Оглавление

[1. Однопользовательский, 1 ветка. 2](#_Toc520297747)

[2. Однопользовательский, 3 ветки 4](#_Toc520297748)

[3. Многопользовательский совместный 6](#_Toc520297749)

[4. Многопользовательский двухуровневый (с архитектором) 7](#_Toc520297750)

[5. Многопользовательский, 2 общих ветки (продакшен и препродакшен) 11](#_Toc520297751)

# Однопользовательский, 1 ветка.

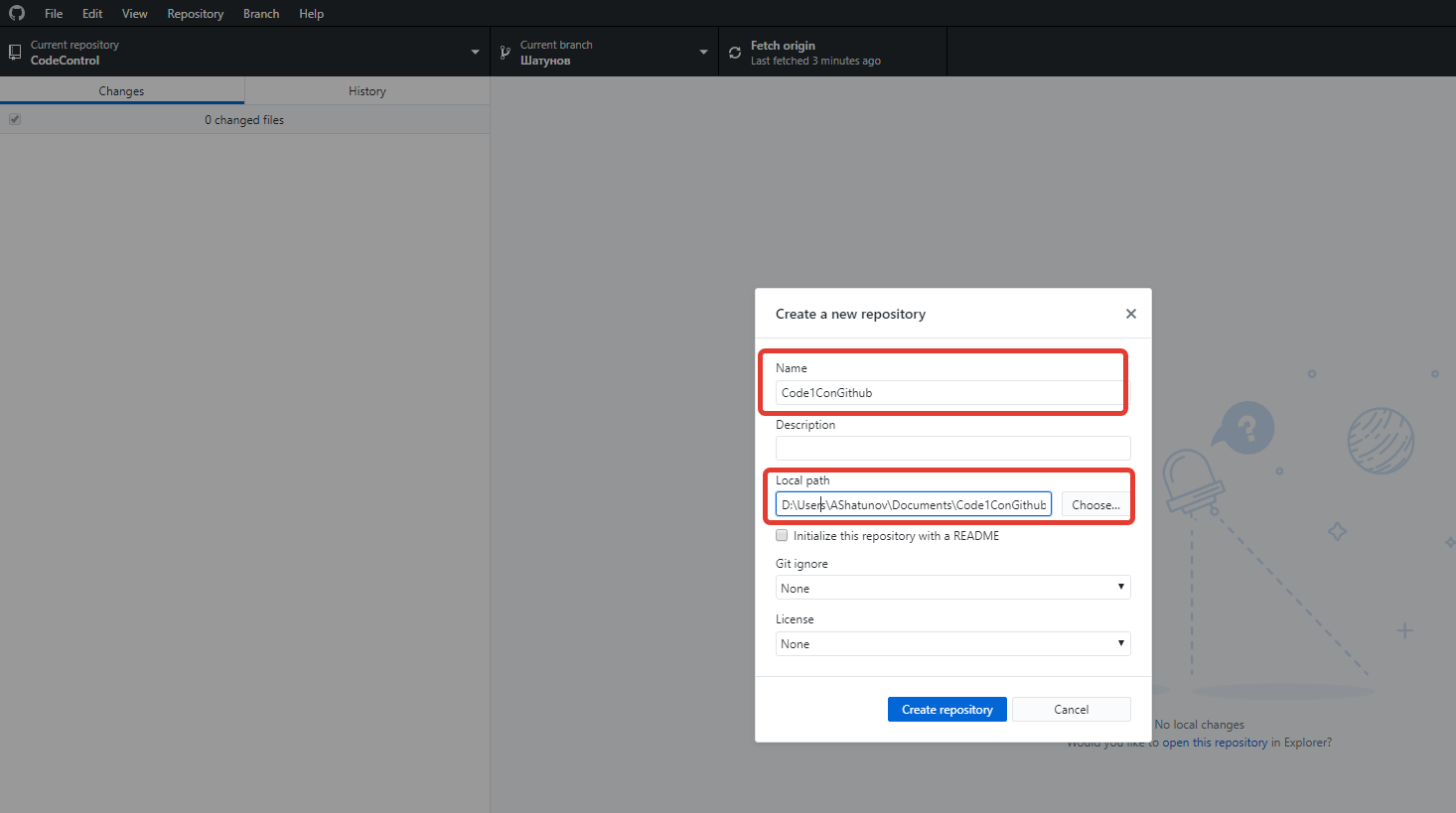
В этом случае мы извлекаем 2 функции гитхаба – систему контроля версий и облачное хранилище.

Ситуация: есть несколько компьютеров с интернетом, пользователь ведет разработку чего-либо.

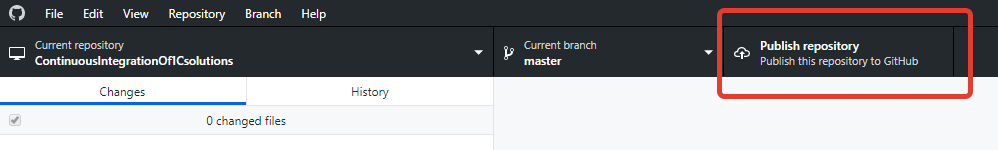
* 1. Создание репозитория

На любом из компьютеров открыть клиент и выполнить **File/New repository**

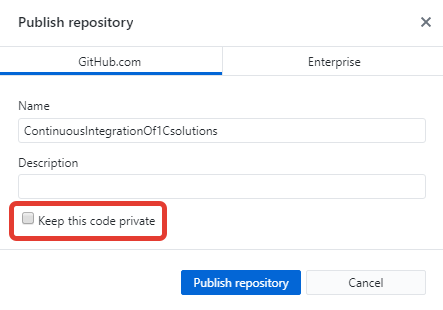
В появившемся окне указать имя проекта и путь к локальному хранилищу.



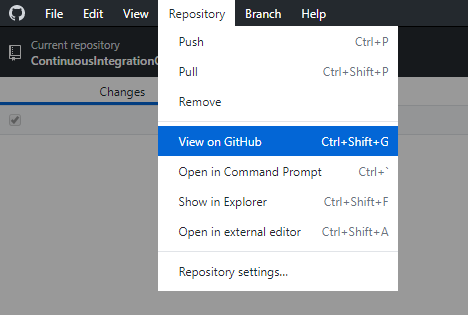
После создания нового проекта нужно отправить его в глобальный репозиторий. Для этого выполнить появившуюся команду **Publish repository**



