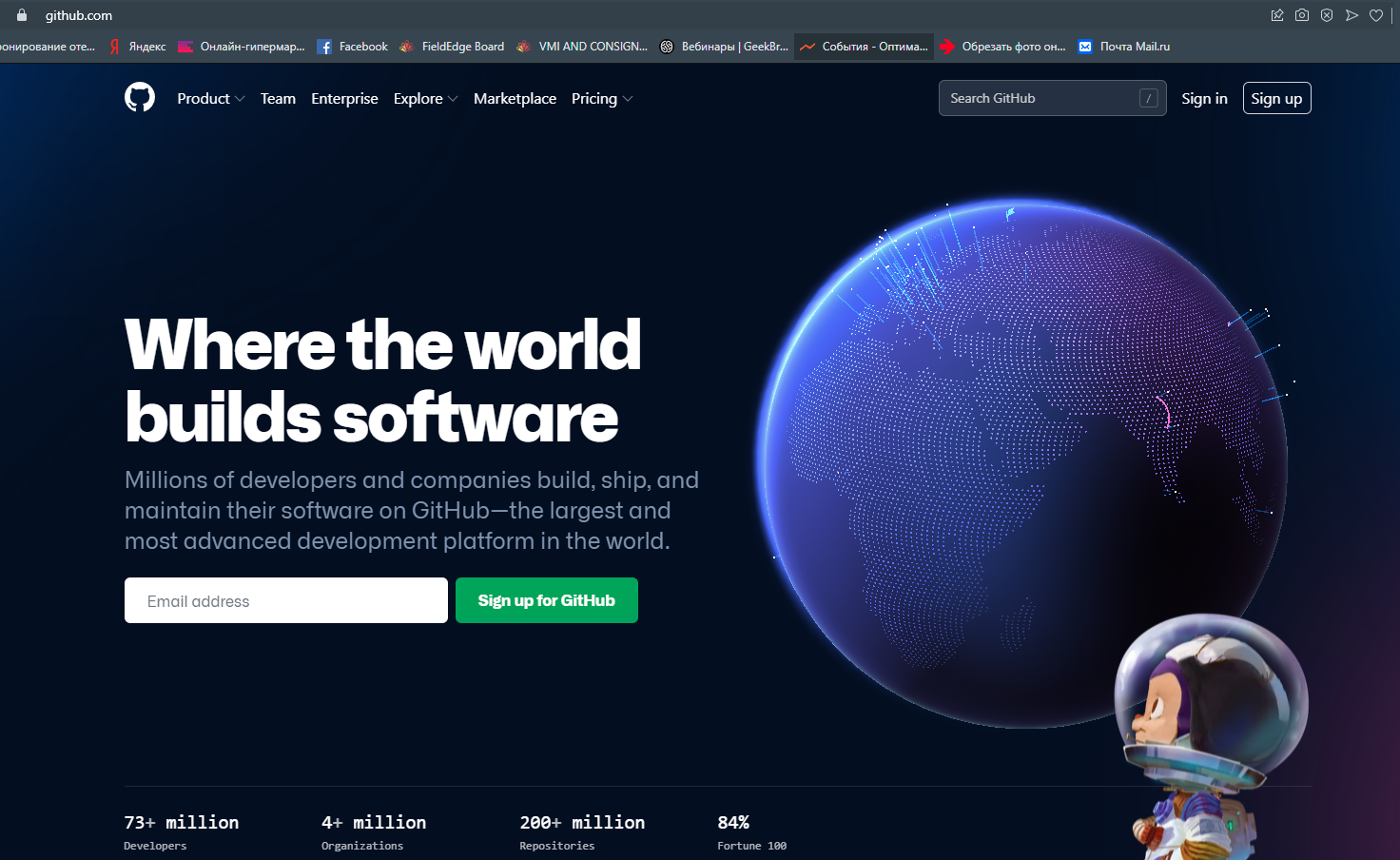
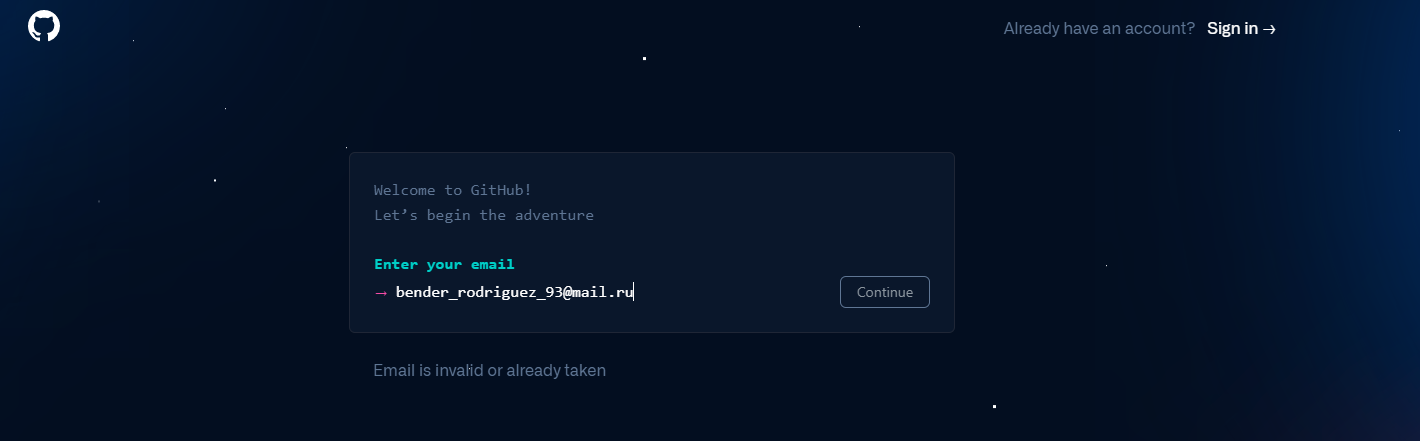
**Задание №1 Системы контроля версий (Git)**

Заведите бесплатную учетную запись на сайте https://github.com/ .

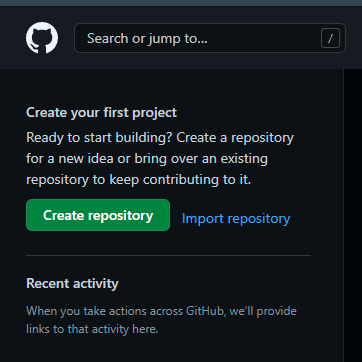
Регистрация на сайте заключается в выборе имени пользователя и пароля.

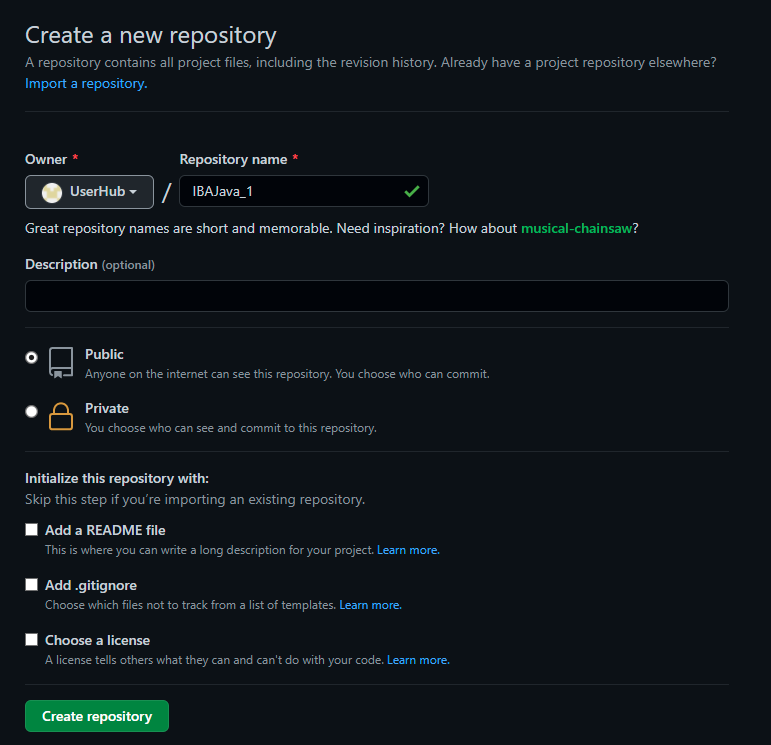
Заполните информацию о себе на странице вашего профиля.





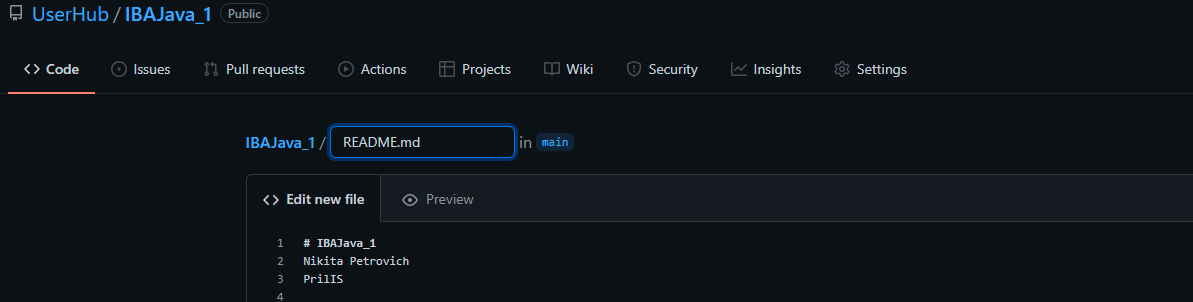
Создайте репозиторий для вашего первого проекта (можно загрузить Старый проект). Для этого нужно перейти по https://github.com/new, выбрать название репозитория (произвольным образом), описание и режим доступа "Public".

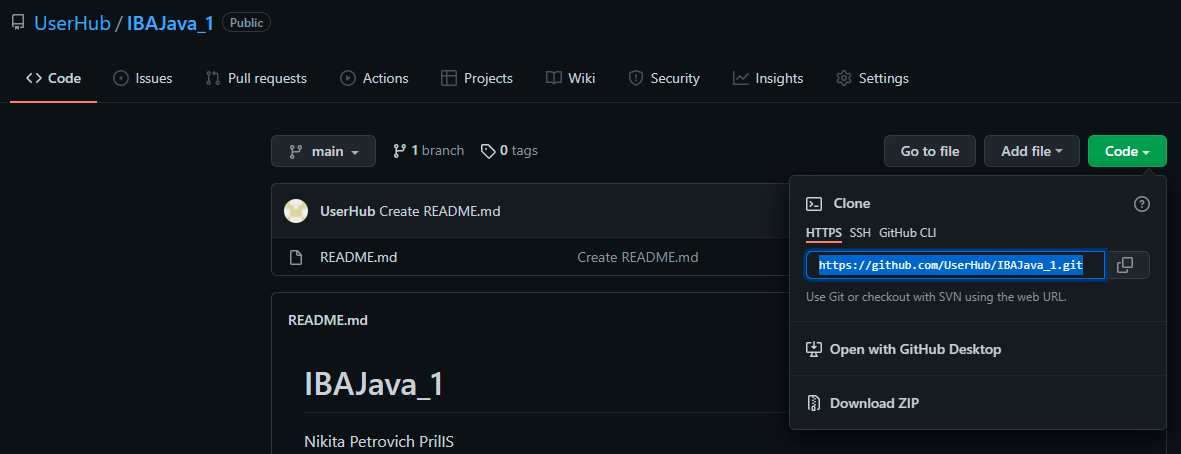




В дальнейшем можно будет использовать разделы подсказок: Воспользуйтесь разделом "…or create a new repository on the command line" из подсказки, чтобы инициализировать репозиторий и создать в нем файл README.md и.т.д

Мы создадим пустой репозиторий с файлом README.md. Для этого в шапке нужно нажать соответствующую ссылку. Модифицируйте файл README.md так, чтобы он содержал ваше имя, номер группы. Добавляем его в проект (зеленая кнопка внизу).



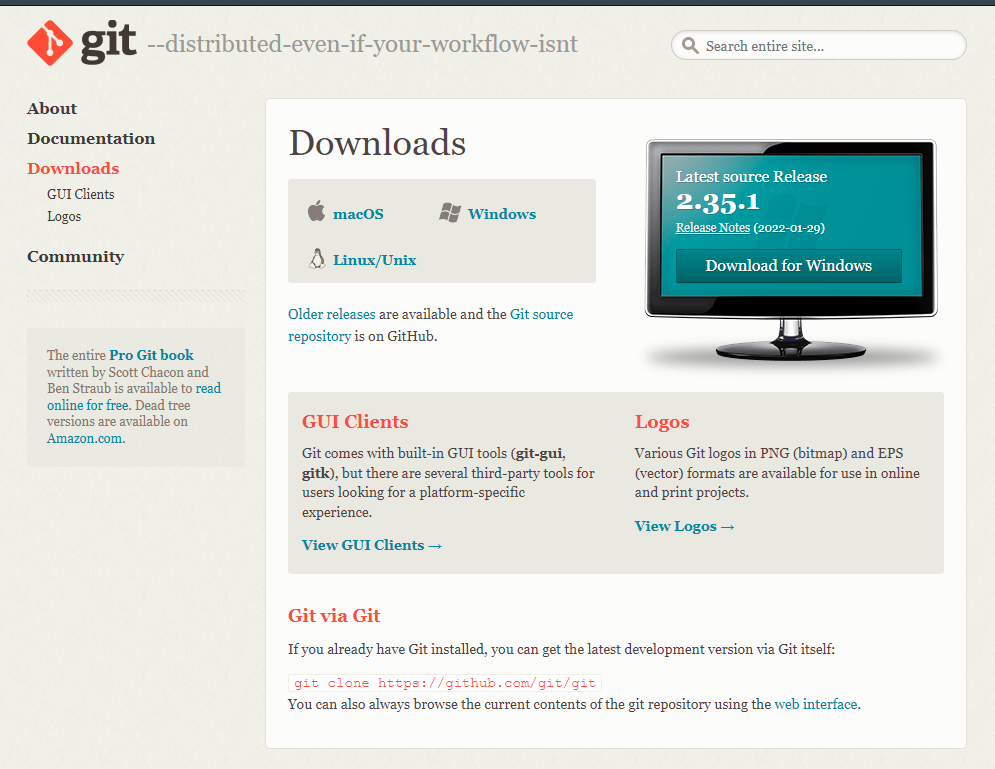


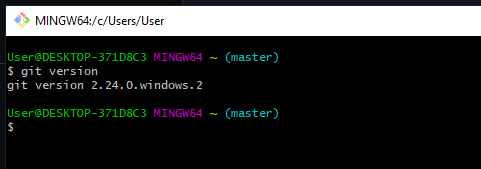
**Базовый уровень Работа с клиентом GitBush**

Скачайте http://git-scm.com/download/ и установите консольного (обязательно!!!) клинета (gitbush). Проверьте версию.

