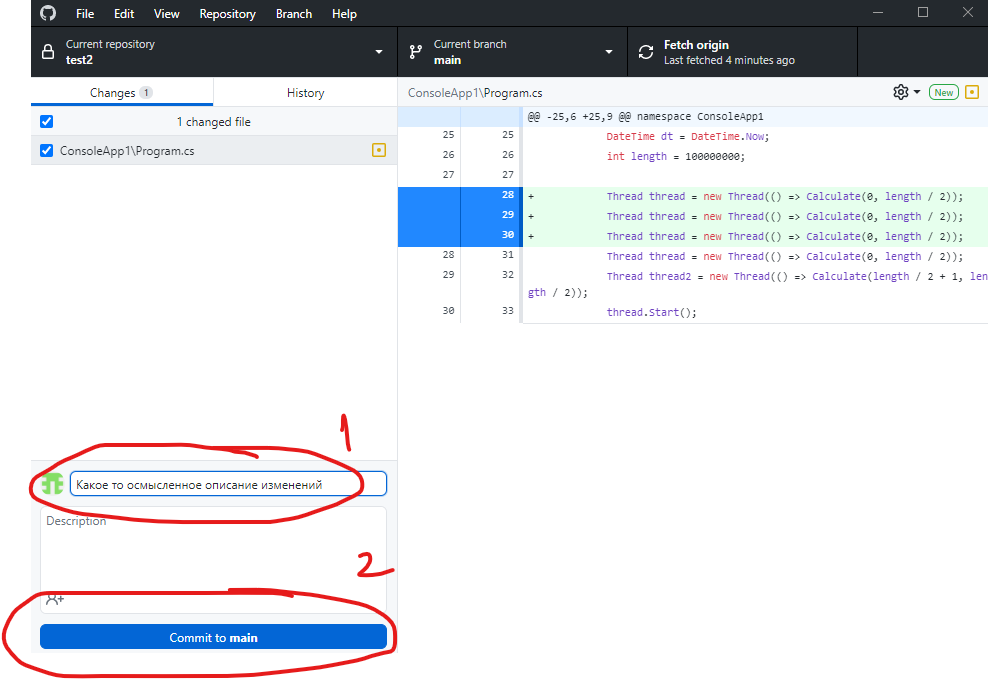




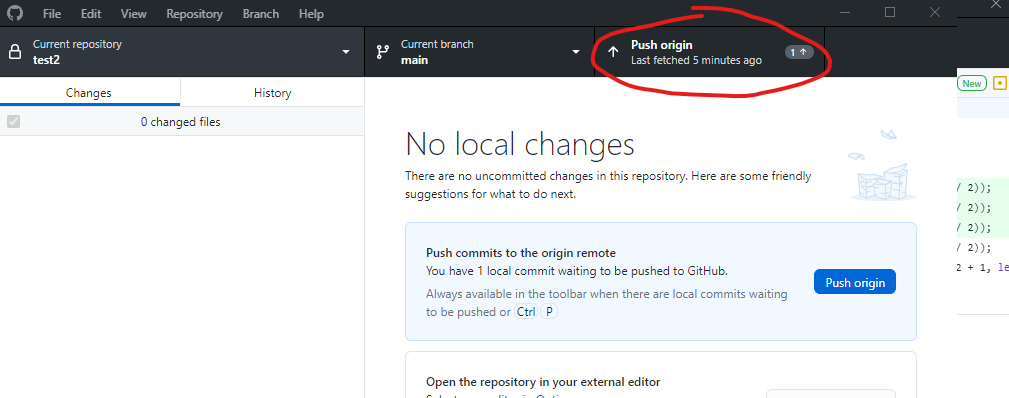
После этого запустится Visual Studio с вашим проектом.

## Сценарий 3: проект уже на GitHub’е, а также на вашей машине, вы внесли какие-то правки и хотите выгрузить их на GitHub.

Запускаете GD, вводите краткое описание изменений (1), нажимаете Commit (2).

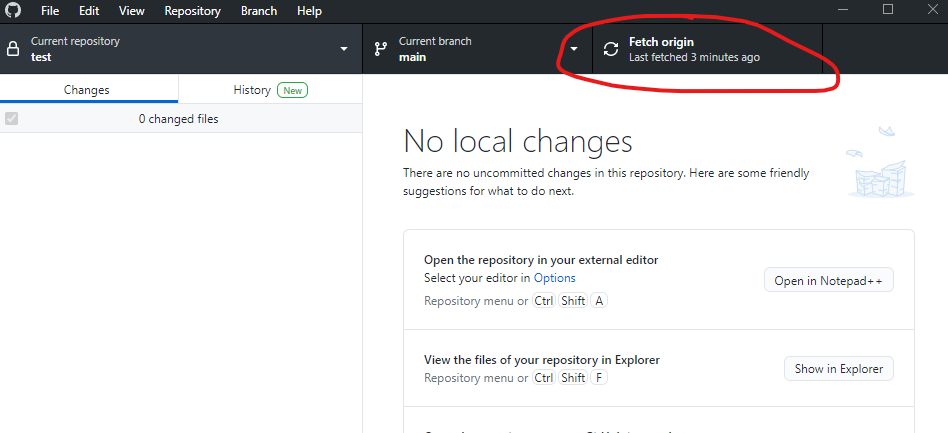


Указанные действия выполнят локальную фиксацию изменений. Чтобы отправить их на GitHub, нажмите Push Origin:

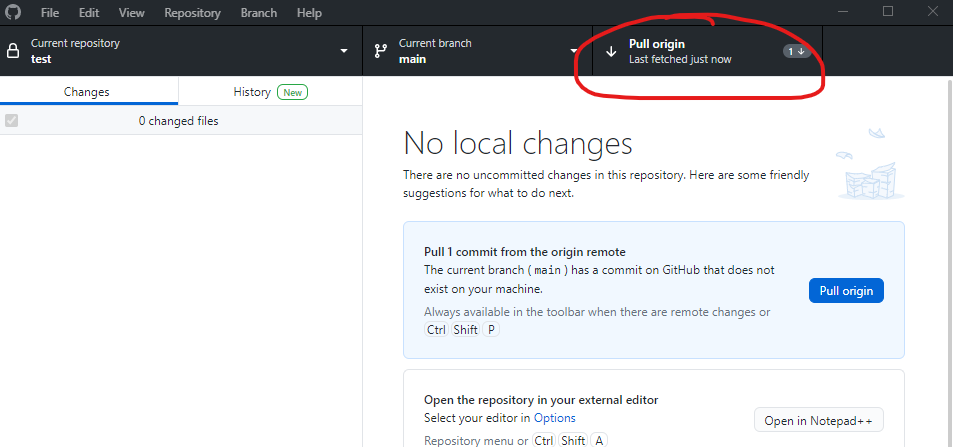


## Сценарий 4: проект уже на GitHub’е, а также на вашей машине. Вы внесли какие-то правки **на другом компьютере**, и они были выгружены на GitHub согласно предыдущему описанию, и теперь вы хотите подтянуть их к себе на компьютер (только последние правки, не весь проект целиком).

Нажимаем Fetch origin:



Если есть изменения, которые можно скачать, текст кнопки изменится на Pull origin и появится кол-во невытянутых изменений:



Нажимаем на эту кнопку, после чего GD скачает актуальную версию изменений.