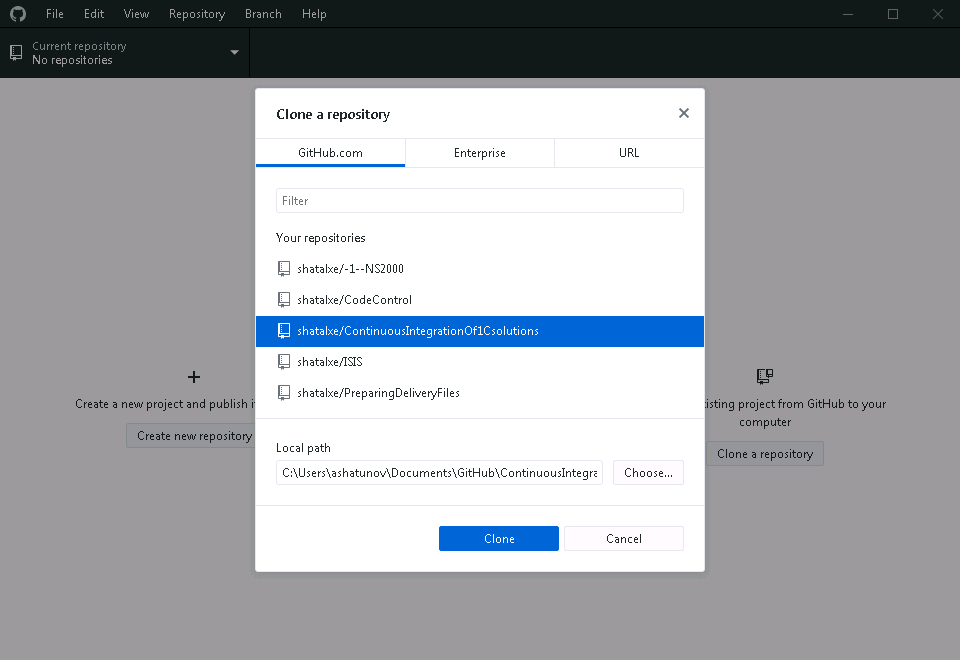
В новом окне убрать флаг **Keep this code private**



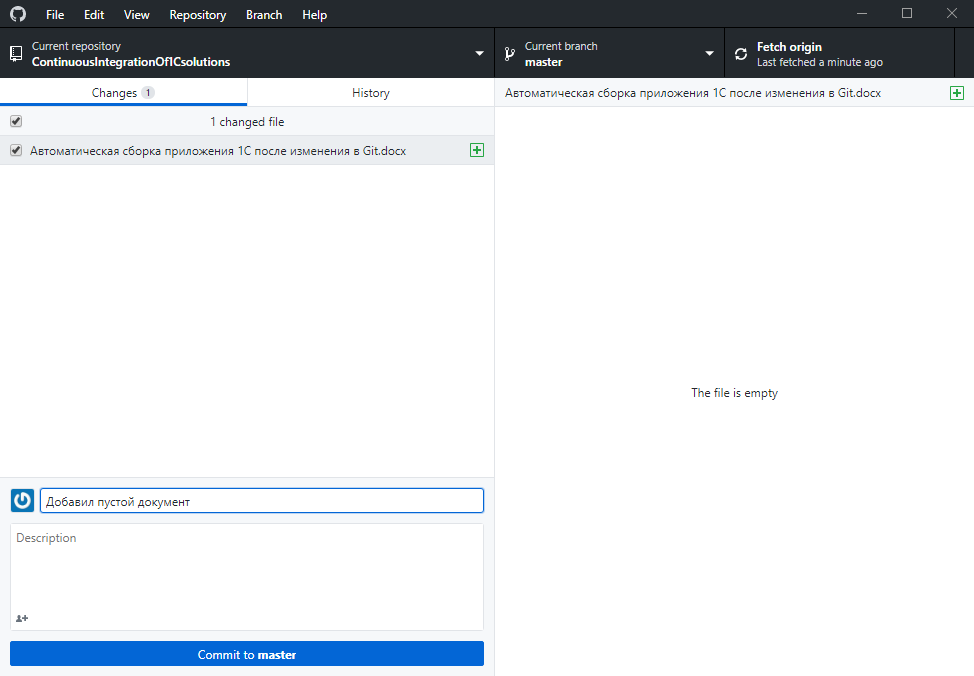
После публикации, за проектом можно следить через Веб-клиент.



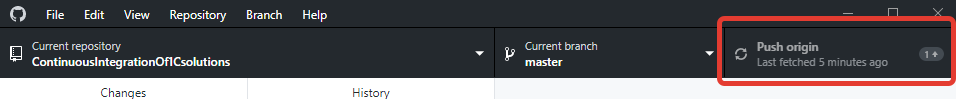
* 1. Подключение нового компьютера к репозиторию
     1. На втором устройстве открыть клиент GitHub
     2. Авторизоваться
     3. Выполнить **File/Clone repository**
     4. Указать нужный проект, нажать **Clone**



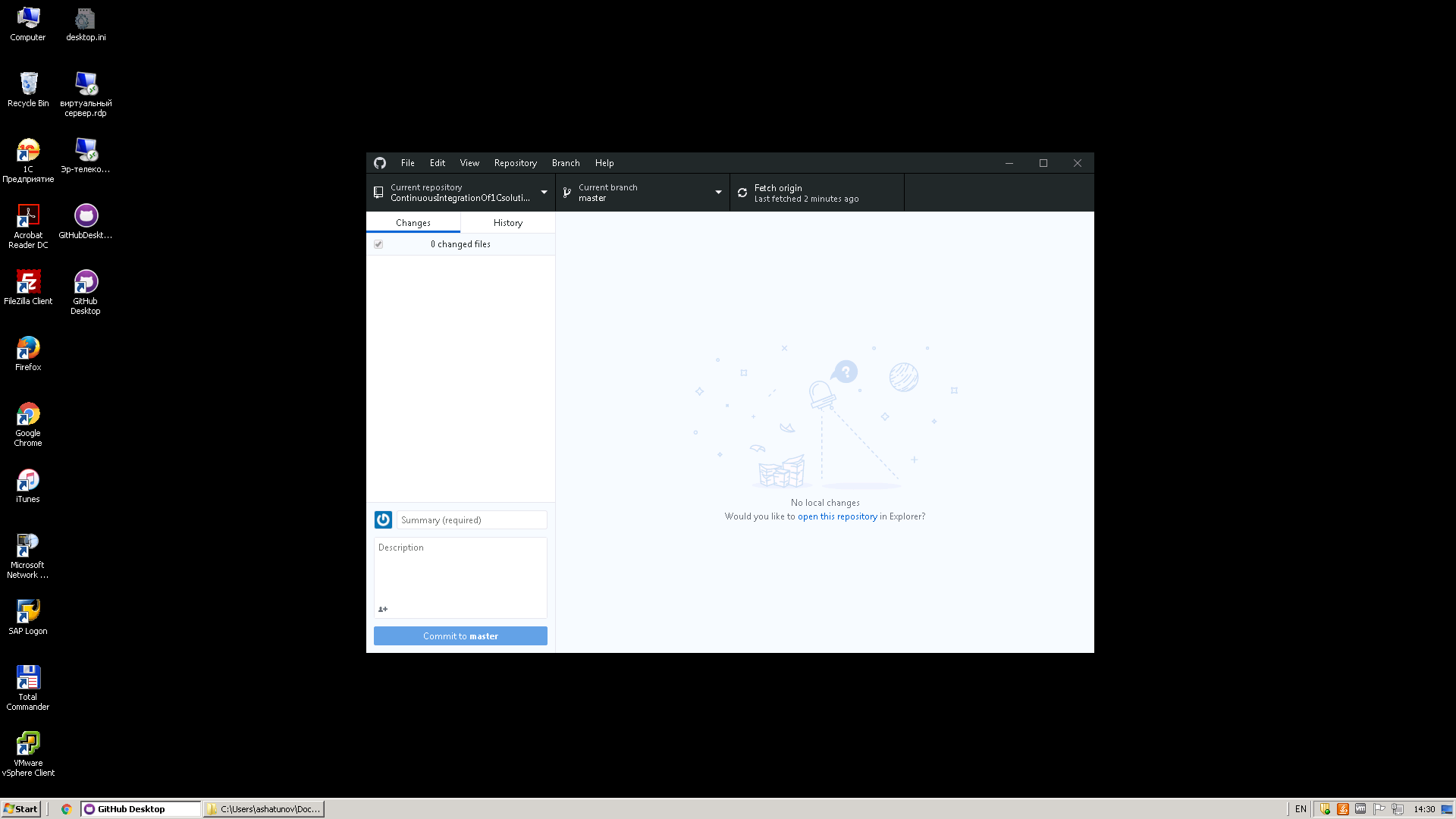
* 1. Выгрузка изменений в глобальный репозиторий
     1. Внести изменения в каталог проекта. Клиент GitHub покажет эти изменения.