Для дополнительной информации используйте книгу:

<https://git-scm.com/book/ru/v2>





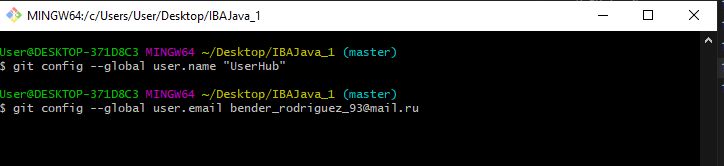
**Выполните настройку конфигурации**

Мы можем выбрать пустую папку или существующий проект. Нажать внутри правой кнопкой мыши и выбрать Git Bash Here. После чего откроется консоль для управления версиями в текущей директории:

Выполните конфигурацию (git config) (настройте имя пользователя и т.п.).

$ git config --global user.name "XXX"

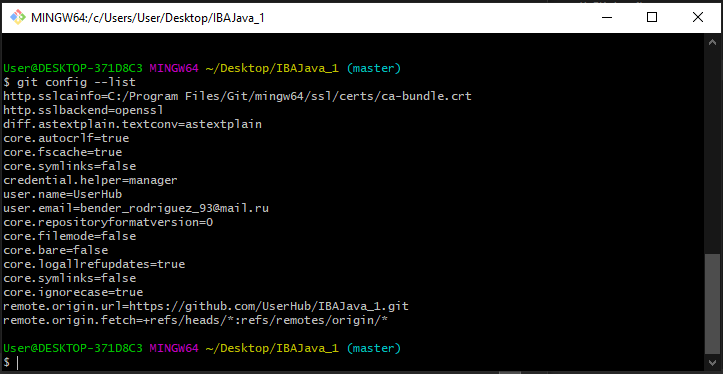
$ git config --global user.email [XXX@XXX.com](mailto:XXX@XXX.com)



Передача параметра --global позволяет сделать эти настройки всего один раз.

Если для конкретного проекта требуется указать другое имя или адрес электронной почты, войдите в папку с проектом и выполните эту команду без параметра --global.

Проверить выбранные настройки позволяет команда git config --list, выводящая список всех обнаруженных в текущий момент параметров:



## Создайте репозиторий

Есть два подхода к созданию Git-проекта. Можно взять существующий проект или папку и импортировать в Git. А можно клонировать уже существующий репозиторий с другого сервера.

## Инициализируйте репозиторий.

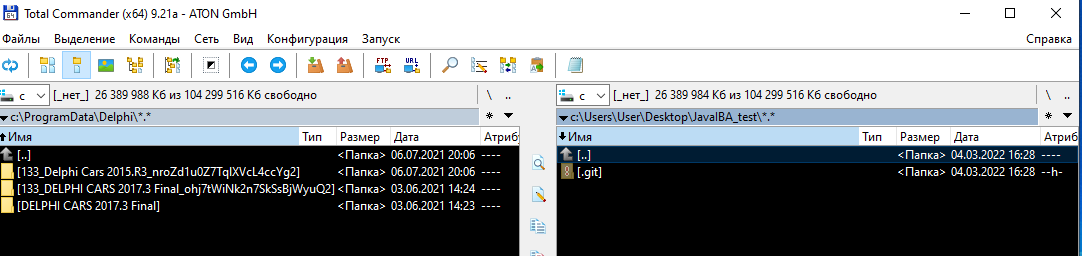
## Чтобы начать слежение за существующим проектом, перейдите в папку этого проекта и введите команду (выполнять можете, например, в нашем старом проекте. У меня это будет проект с gson -ом):

## $ git init

## Эта команда создаёт в текущей директории новую скрытую поддиректорию с именем .git, содержащую все необходимые файлы репозитория — основу Git- репозитория. Если вы сделали ошибку – можно просто ее удалить.

## 

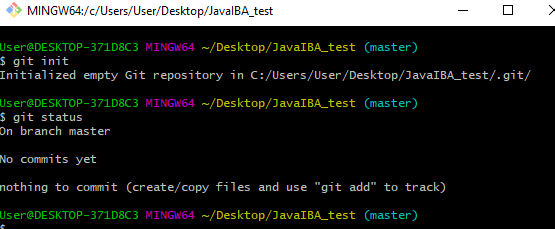
В каталоге проекта появилась папка:



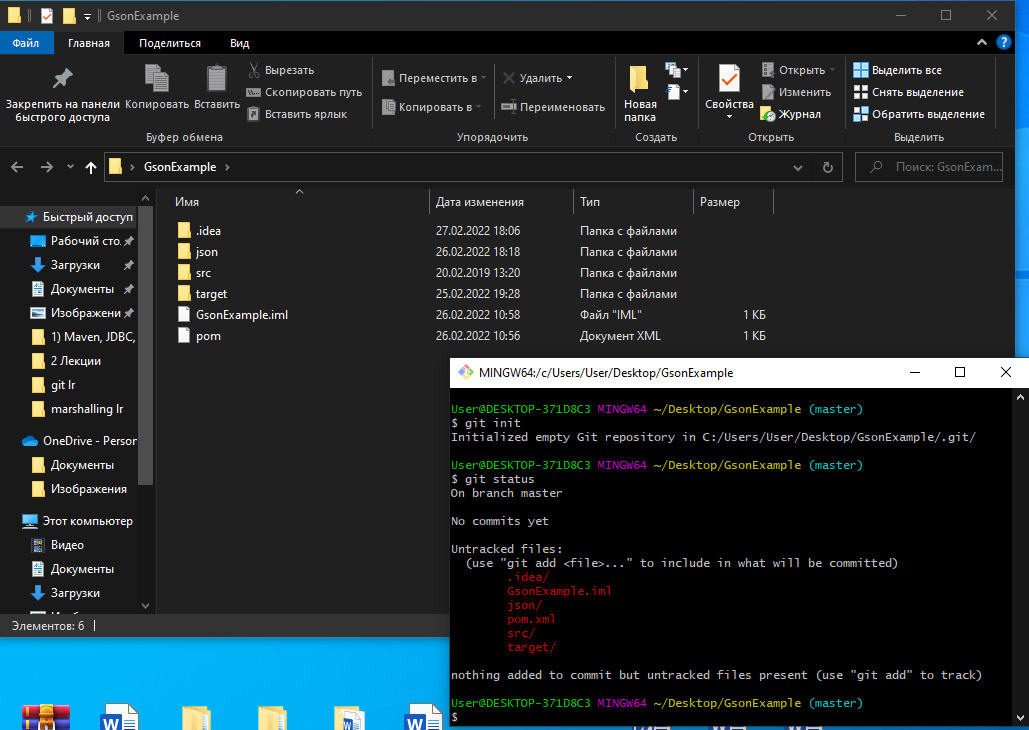
Выполните:

$ git status

Если использовать пустой проект: Основным инструментом определения состояния файлов является команда git status. Показывает, что изменения не зафиксированы:



При создании в существующем проекте (обратите внимание на файлы красным цветом):



Статус «Untracked files», по сути, означает, что Git видит файл, отсутствующий в предыдущем снимке состояния (коммите); Git не станет добавлять его в ваши коммиты, пока вы его явно об этом не попросите.

Выполните первую фиксацию изменений. Чтобы начать слежение за новым файлом, воспользуйтесь командой git add. добавляющую файлы, за которыми вы хотите следить.

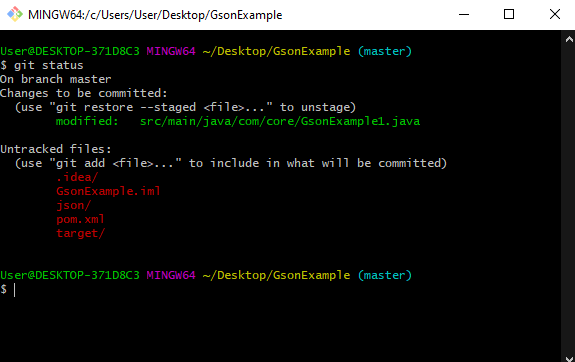
Теперь команда status покажет, что этот файл является отслеживаемым и проиндексированным.

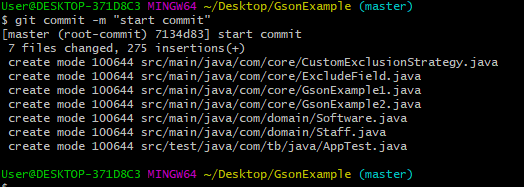
Затем команда git commit.

$ git add \*.java

$ git commit -m 'first commit'

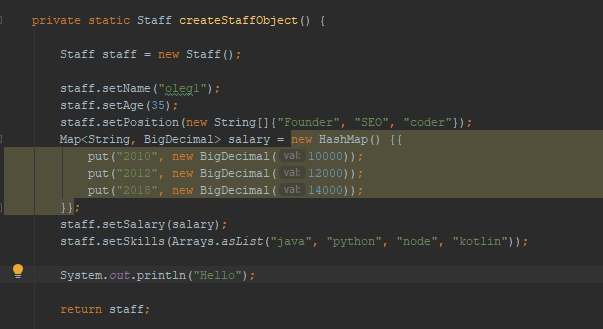
Обратите внимание, что после добавления add все файлы из src пропали как не отслеживаемые (modified у вас не будет):



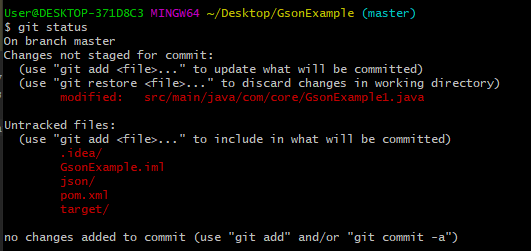


Сообщение фиксации можно задать и в команде commit, поставив перед ней флаг -m (как на рисунке выше). Вы создали первую версию изменений. Обратите внимание, что версия предоставила вам ряд данных о себе: зафиксированная вами ветка (master), контрольная сумма SHA-1 версии, количество подвергшихся изменениям файлов и статистика добавленных и удаленных строк

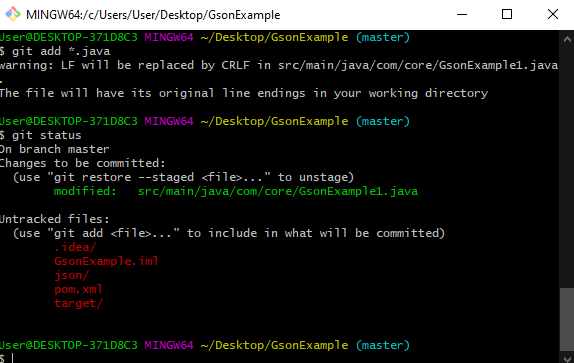
Давайте внесем изменения в файл, находящийся под наблюдением. Если отредактировать ранее отслеживаемый файл GsonExample.java



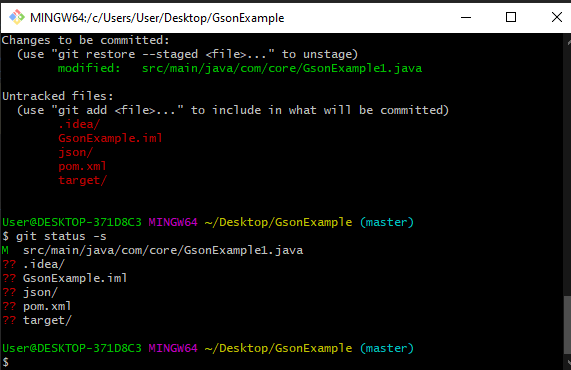
и воспользоваться командой git status, результат будет примерно таким:



Оказывается, Git индексирует файл в том состоянии, в котором он пребывал на момент выполнения команды git add. Если сейчас зафиксировать изменения, в коммит войдет версия файла GsonExample1.java, появившаяся после последнего запуска команды git add, а не версия, находившаяся в рабочей папке при запуске команды git commit. Редактирование файла после выполнения команды git add требует повторного запуска этой команды для индексирования самой последней версии файла:



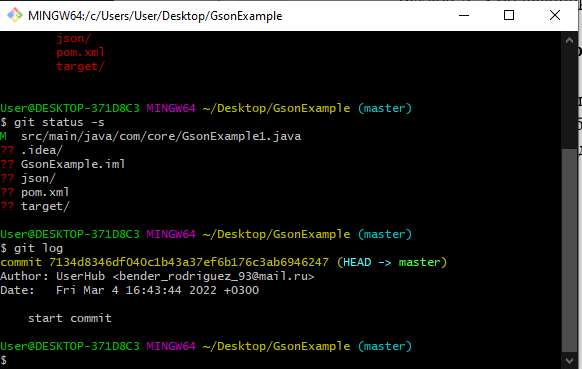
В Git существует флаг, позволяющий получить сведения в более компактной форме. Запустив команду git status -s или git status --short, вы получите упрощенный вариант вывода.



Рядом с именами новых не отслеживаемых файлов стоит знак ??, новые файлы, добавленные в область предварительной подготовки, помечены буквой A, а модифицированные файлы — буквой М.

**Просмотр истории версий**

После сохранения нескольких версий файлов вы, скорее всего, захотите взглянуть на то, что было сделано ранее. Базовым инструментом в данном случае является команда git log.

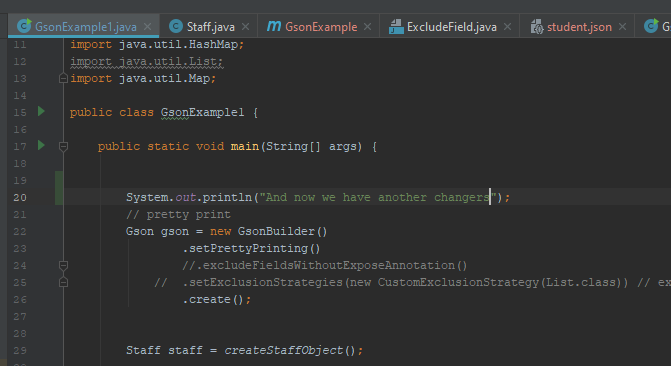


Команда git log выводит в обратном хронологическом порядке список сохраненных в данный репозиторий версий. То есть первыми показываются самые свежие коммиты. Как видите, рядом с каждым коммитом указывается его контрольная сумма SHA-1, имя и электронная почта автора, дата создания и сообщение о фиксации.

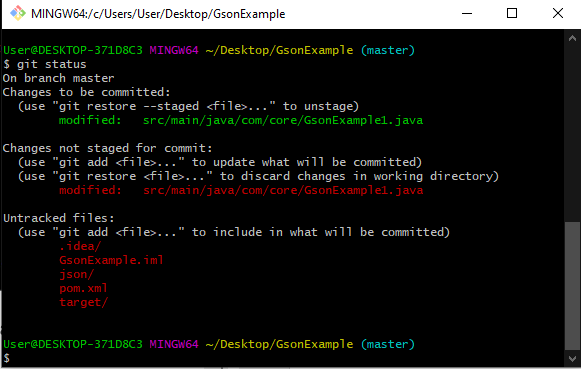
## Отмена внесенных в файл изменений

Если вы внезапно поняли, что не хотите сохранять внесенные в файл изменения. Можно отменить их и вернуть файл в состояние, в котором он находился до последней фиксации.

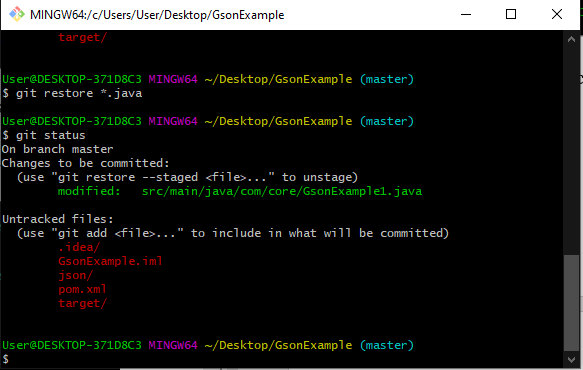
Например, мы внесли изменения.



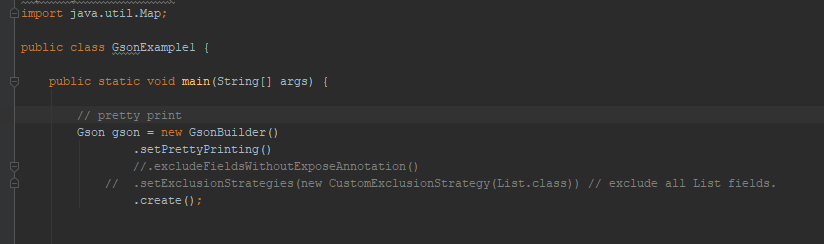
Проверим изменения через status:



Если прочитать сообщения, то увидите, что здесь указано, как отбросить сделанные изменения.



После отмены проверьте результат:



Как видите, откат изменений выполнен. Важно понимать, что git restore [file] — опасная команда. Любые изменения соответствующего файла пропадают — вы просто копируете поверх него другой файл. Ни в коем случае не используйте эту команду, если вы не убеждены, что файл вам не нужен.

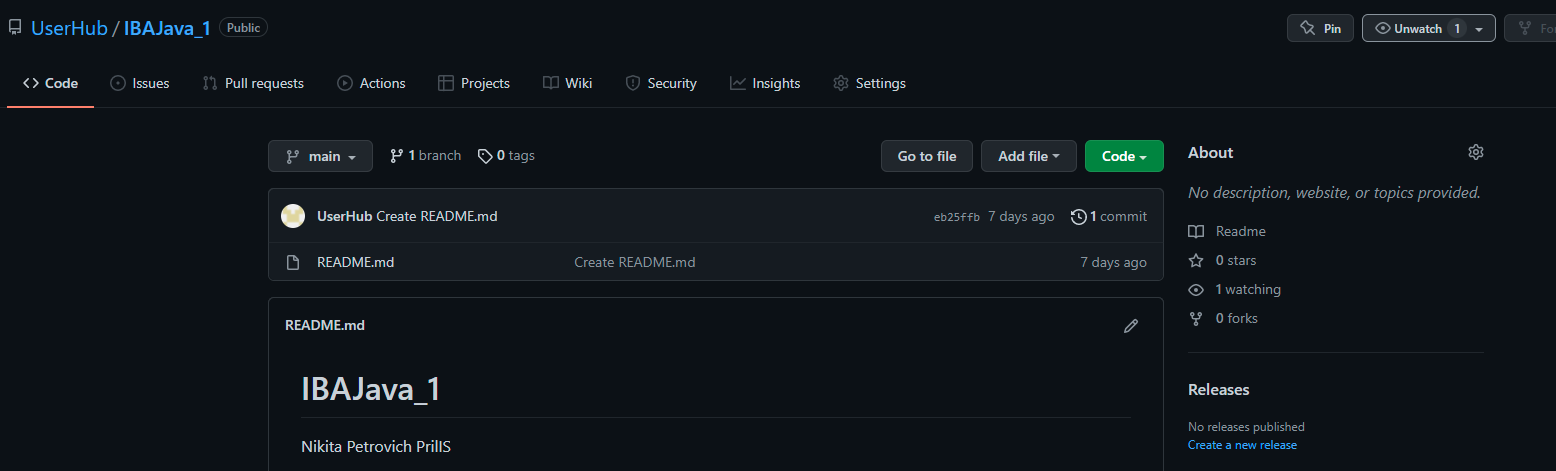
Все, что зафиксировано коммитом в Git, почти всегда можно восстановить. Но все, что вы потеряете, не сделав коммит, скорее всего, вам больше не увидеть.

## Клонирование существующего репозитория

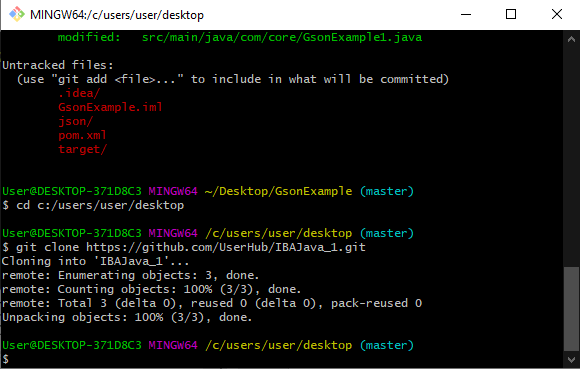
Получение копии существующего репозитория, например проекта, в котором вы хотите принять участие, выполняется командой git clone.

Команда git clone по умолчанию забирает все версии всех файлов за всю историю проекта.

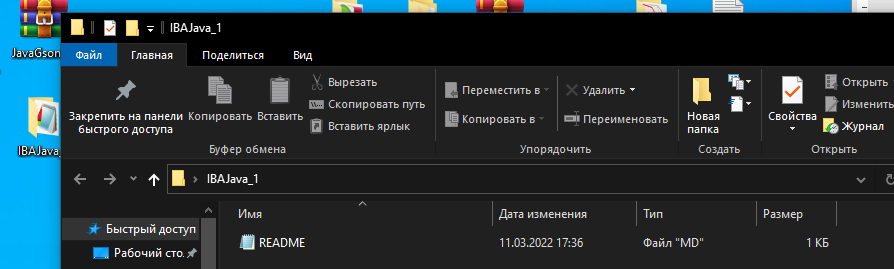
На GitHub у нас был репозиторий:



Клонирование репозитория осуществляется командой git clone [url].



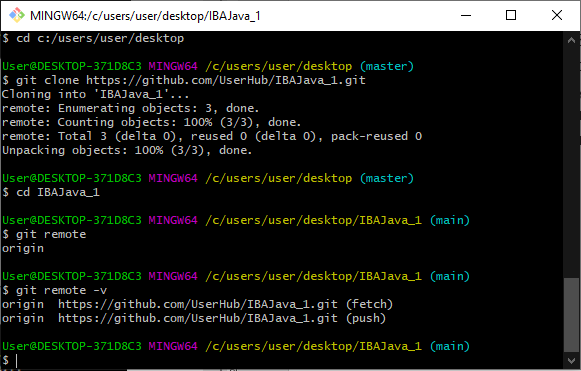
Команда создает папку с именем myIBAJava\_1, инициализирует в ней папку .git, считывает из репозитория все данные и выгружает рабочую копию последней версии. В папке вы найдете все файлы проекта, подготовленные к работе.



***Работа с удаленными репозиториями***

Удалённые репозитории представляют собой версии вашего проекта, сохранённые в интернете или ещё где-то в сети. У вас может быть несколько удалённых репозиториев, каждый из которых может быть доступен для чтения или для чтения-записи. Взаимодействие с другими пользователями предполагает управление удалёнными репозиториями, а также отправку и получение данных из них. Управление репозиториями включает в себя как умение добавлять новые, так и умение удалять устаревшие репозитории.

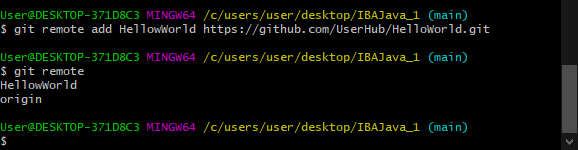
Просмотр уже настроенных удаленных серверов осуществляется командой git remote. Если репозиторий был клонирован, вы должны увидеть по крайней мере источник, то есть имя, которое Git по умолчанию присваивает клонируемому серверу. Параметр -v позволяет увидеть URL-адреса, которые Git хранит для сокращенного имени, используемого при чтении из данного удаленного репозитория и при записи в него:



## Добавление удаленных репозиториев

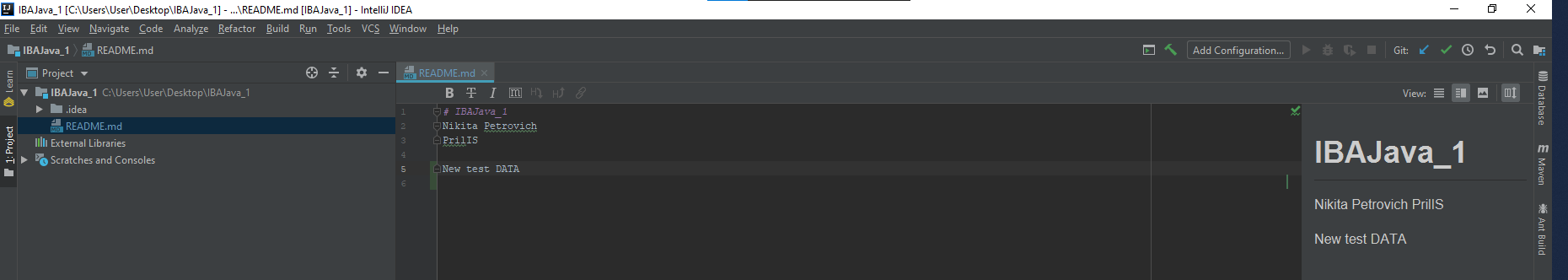
## 

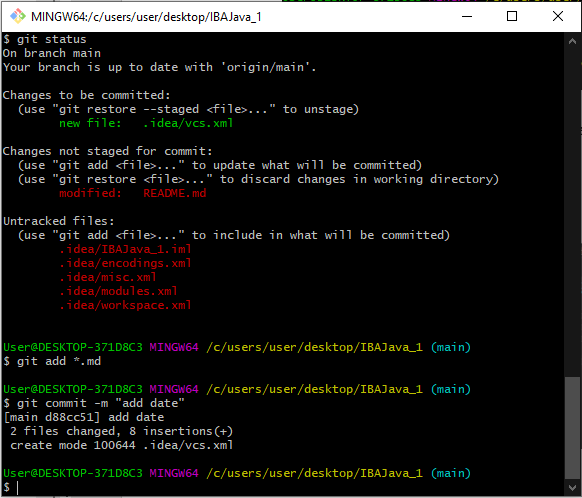
Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду:



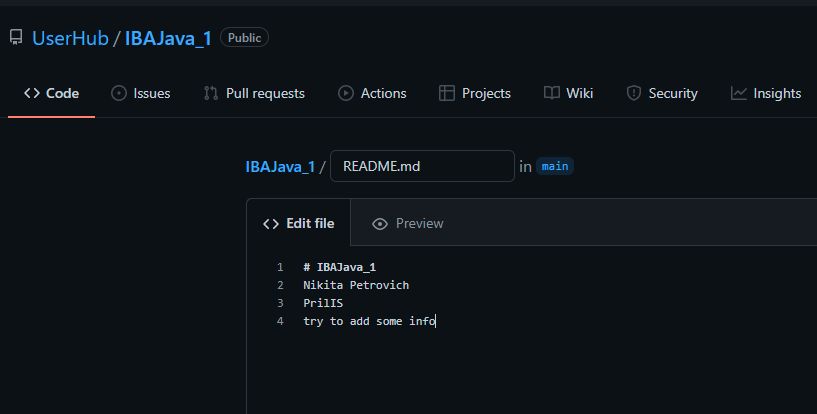
## Извлечение данных из удаленных репозиториев

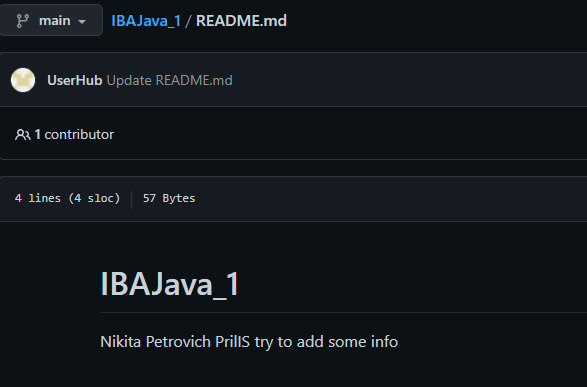
Внесем изменения в файл

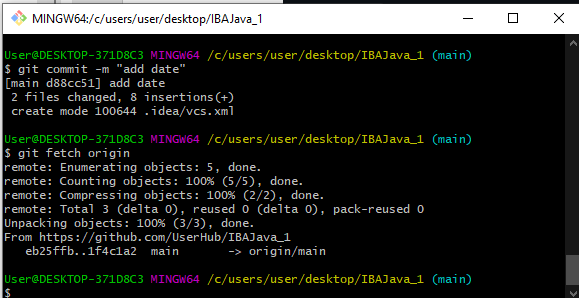




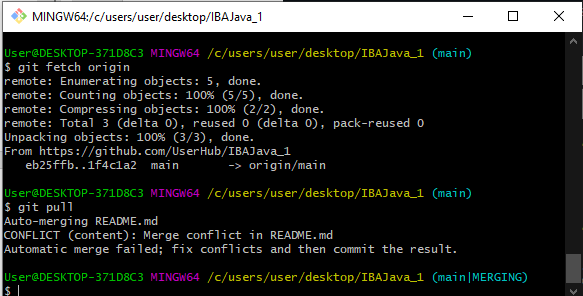
С другой стороны, через веб, добавим другие изменения в этот же файл.



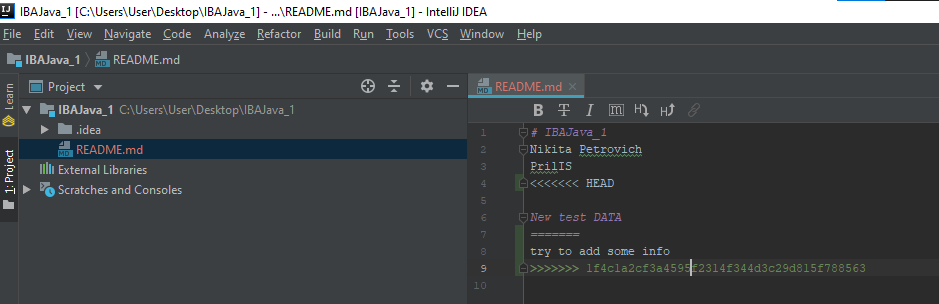




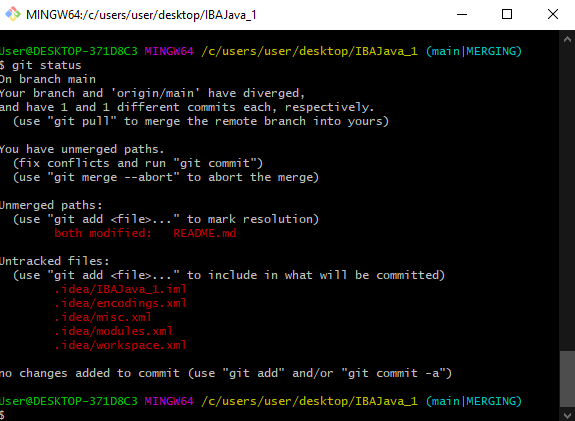
Используем команду выгрузки pull

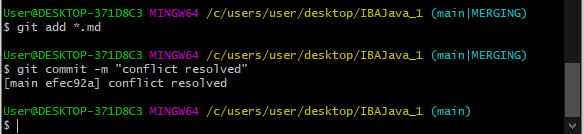


Видно, что произошел конфликт. Требуется его разрешить. Зайдем в файл в каталоге.