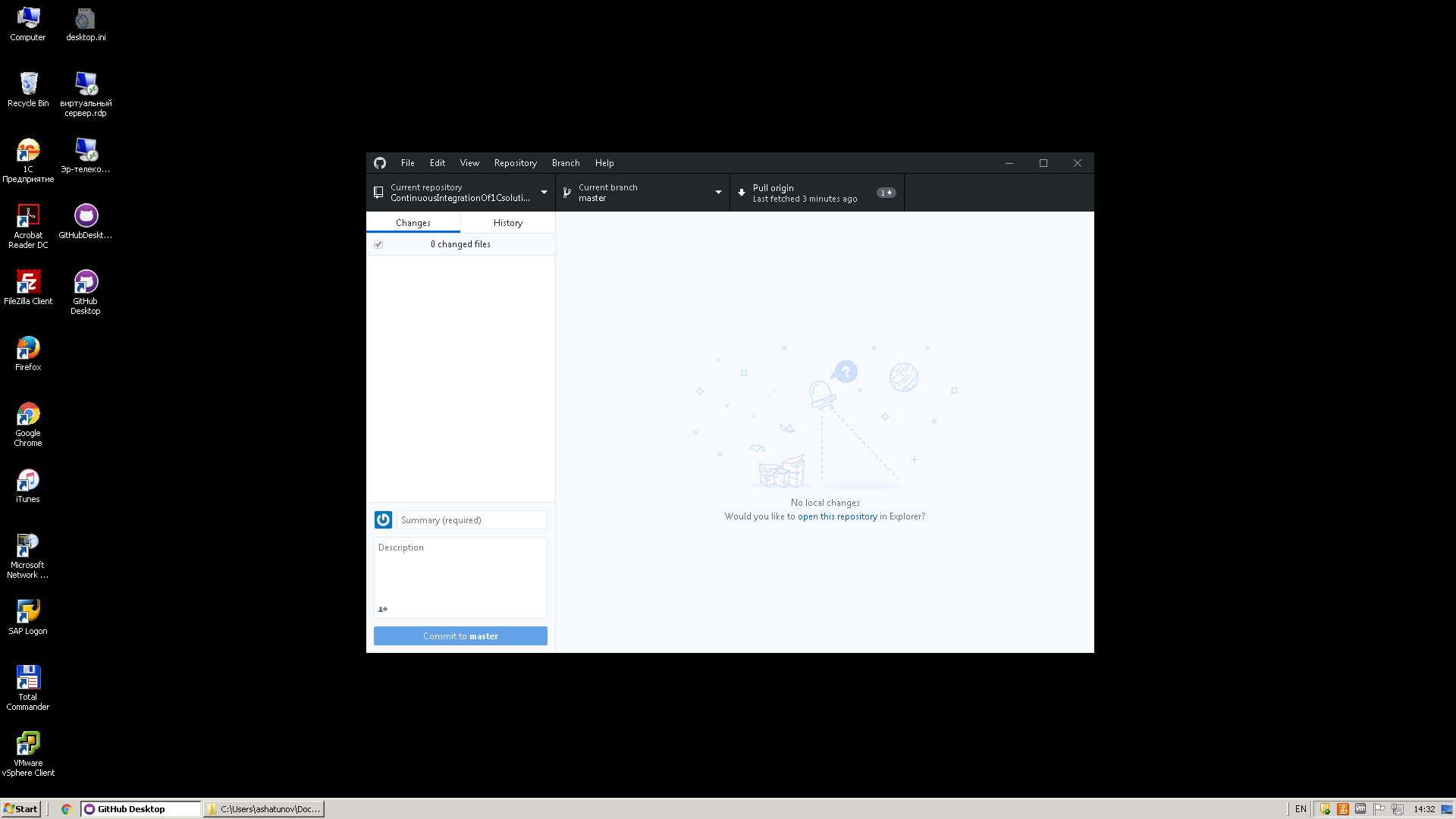
* + 1. В поле summary нужно ввести краткое описание вносимых изменений
    2. Выполнить **commit to master**
    3. Выполнить появившуюся команду **Push origin**



* 1. Загрузка из глобального репозитория
     1. На втором устройстве выполнить **Fetch origin**



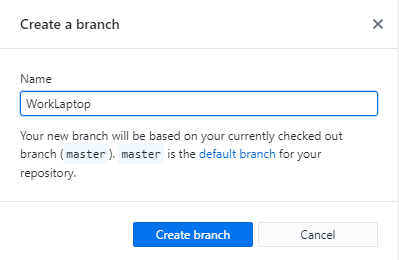
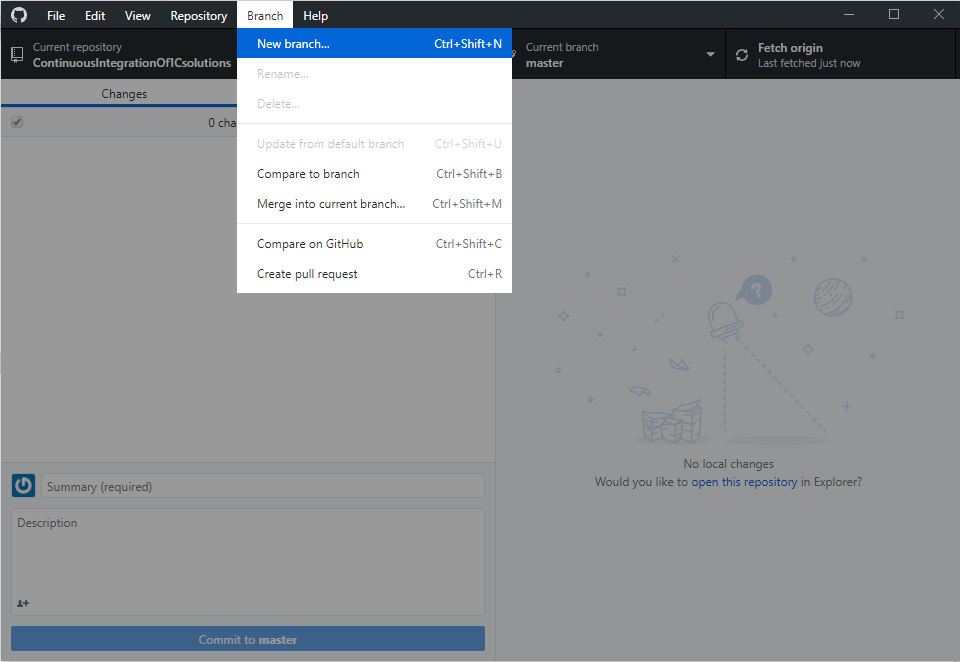
* + 1. Выполнить **Pull origin**



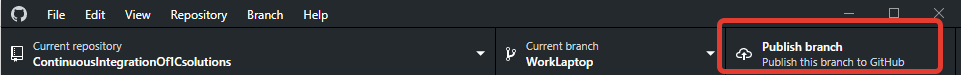
# Однопользовательский, 3 ветки

Вынесем работу на разных устройствах в разные ветки. Время от времени изменения будем отправлять в ветку master и обновляться из неё.