Внесем изменения и посмотрим статус





Извлечение данных из удаленных проектов выполняется такой командой:

$ git fetch [имя удаленного репозитория]

Эта команда связывается с удаленным проектом и извлекает оттуда все пока отсутствующие у вас данные. После этого у вас должны появиться ссылки на все ветки удаленного проекта, которые можно подвергнуть слиянию или просмотреть.

При клонировании данная команда автоматически добавляет удаленный репозиторий под именем «origin». Соответственно команда git fetch origin извлекает все, что появилось на этом сервере после его клонирования (или после момента последнего извлечения информации). Важно понимать, что команда git fetch помещает все данные в ваш локальный репозиторий, — она не выполняет автоматическое слияние с ветками, с которыми вы работаете в данный момент, и вообще никак не затрагивает эти ветки. Слияние вы выполните вручную, как только в этом возникнет необходимость.

***Отправка изменений в удаленный репозиторий (Push)***

Когда вы хотите поделиться своими наработками, вам необходимо отправить (push) их в главный репозиторий. Команда для этого действия простая: git push [remote- name] [branch-name]. Чтобы отправить вашу ветку master на сервер origin (повторимся, что клонирование, как правило, настраивает оба этих имени автоматически), вы можете выполнить следующую команду для отправки наработок на сервер:

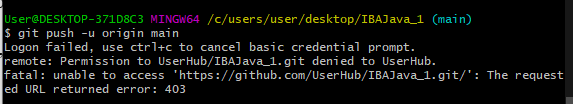
$ git push origin master

В начале нужно:

<https://exerror.com/remote-support-for-password-authentication-was-removed-on-august-13-2021-please-use-a-personal-access-token-instead/>

Эта команда срабатывает только в случае, если вы клонировали с сервера, на котором у вас есть права на запись, и если никто другой с тех пор не выполнял команду push. Если вы и кто-то ещё одновременно клонируете, затем он выполняет команду push, а затем команду push выполняете вы, то ваш push точно будет отклонён. Вам придётся сначала вытянуть (pull) их изменения и объединить с вашими. Только после этого вам будет позволено выполнить push.

Еще возможно возникновение конфликтов, если у вас уже были учетная запись на github и тогда вы получите:

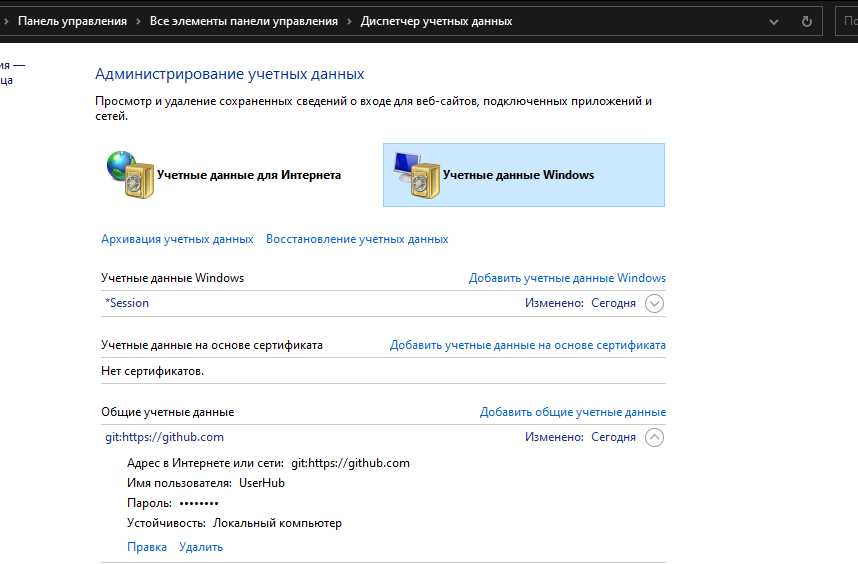


Ошибка говорит нам о том, что мы пытаемся отправить данные в чужой репозиторий. И о том, что текущий пользователь Git не имеет прав для отправки данных в указанный в команде репозиторий.

Можно 1) дать текущему пользователю права для работы с репозиторием. Данный способ подойдет, если вы являетесь владельцем обеих учетных записей указанных в ошибке. Необходимо зайти на GitHub под именем того пользователя, в чей репозиторий вы не можете отправить данные. В настройках репозитория указать необходимого соавтора. Ему отправится приглашение, которое необходимо подтвердить, перейдя по ссылке.

После этого вы сможете работать с репозиторием как со своим и ошибка больше не появится.

2) Назначить в системе текущим пользователем Git ту учетную запись, в которую не можете отправить данные



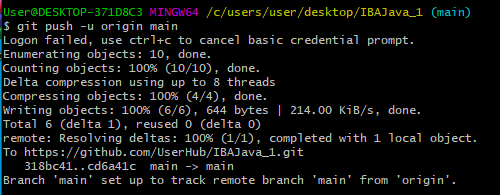
Нажмите удалить

После этого при попытке выполнить команду:

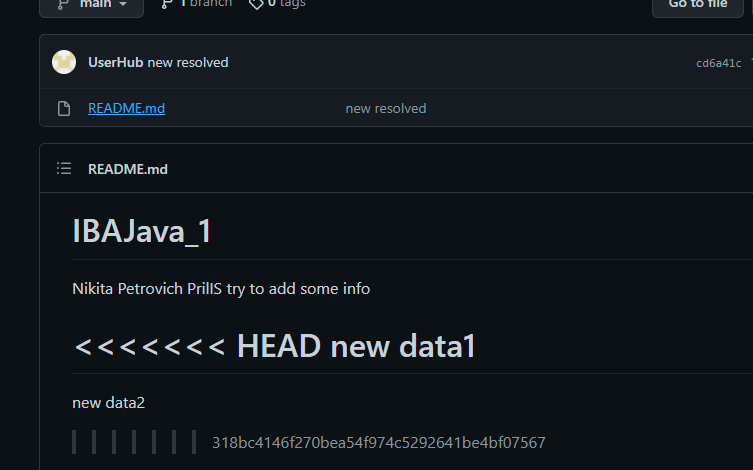
$ git push -u origin main

Терминал запросит у вас пароль от учетной записи на Github в чей репозиторий вы пытаетесь отправить данные.

После ввода пароля система установит данного пользователя как основного и будет использовать его в дальнейшем при работе с Git.



Изменения:



Т.е. все данные из вашего проекта перенесены на удаленный репозиторий.

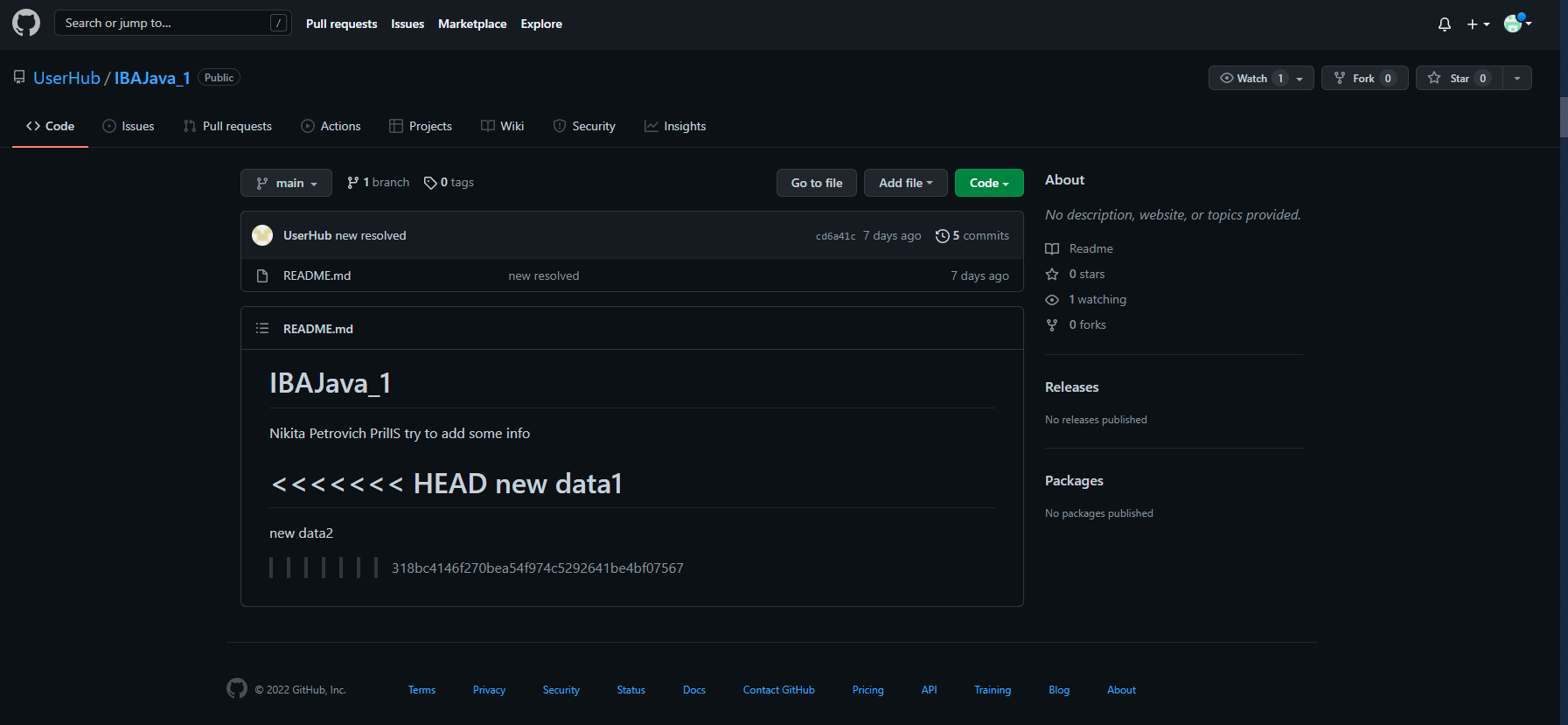
## Совместная работа

Есть два варианта совместной работы.

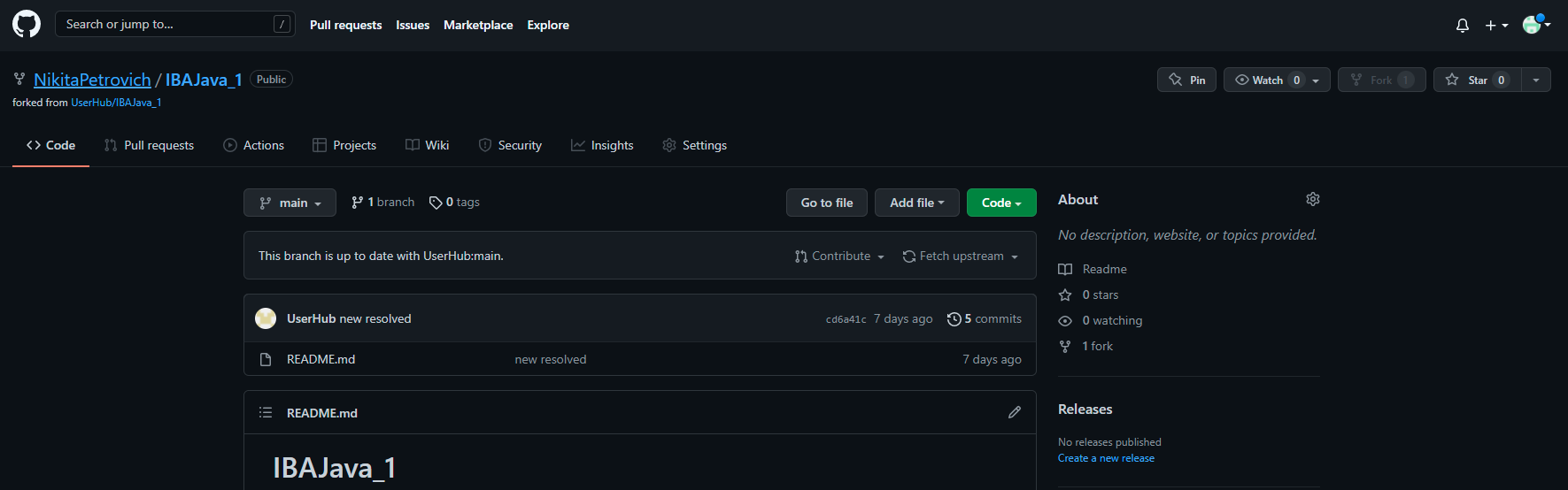
Если количество участников проекта скажем два, то можно работать с одним общим репозиторием и несколькими ветками (как будто это один человек). В этом случае каждый человек имеет свою ветку и общую ветку куда помещаются и объединяются изменения.

Второй вариант – у нас есть репозиторий, в который будут приходить pull request и фиксироваться владельцем репозитория.

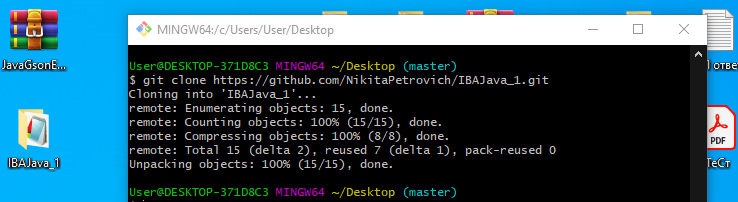
Для этого зайдем под другим пользователем и найдем нужный репозиторий, например, Java. В вашем случае можно найти репозиторий коллеги над которым вы хотели бы вести совместную работу.



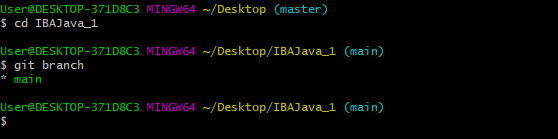
Найдите копку Fork и нажмите ее. После этого создается копия этого репозитория, но уже в вашем аккаунте.



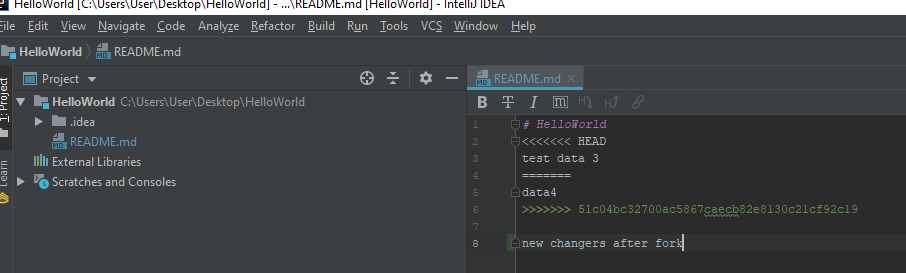
И теперь этот уже свой репозиторий клонируем себе на локальную машину.