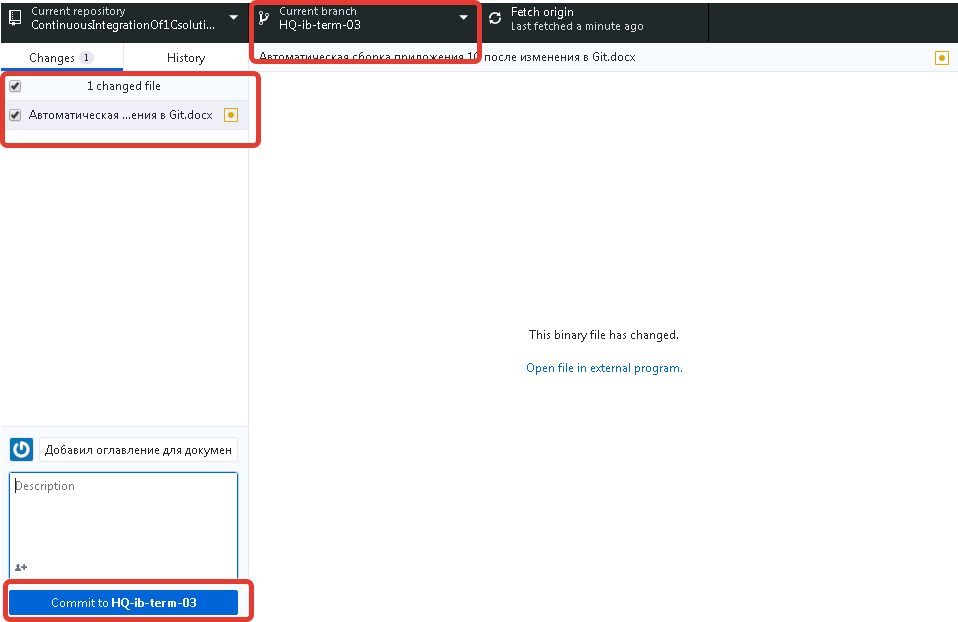
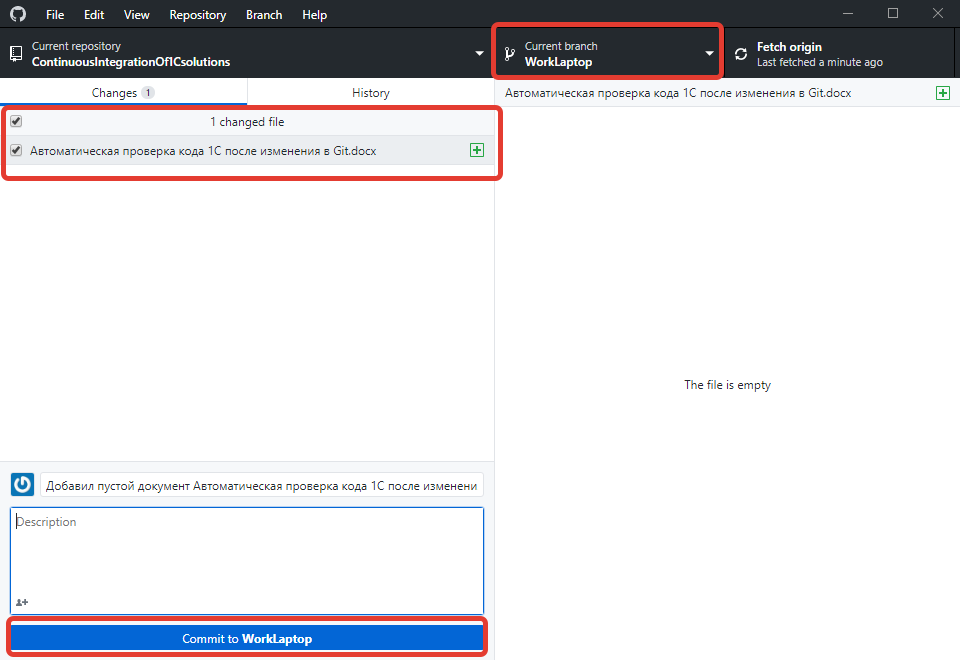
* 1. На обоих устройствах создадим по ветке



* 1. Отправим их в облако командой **Publish branch**



* 1. Добавим по коммиту на обоих устройствах.

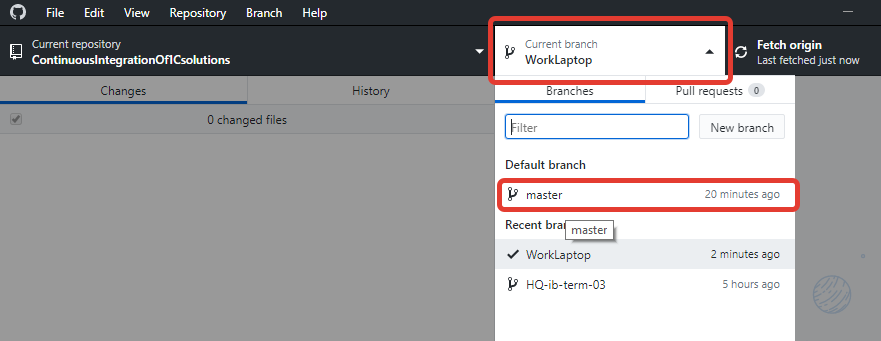


* 1. Отправим изменения в облако

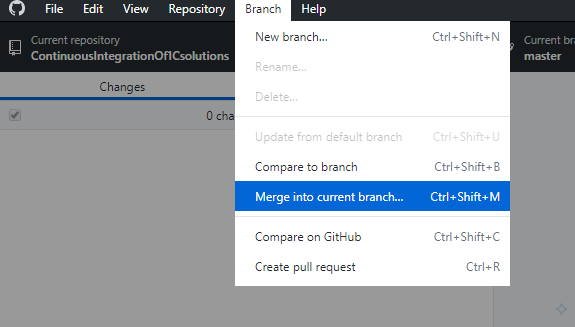
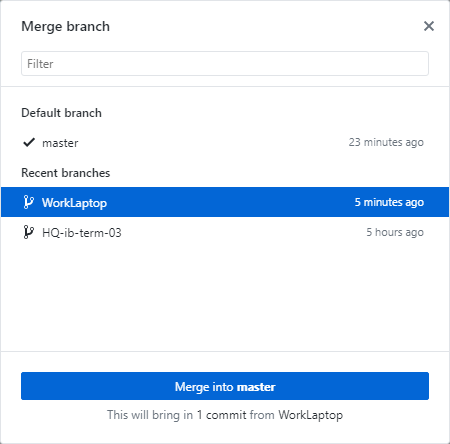


Когда завершаем работу над задачей, и хотим обновить ветку master:

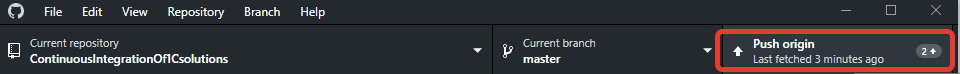
* 1. Переключаемся на ветку Master



* 1. Выполняем слияние с нужной веткой. Нажимаем **Merge into [имя ветки]** (если ветки равны, то команда не будет доступна)

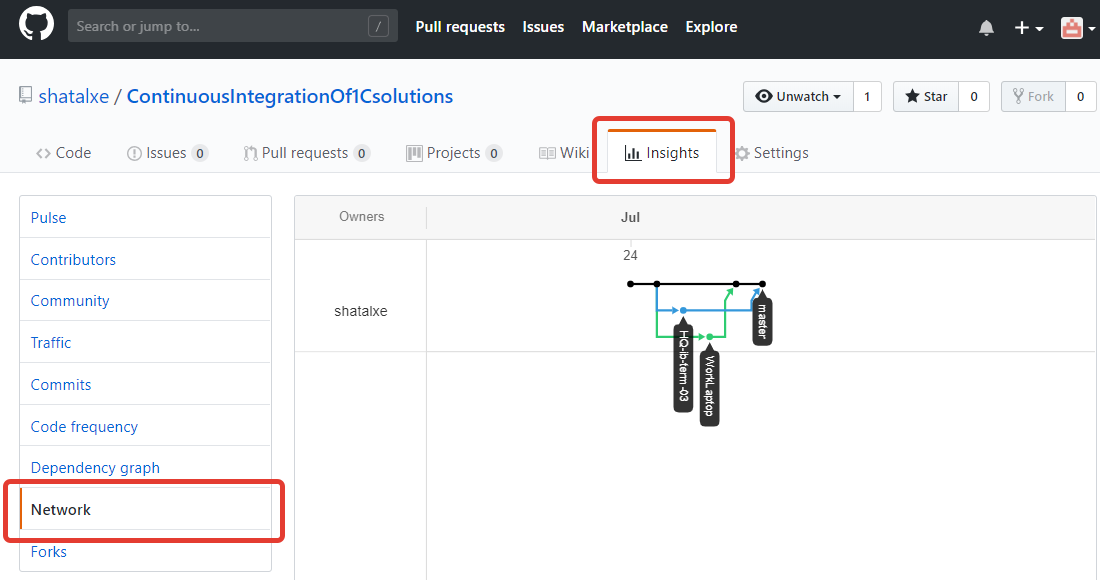
 

* 1. Отправляем изменения в облако



* 1. Повторяем процедуру слияния на других устройствах, предварительно обновив локальный репозиторий.

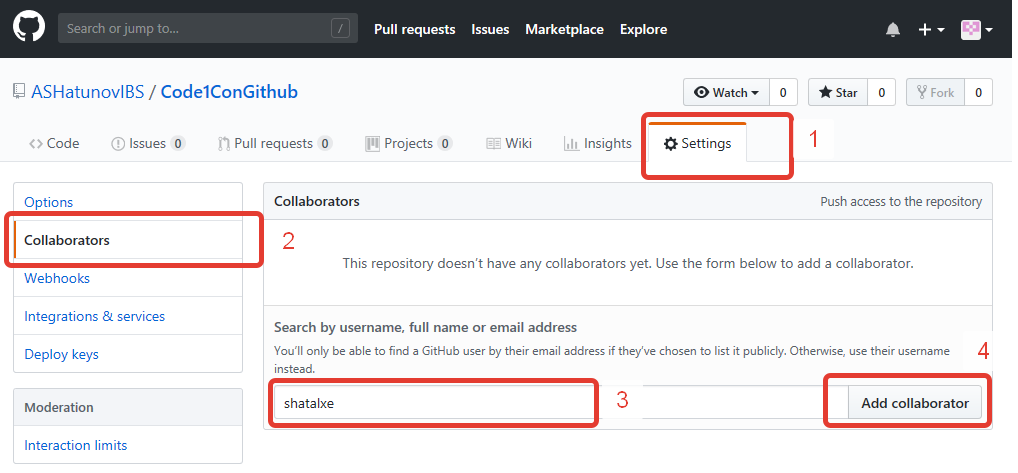
Изменения глобального репозитория в графическом виде можно посмотреть в веб-интерфейсе



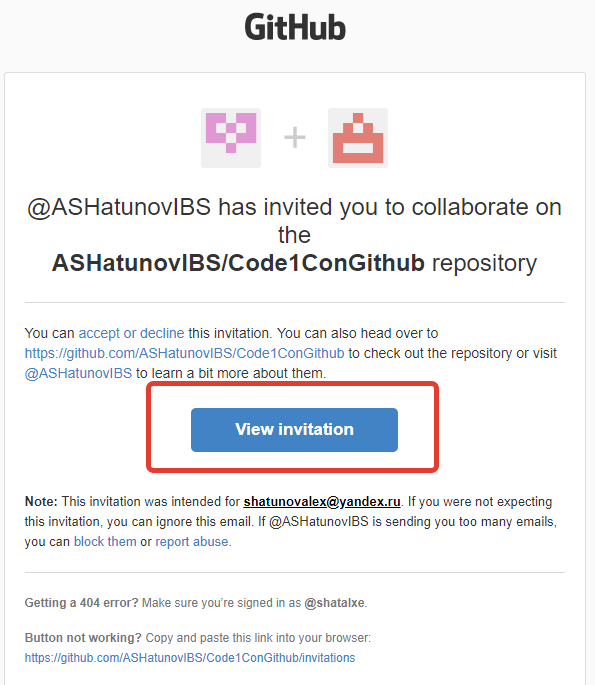
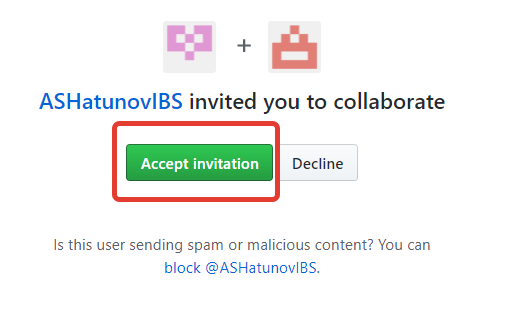
# Многопользовательский совместный

Для организации совместной работы достаточно пригласить соучастников (**Collaborators**) в наш исходный репозиторий (**origin**).

* 1. На странице репозитория ([https://github.com/[Имя](https://github.com/%5bИмя)Пользователя]/[ИмяРепозитория]) перейти на закладку **settings**
  2. Выбрать раздел **Collaborators**
  3. Указать логин участника
  4. Выполнить **Add collaborators**



На почту нового участника придет приглашение. Ему необходимо будет его принять.

Работа с новым участником ничем не отличается от [второго сценария](#_Однопользовательский,_3_ветки): под свои задачи пользователь создает ветки из master и по завершению работы над ними вливает изменения в master.