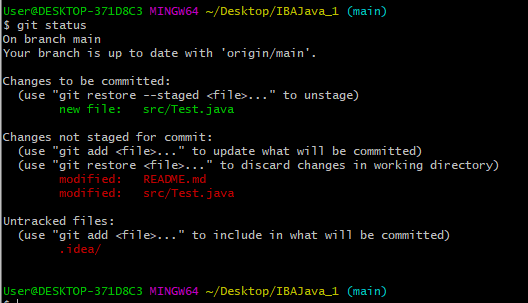
Давайте туда перейдем и посмотрим есть ли какие-нибудь ветки.



Внесем пару изменений



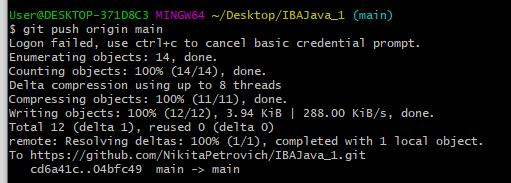
Проверим изменения и добавим их:



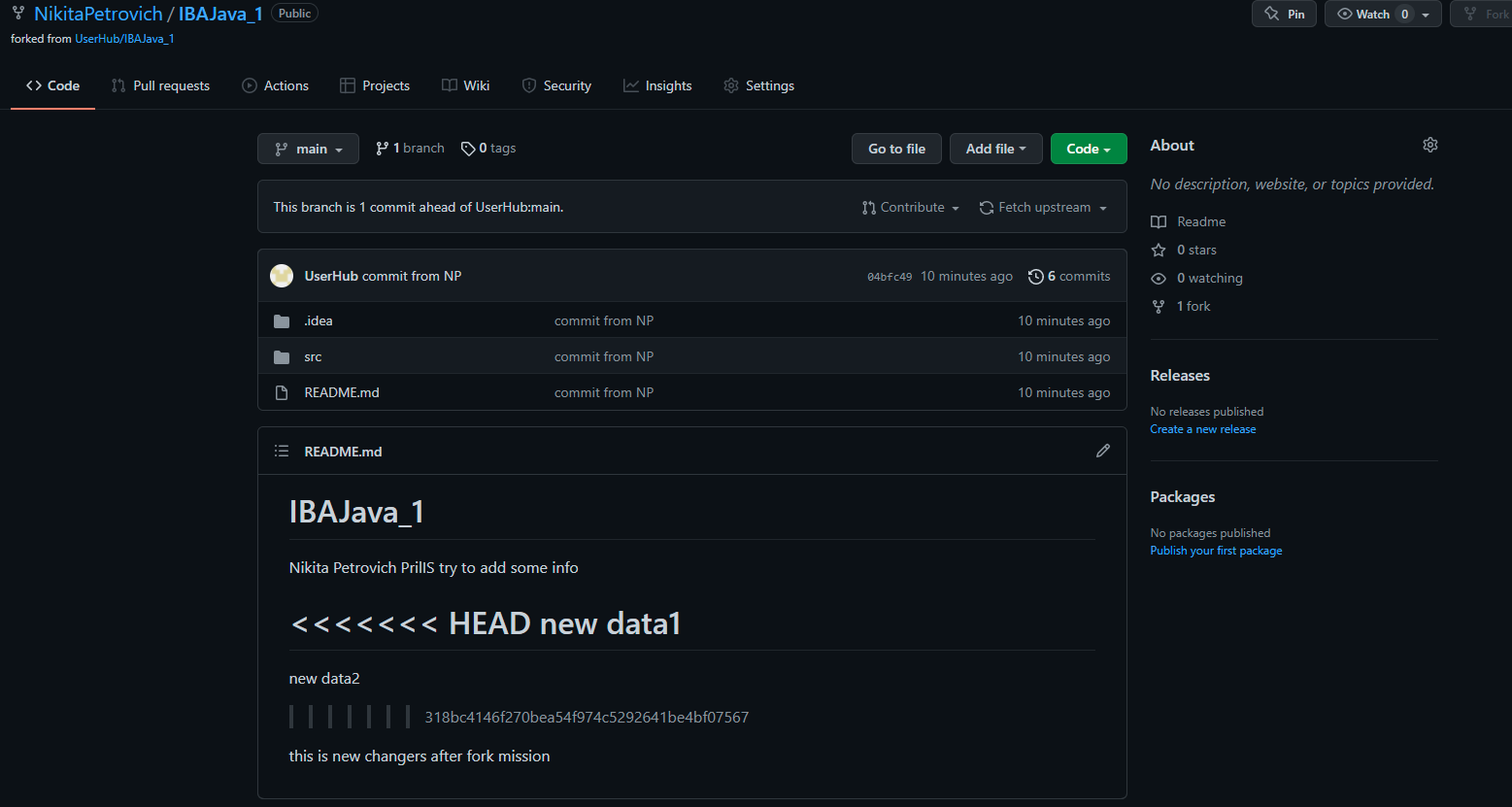
Сделаем commit



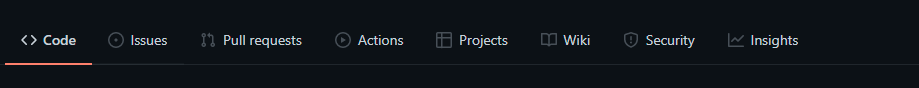
Теперь отправляем изменения в свой удаленный облачный репозиторий



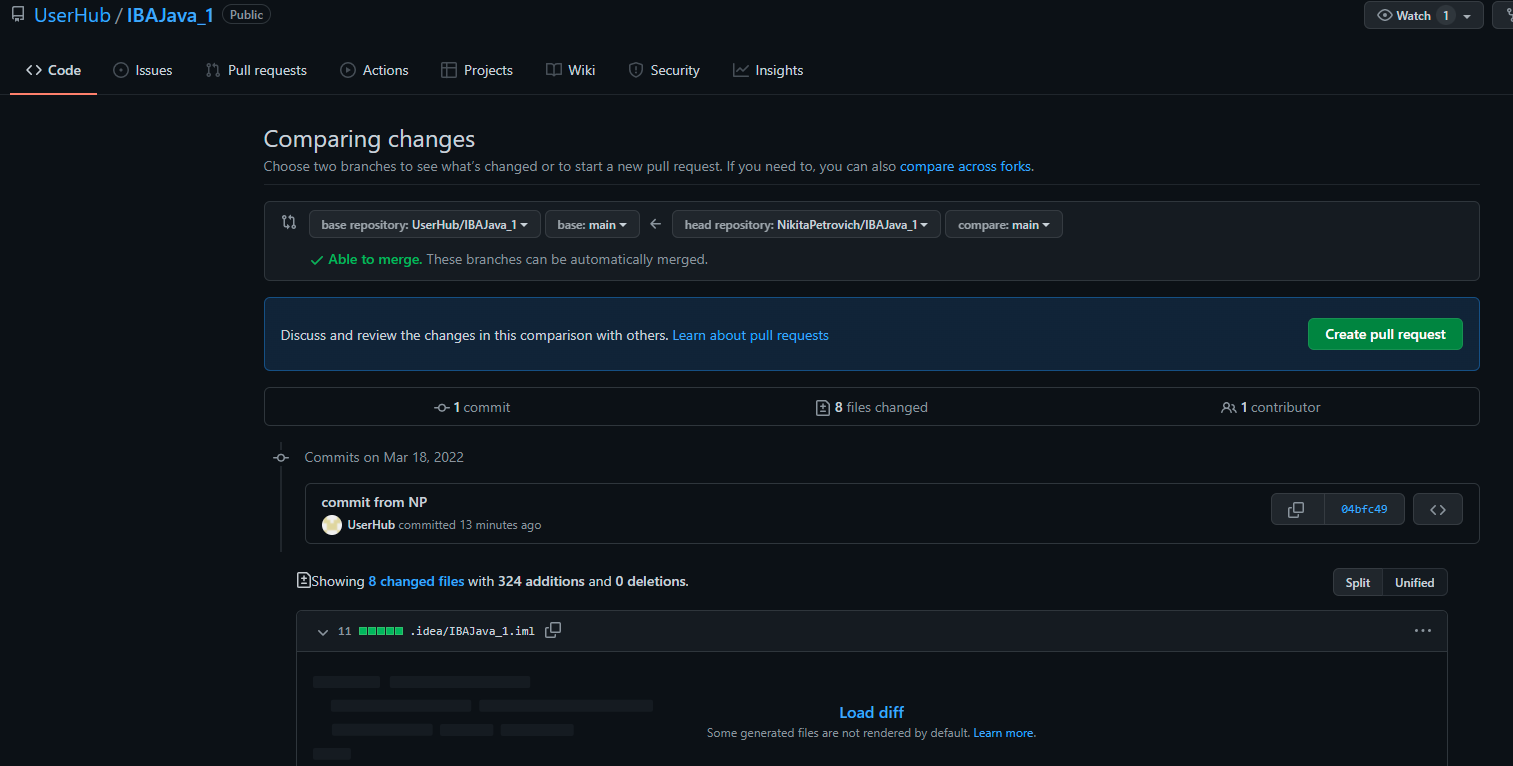
Посмотрите, что изменения принялись.

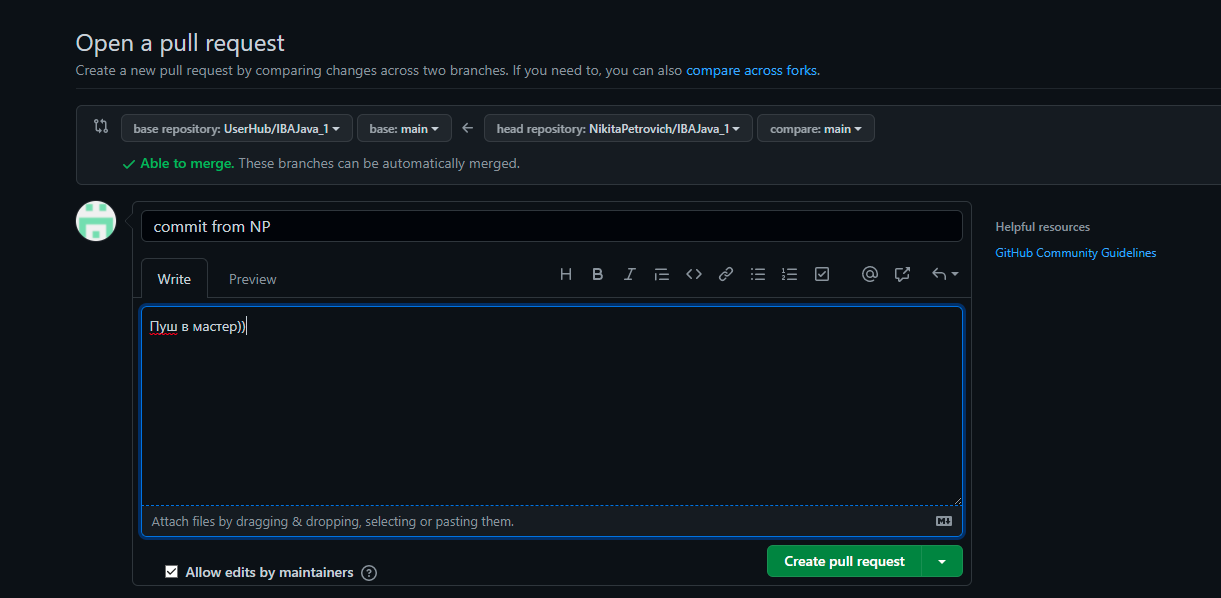


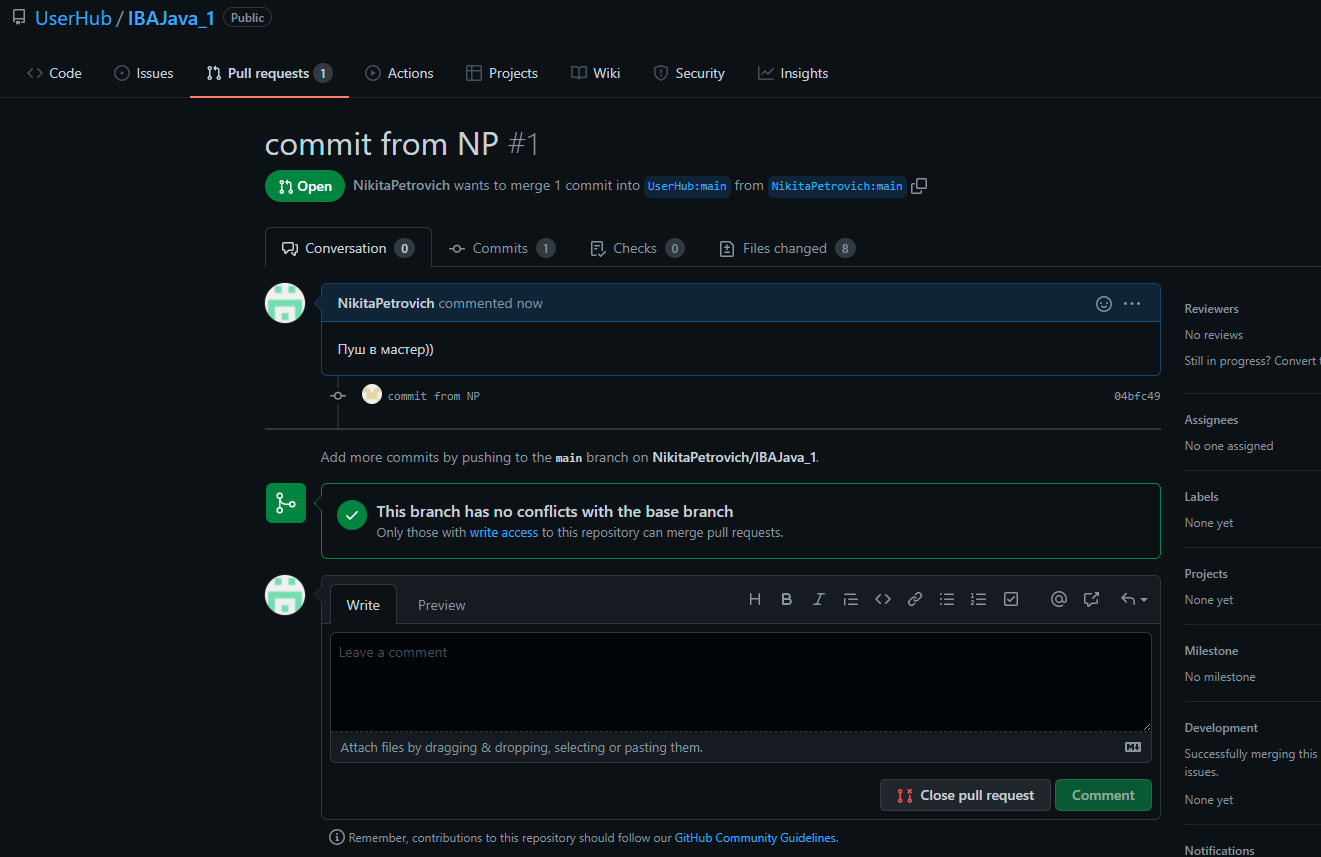
Создаем Pull requests через интерфейс Github. Для этого есть кнопка, которая так и называется



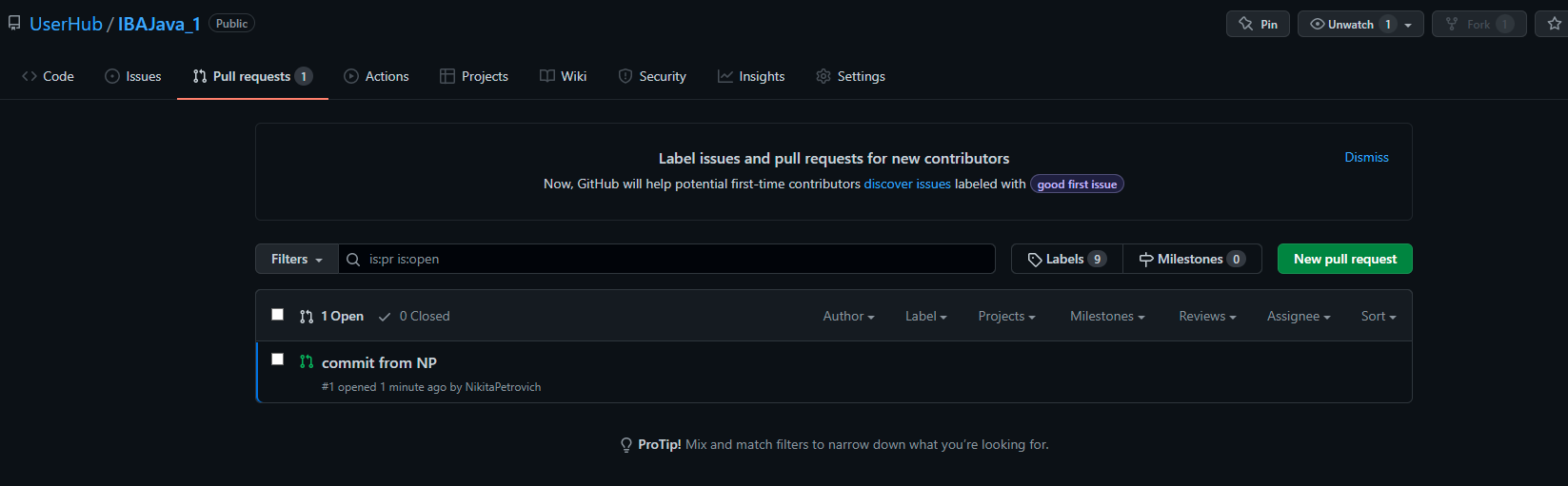
Далее подтверждаем его и смотрим что добавлено в запрос на изменение.



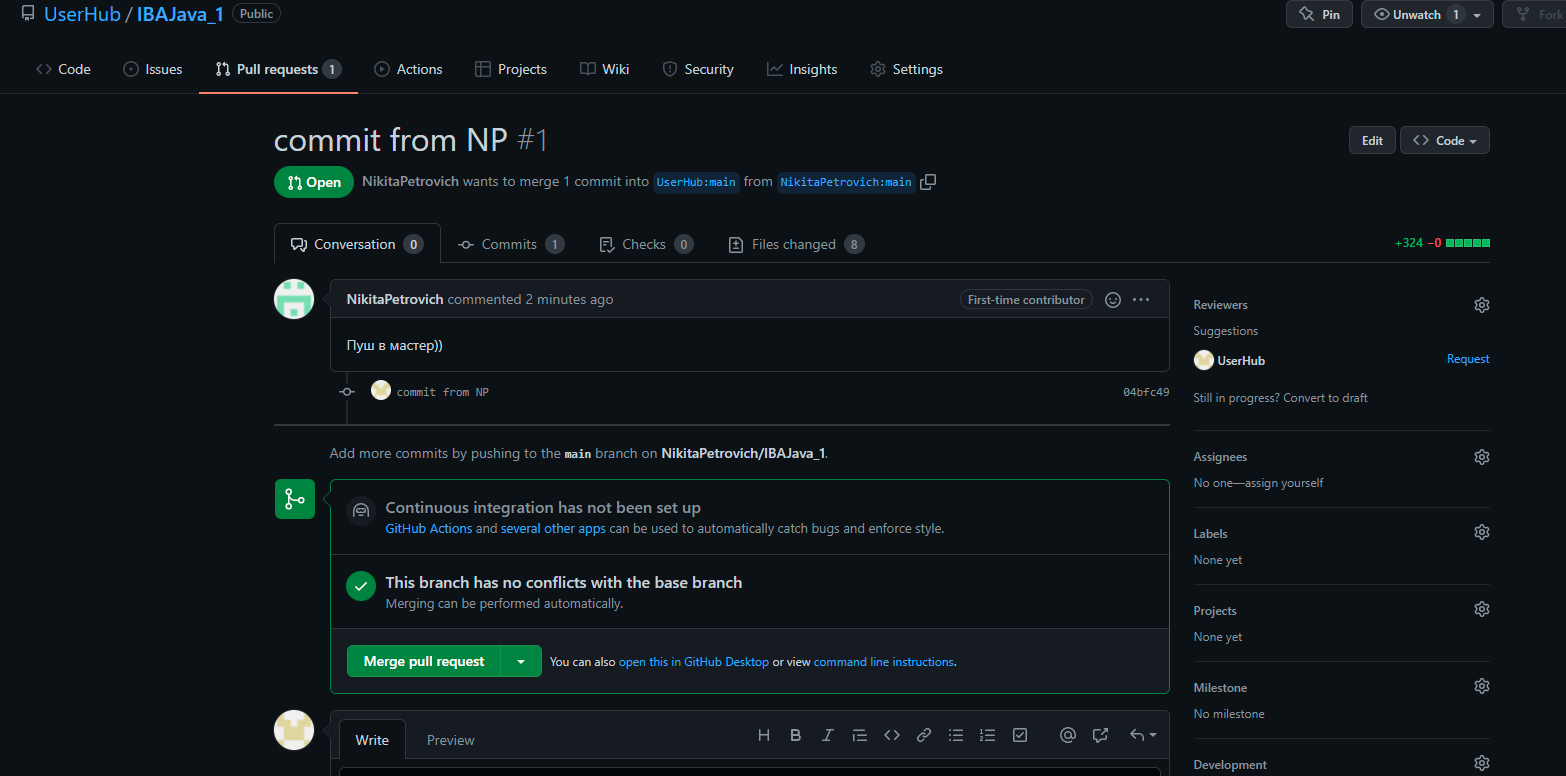




Заходим под другим пользователем:

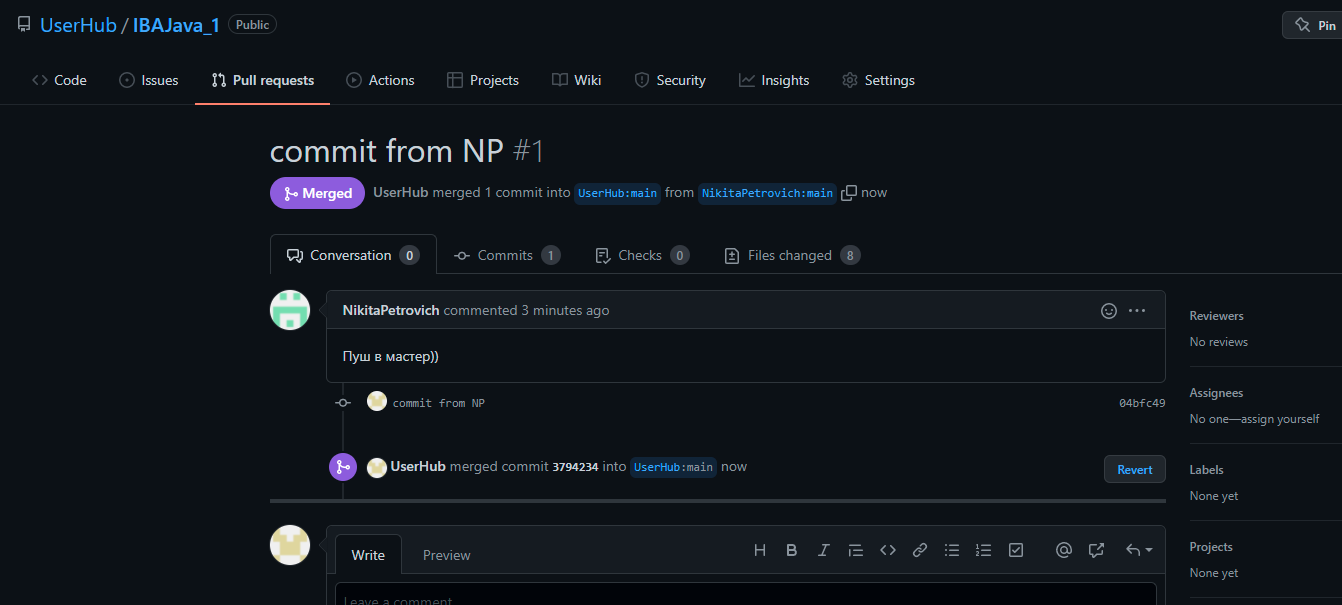


Заходим в доступный запрос

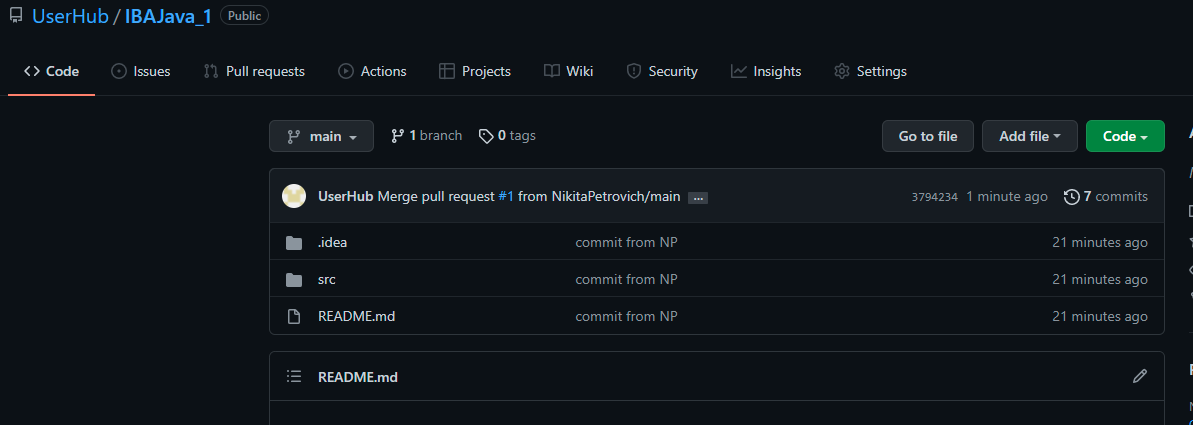


И подтверждаем его с помощью merge, confirm merge

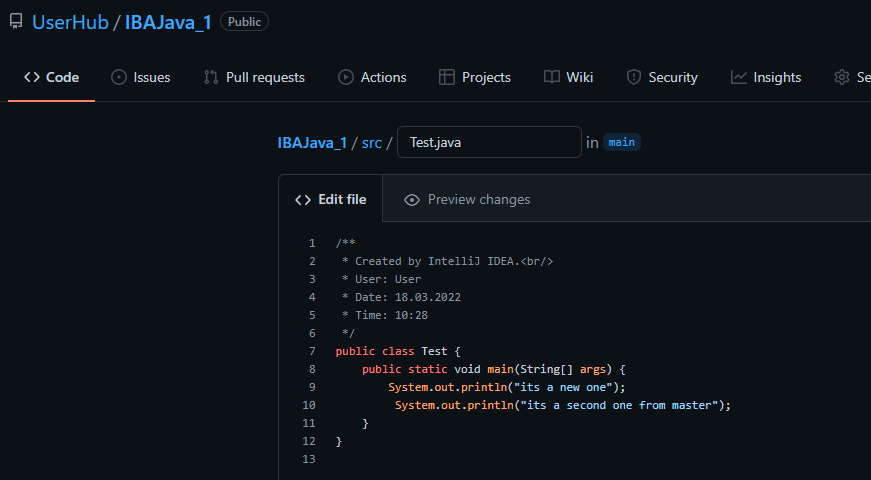
После чего видим примерно такой вариант:



Все изменения применены и наш лид видит видит весь код и все изменения:

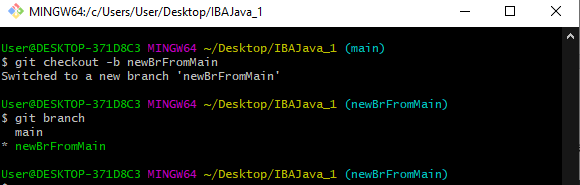


Теперь владелец репозитория внесет изменения и нам тоже надо их получить.

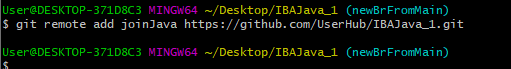


Изменения внесены. С нашей точки зрения, кто хочет получить новые данные:

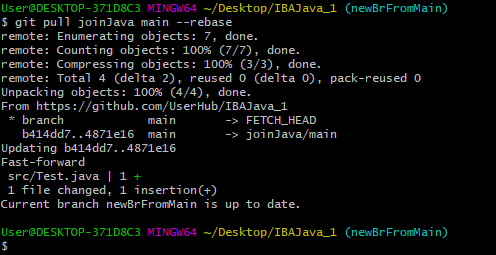
Для этого сделаем новую ветку и переключимся в нее



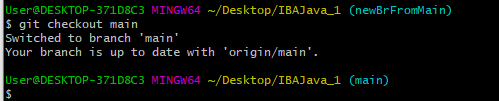
Зафиксируем ряд изменений с новой ветки для себя.



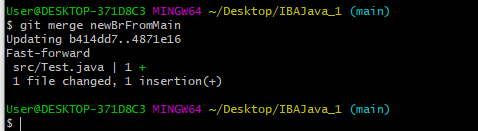
И скопируем репозиторий (все изменения из удаленного репозитория), который называется joinJava в ветку newJava. Флаг –rebase говорит, что надо перезаписать ветку.