# Многопользовательский двухуровневый (с архитектором)

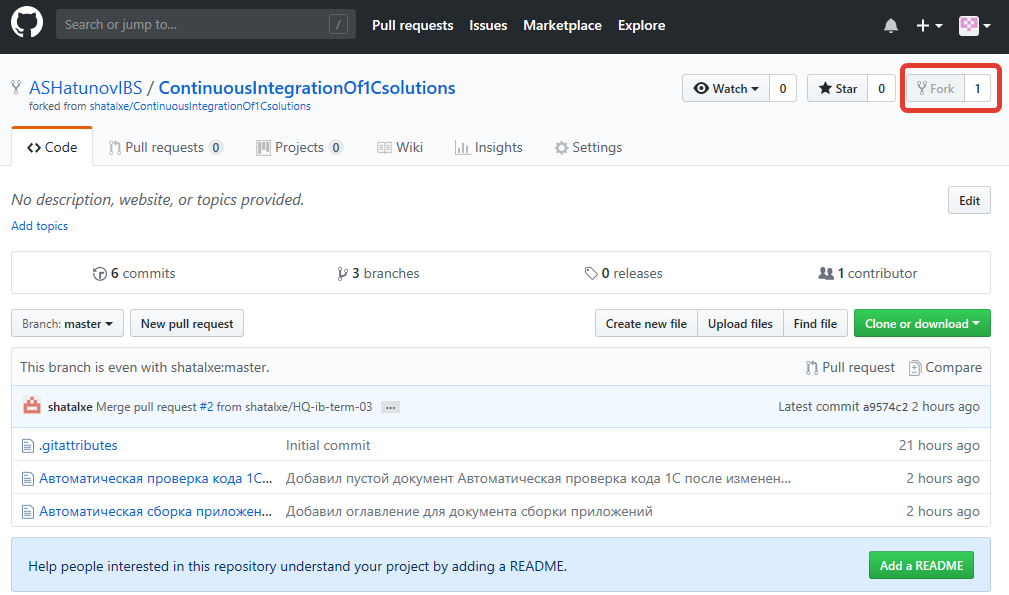
Архитекторами будем считать автора проекта и всех соучастников **(Collaborators)**.

Все остальные пользователи клонируют исходный репозиторий **(origin)** и отправляют в него запросы на изменения **(Pull Request)**.

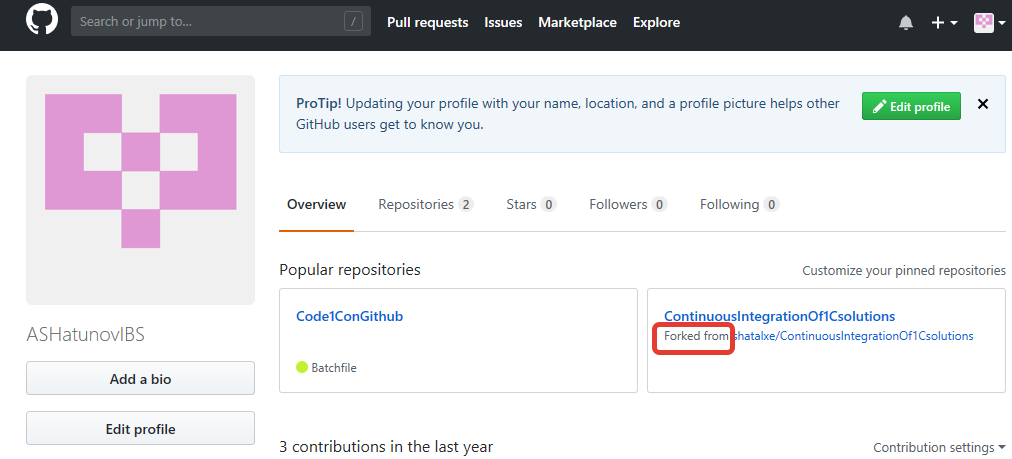
Подключим к нашему проекту нового пользователя. Для этого ему нужно сделать форк проекта и склонировать основной репозиторий.

В Веб-интерфейсе:

* 1. Авторизовываемся под новым пользователем, открываем страницу проекта.
  2. Делаем форк

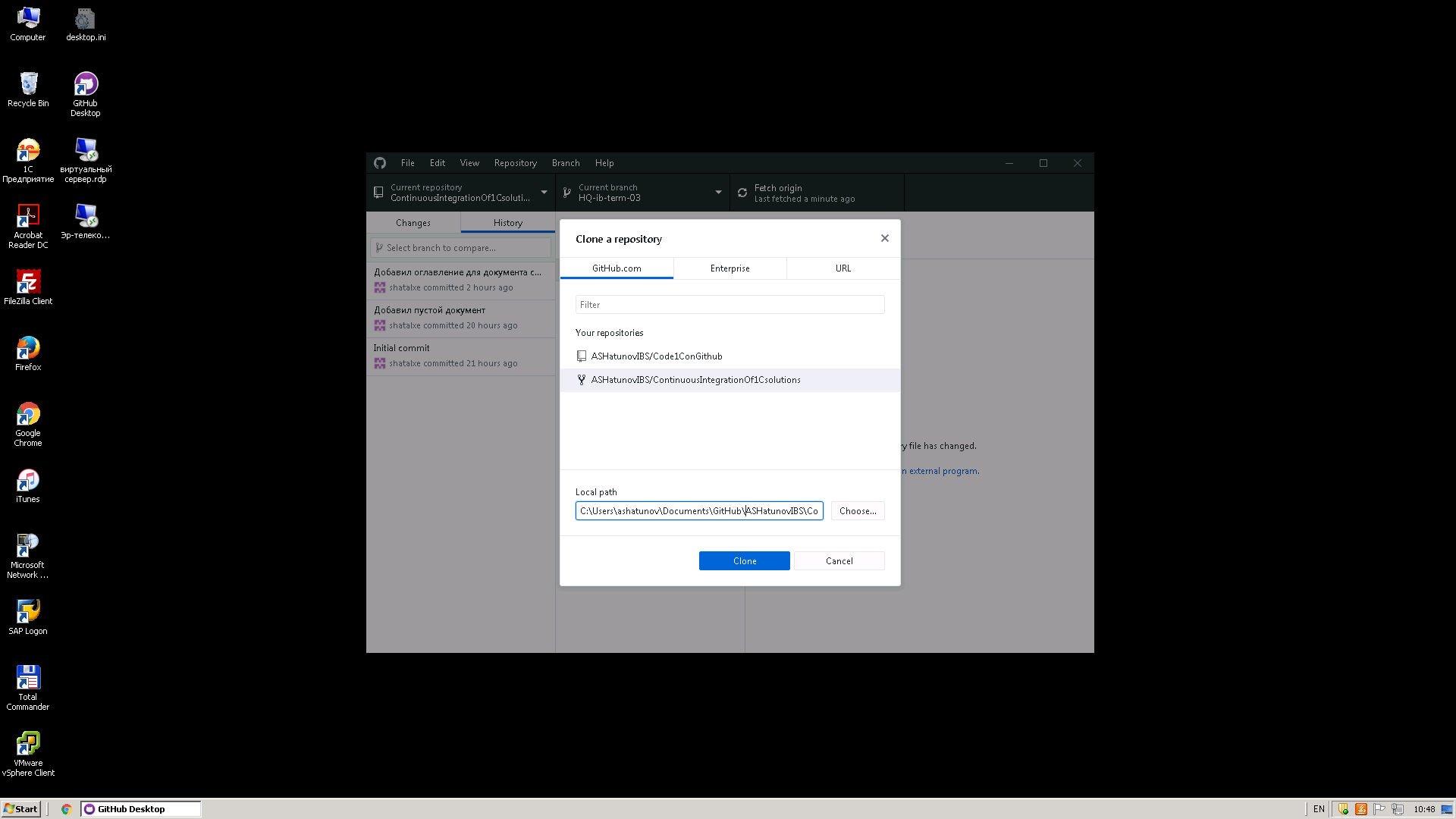


В профиле пользователя появится информация о форке

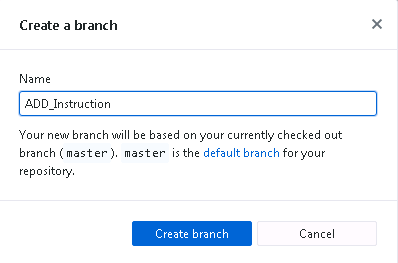


В клиенте windows:

* 1. Авторизовываемся
  2. Клонируем репозиторий



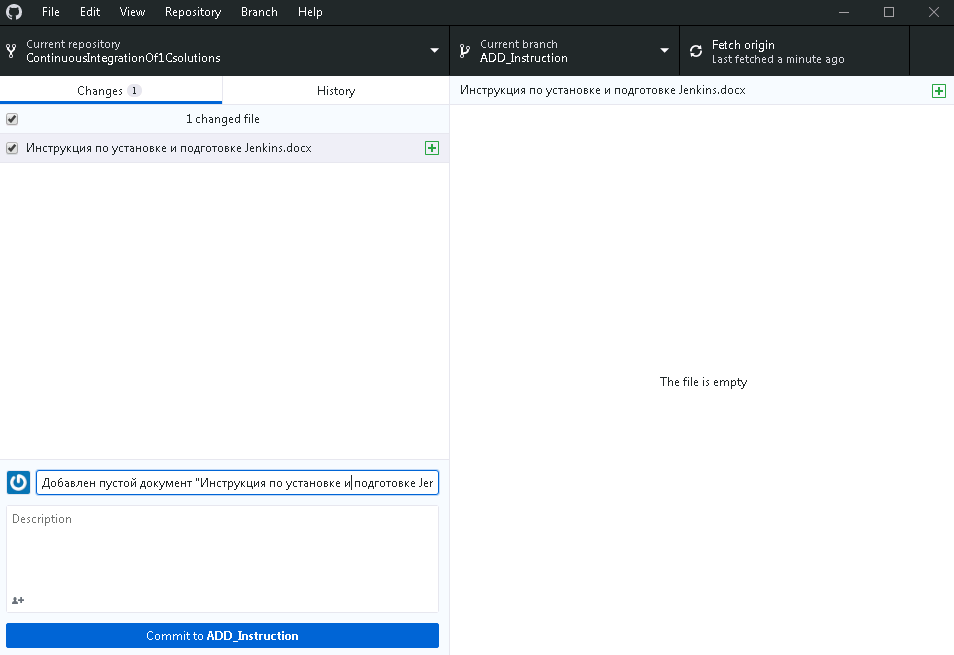
* 1. Добавляем новую ветку



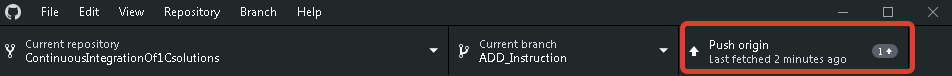
* 1. Отправляем её в облако



* 1. Вносим изменения, делаем коммит

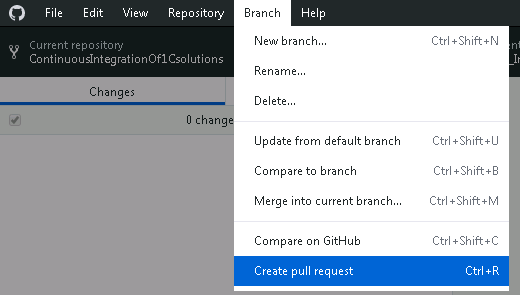


* 1. Пушим в облако



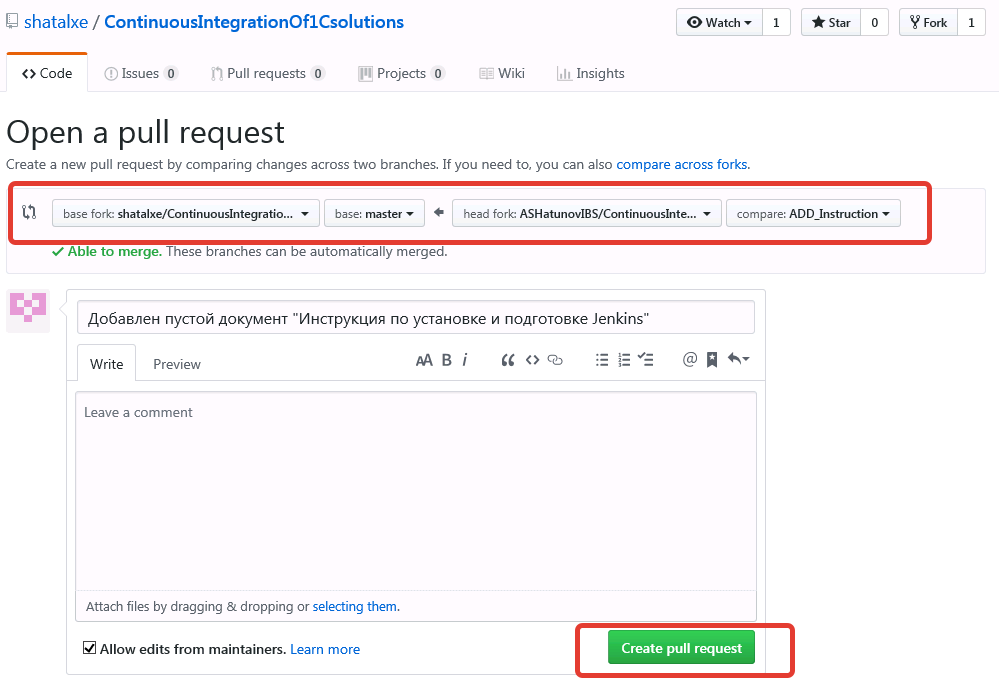
Когда мы закончим работу над задачей, нужно отправить наши изменения в главный репозиторий. Делается это с помощью запросов Pull request

* 1. Открываем страницу создания Pull request



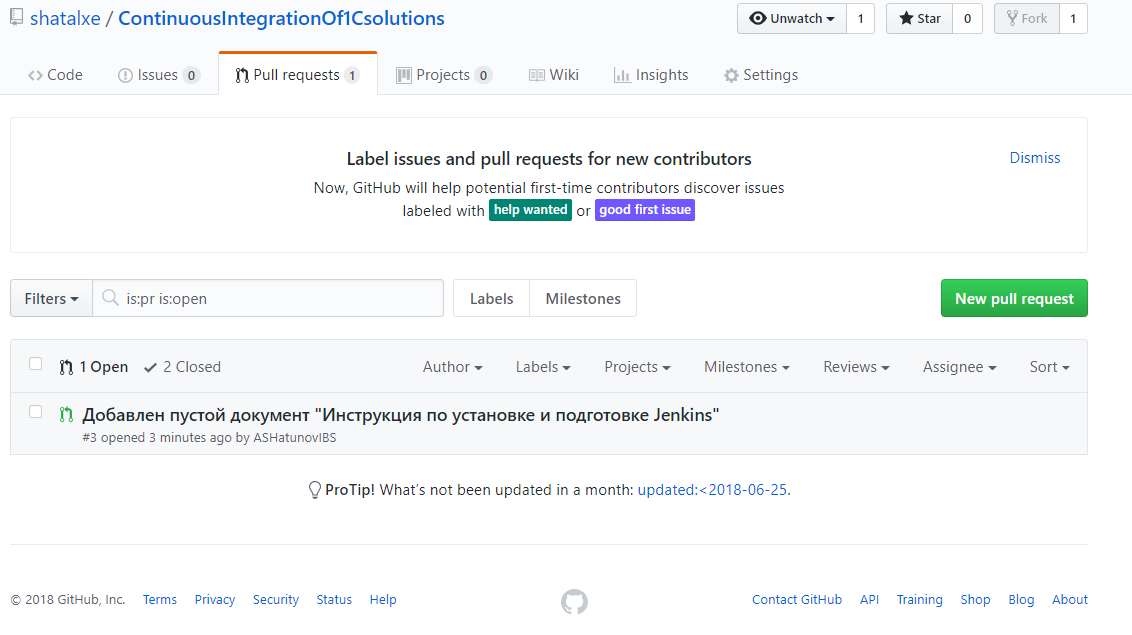
В веб-клиенте:

* 1. Убеждаемся, что слияние идет туда, куда нам надо
  2. Жмем Create pull request

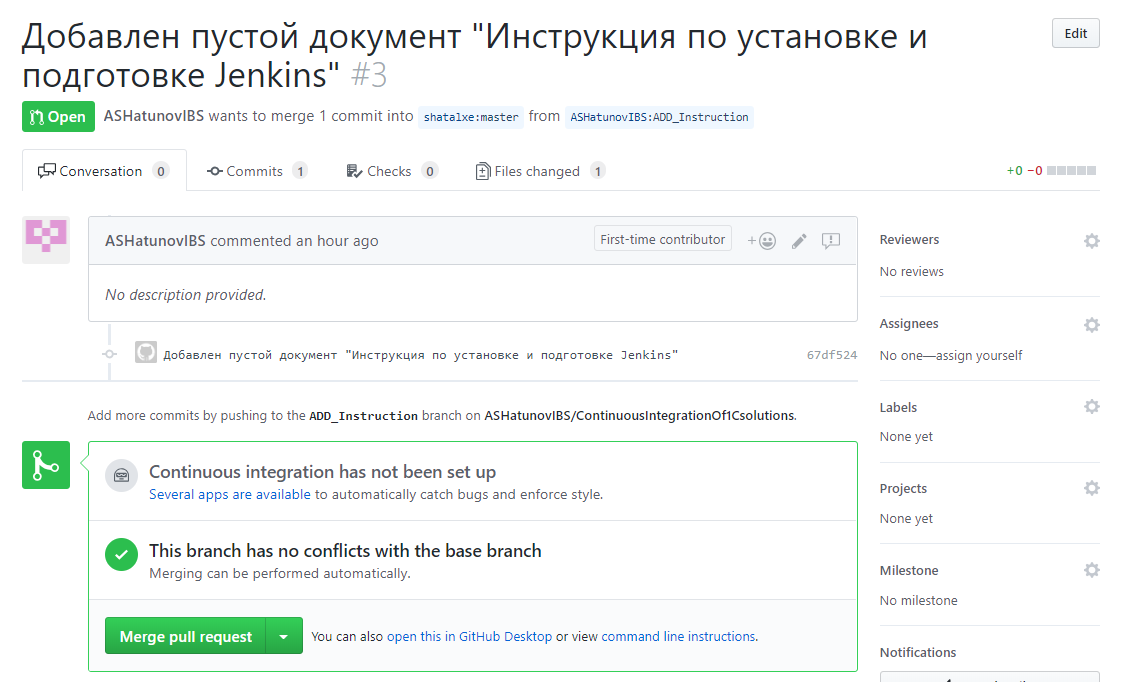


Владелец главного репозитория получит уведомление.

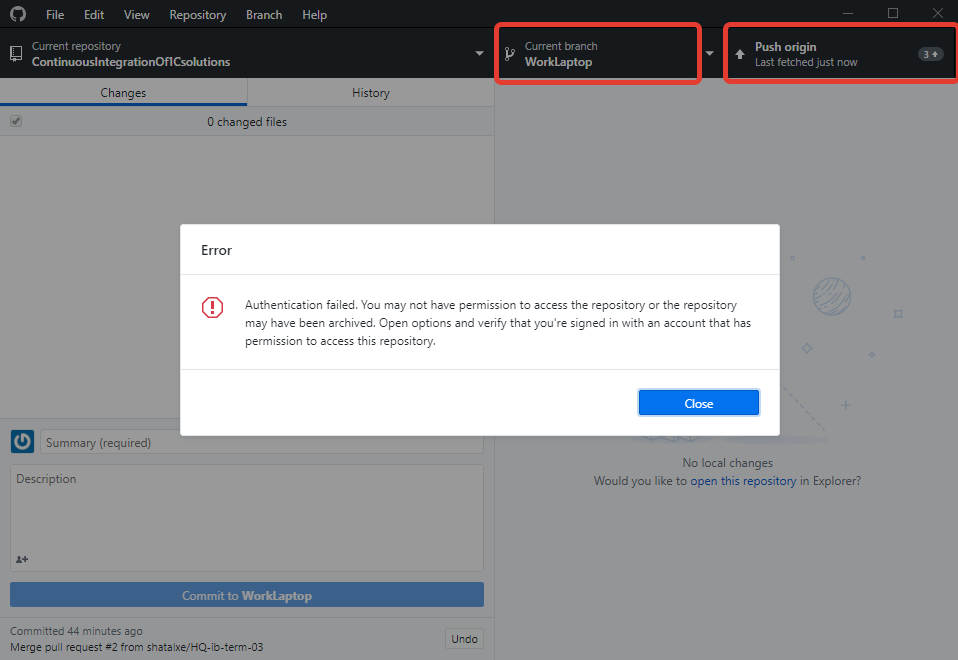
* 1. В главном репозитории (находясь под учетной записью его участника) переходим в раздел Pull requests
  2. Выбираем интересующий Pull request



* 1. Merge pull request



*При попытке обновить репозиторий, к которому у нас нет доступа, получим предупреждение:*



# Многопользовательский, 2 общих ветки (продакшен и препродакшен)

*Очень похожа на классическую, только в ней помимо мастер ветки есть еще девелоперская ветка, которая деплоится на тестовый сервер. Такую ветку обычно называют dev. Схема работы при этом такая. Программист перед выполнением новой задачи заводит для нее ветку от последнего рабочего состояния в мастер ветке. Когда он заканчивает работу над задачей, то мержит ветку задачи в dev ветку самостоятельно. После этого, совместными усилиями задача тестируется на тестовом сервере вместе с остальными задачами. Если есть ошибки, то задачу дорабатывают в той же ветке и повторно мержат с dev веткой. Когда тестирование задачи заканчивается, то ВЕТКУ ЗАДАЧИ мержат с мастер веткой. Важно заметить, что в этой схеме работы с мастер веткой нужно мержить ветку задачи, а не dev ветку. Ведь в dev ветке будут содержаться изменения, сделанные не только в этой задаче, но и в других и не все эти изменения могут оказаться рабочими. Мастер ветка и dev ветка со временем будут расходиться, поэтому при такой схеме работы периодически заводят новую dev ветку от последнего рабочего состояния мастер ветки. Недостатком этого подхода является избыточность, по сравнению с классической схемой. Такую схему работы с ветками часто используют если в проекте нет автоматизированных тестов и все тестирование происходит вручную на сервере разработки.*

<https://habr.com/post/342116/>