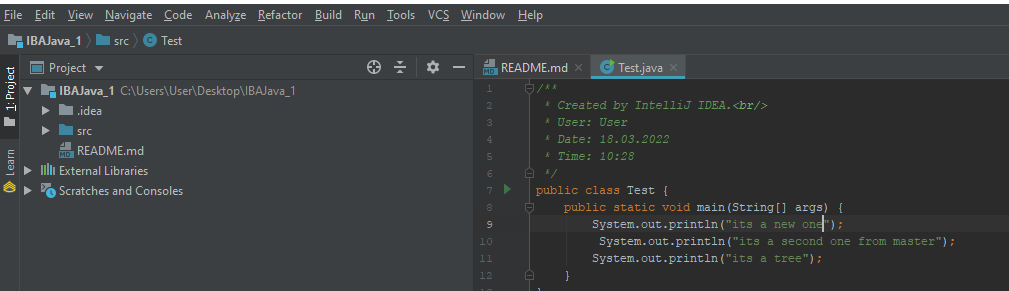
Перейдем в ветку main



И сольем ветки:



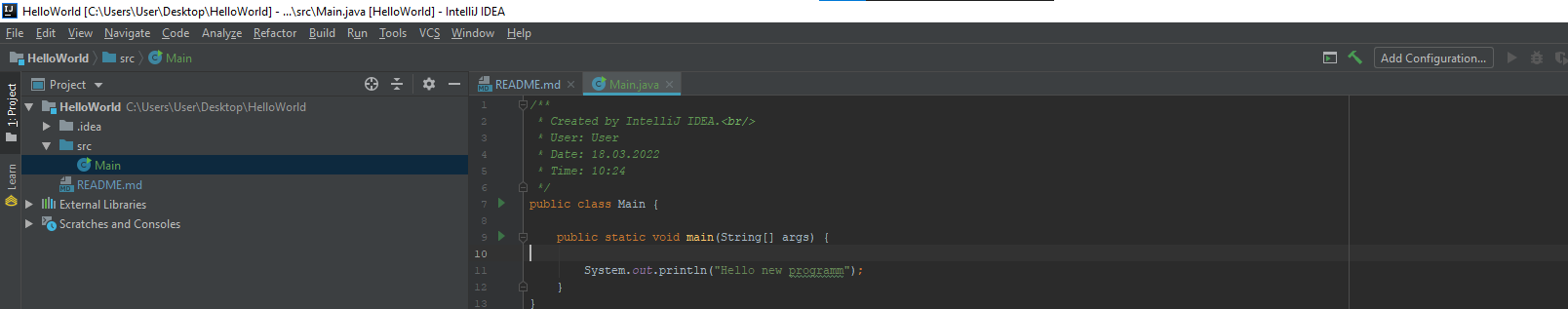
Зайдем и убедимся что мы получили измеренный репозиторий



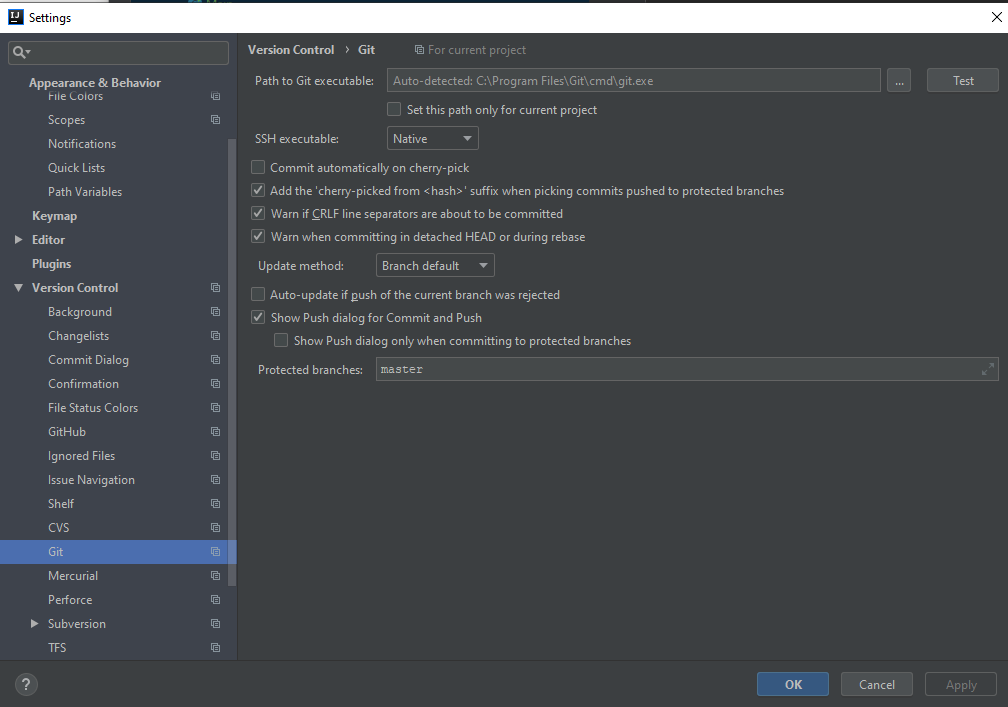
Все данные получены с репозитория владельца.

**Интеграция с IntelijIdea**

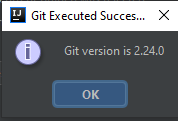
Создайте проект



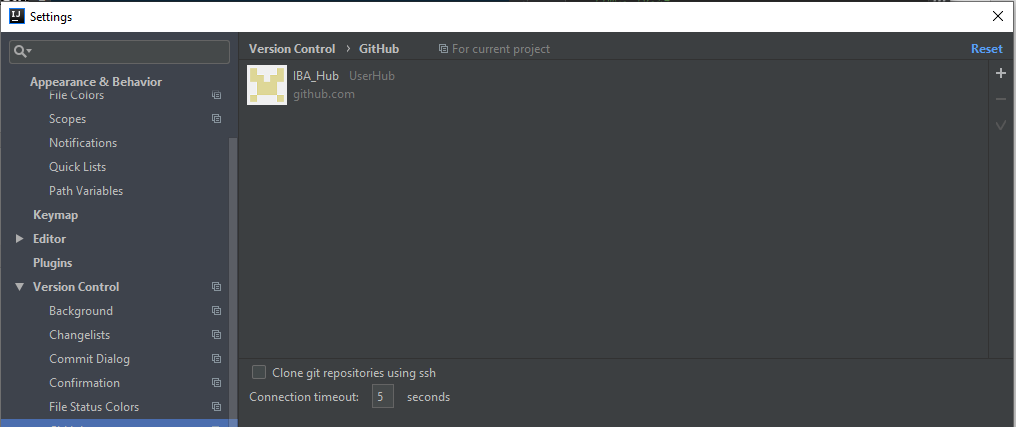
Зайдите в настройки в раздел Version Control - Git



Нажмите кнопку тест



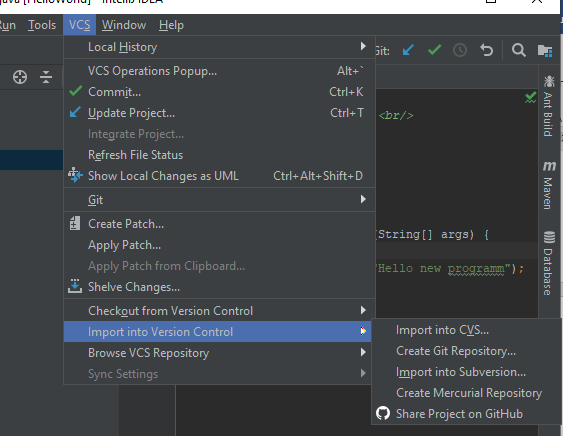
Зайдите на вкладку GitHub. Нажмите справа на + и введите логин и пароль.



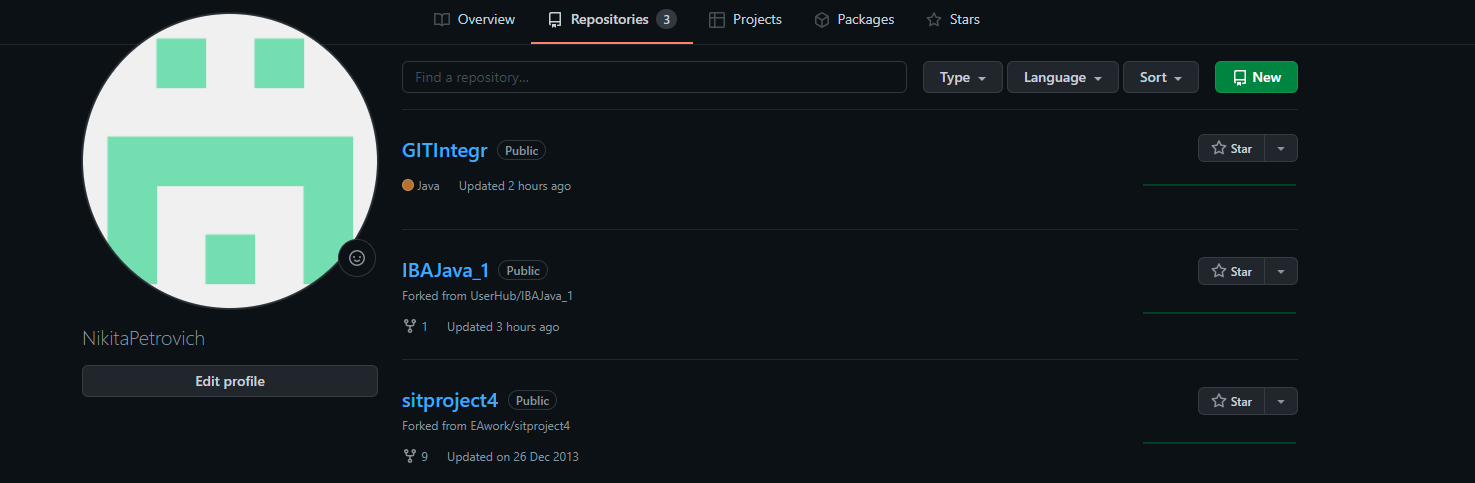
Выполните VCS - Import into Version Control - Share Project on Gthub

ghp\_VH3k1Y6D4wnpn2fzHGu0vm1tkHxCoD4JfAgm

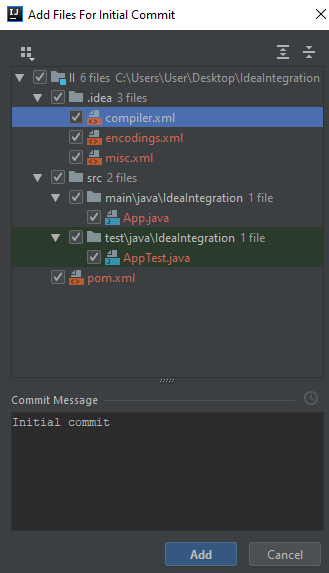
ghp\_X3OUcUEZSf67dDIHUMKDJM2pAanwNV0YmCkh



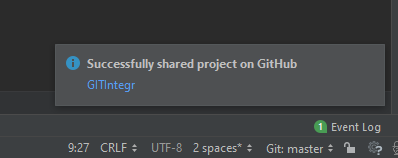
Зайдем на сайт и проверим репозиторий:



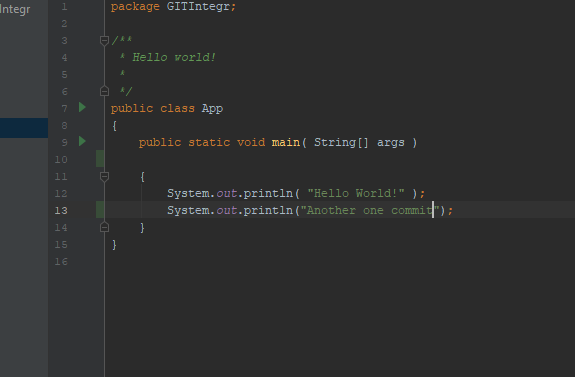
Выбираем файлы для commit



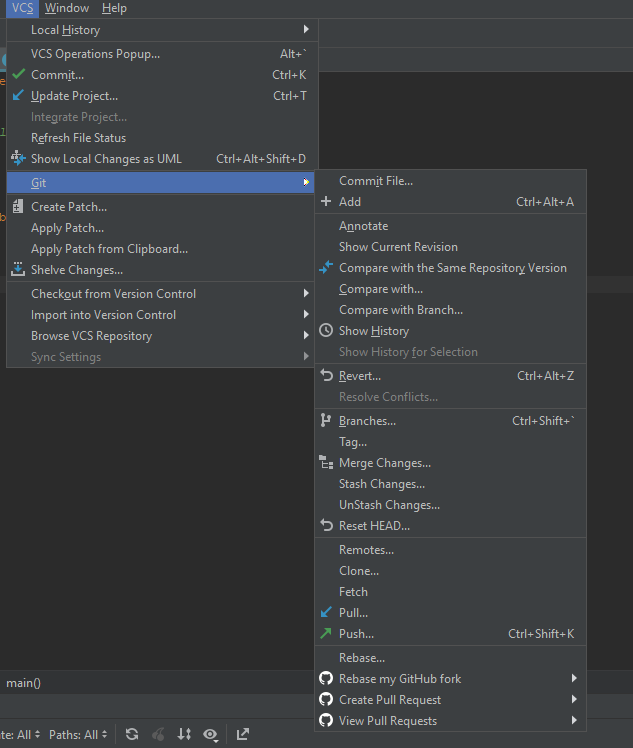
После чего должны увидеть что-то вроде этого:



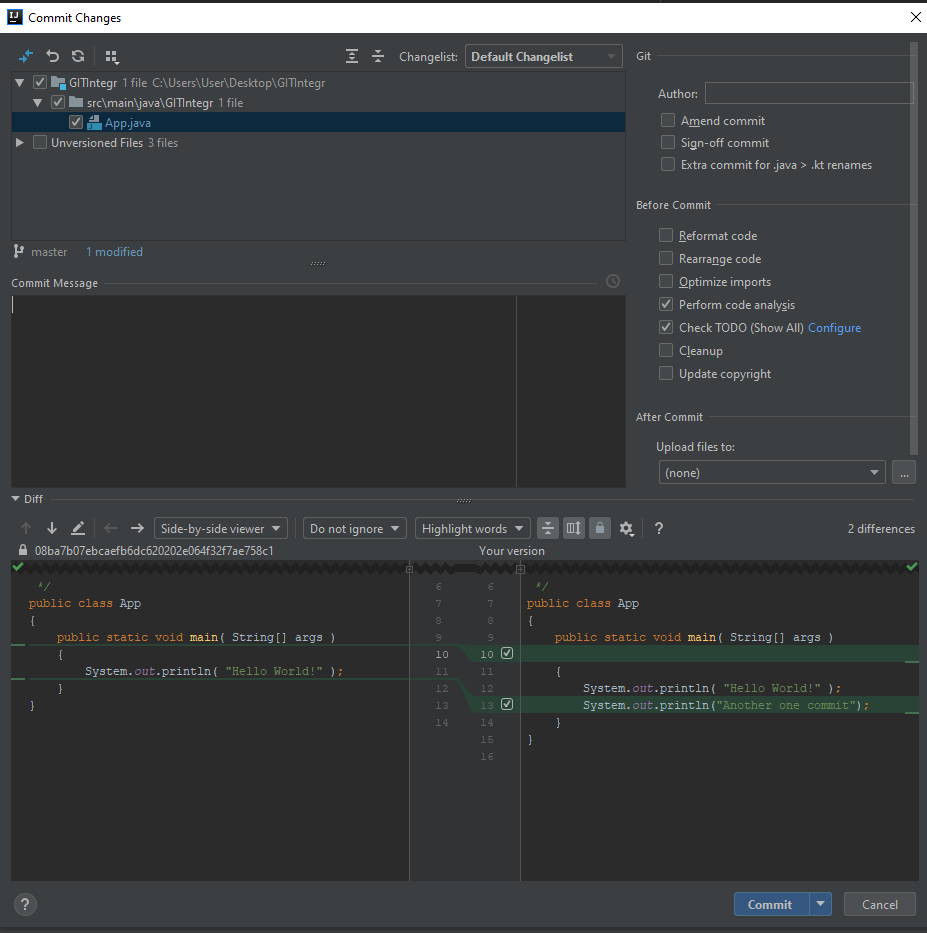
Внесем изменения в файлы:



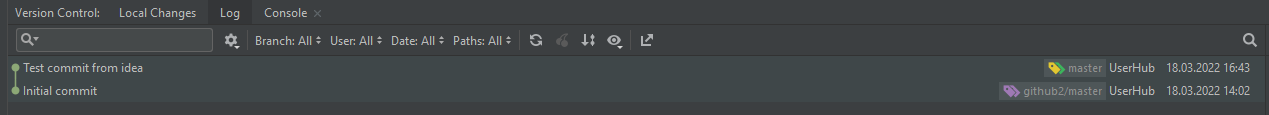
Обратите внимание на команды доступные нам:



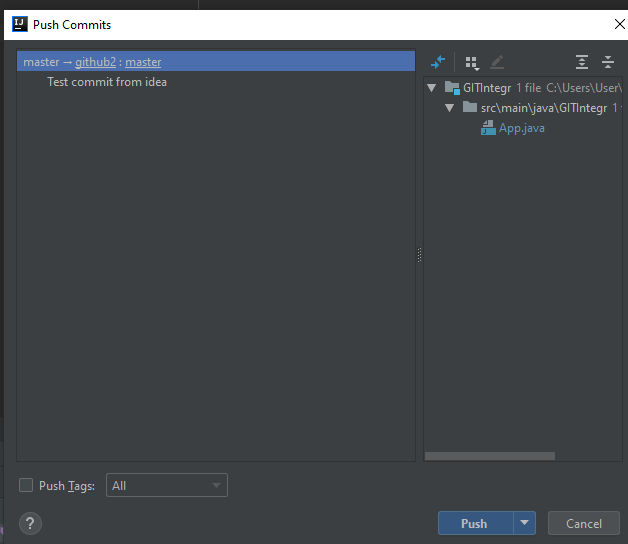
Сделайте новый коммит чтобы зафиксировать изменения. Посмотрите информацию в окне изменений и добавьте комментарий



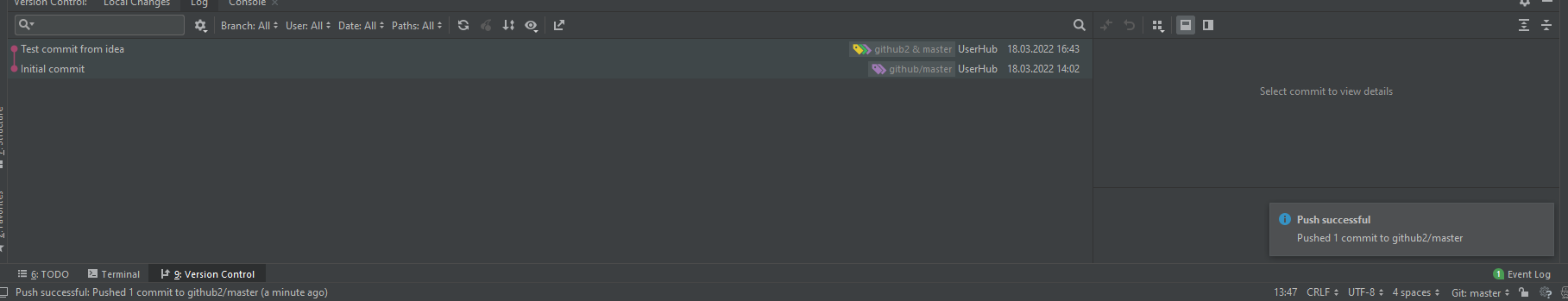
Обратите внимание на логи среды:



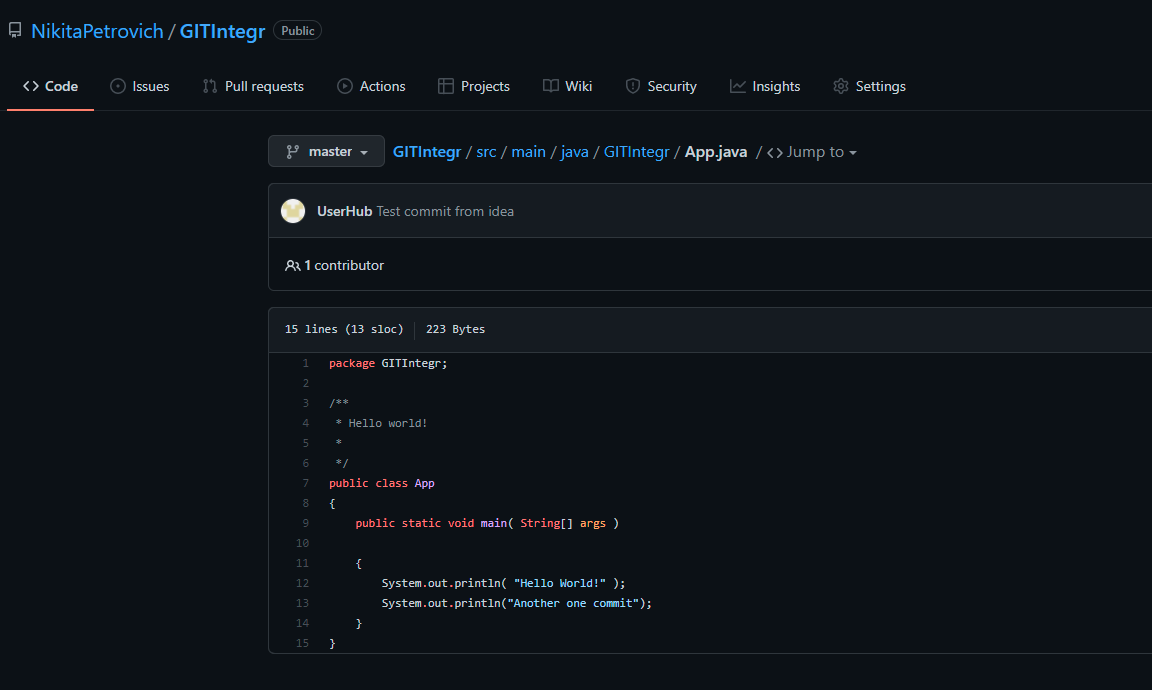
Затем выполните команду VCS – Git - push



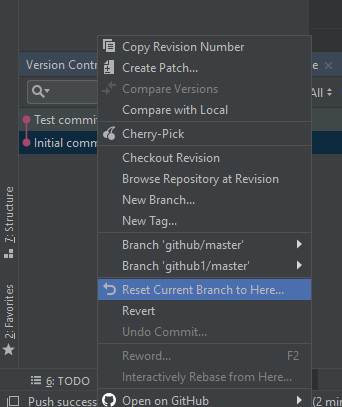
И соответственно подтверждение:



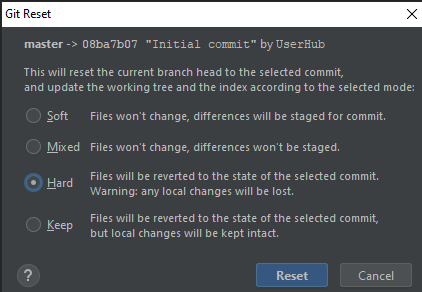
Проверьте изменения на сервере:



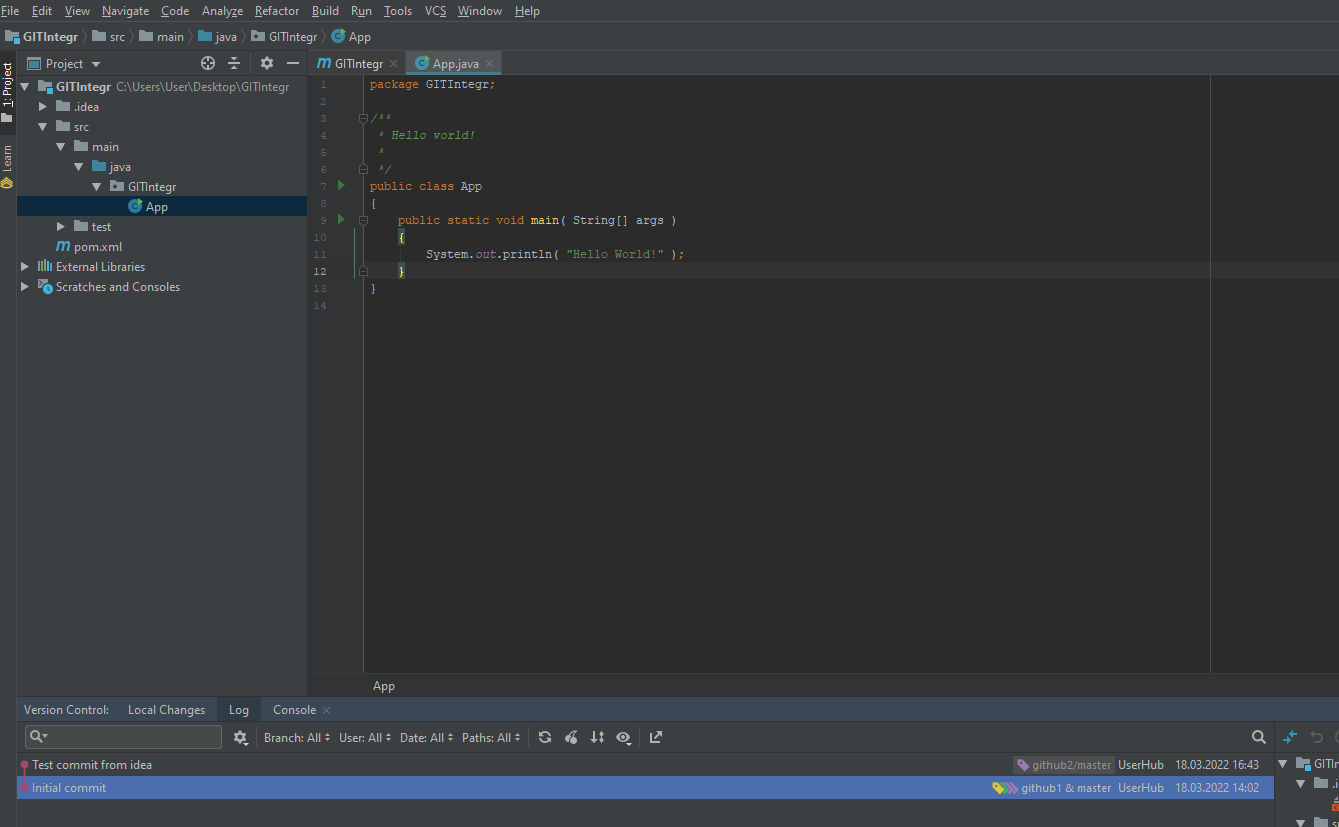
Давайте вернемся к предыдущему состоянию проекта. Команда -Reset Current Branch….



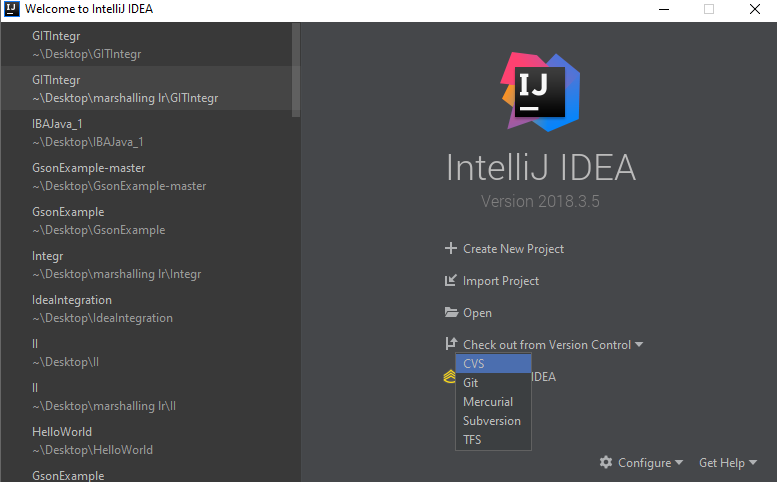
Выберете способ:



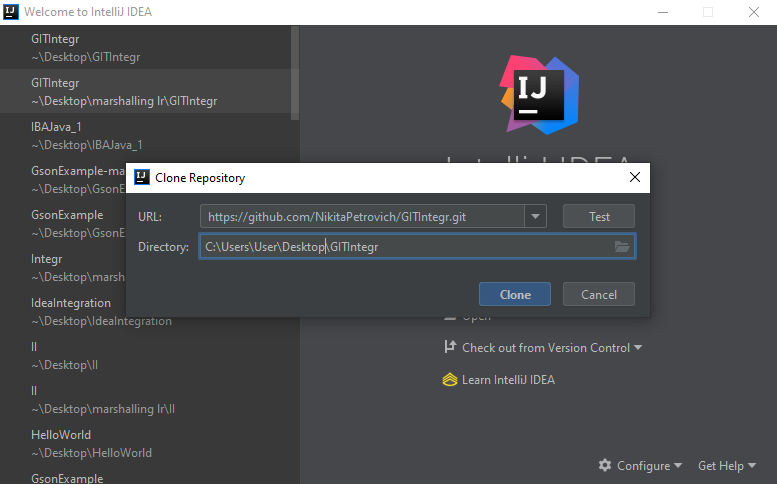
И просмотрим результат:



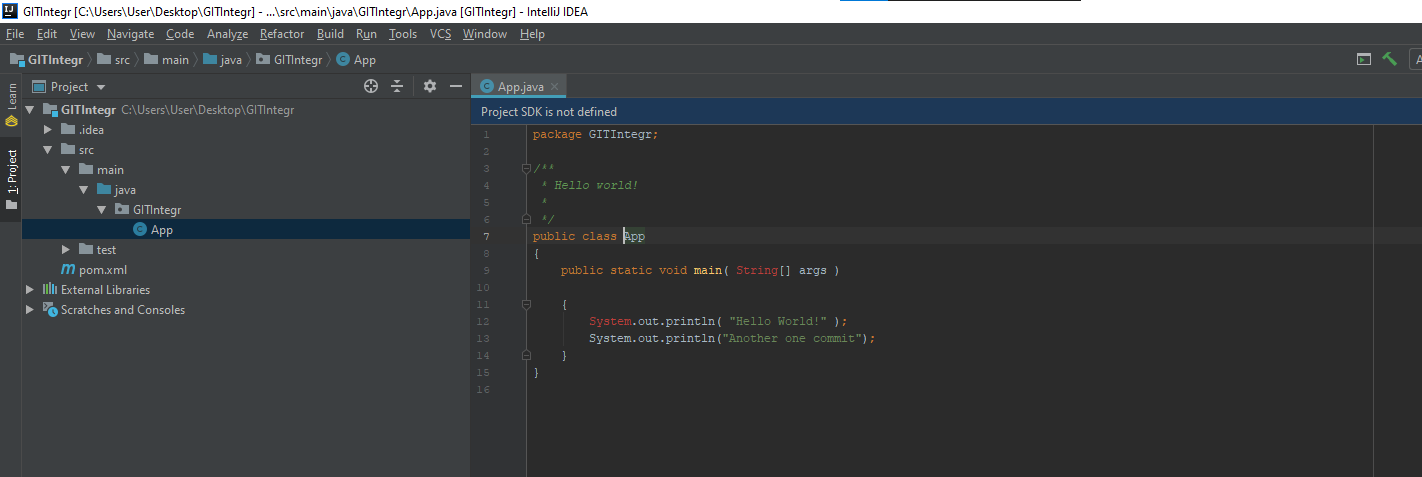
А теперь давайте удалим папку и загрузим ее из удаленного репозитория с помощью IDEA



Выберите URL. Нажмите clone

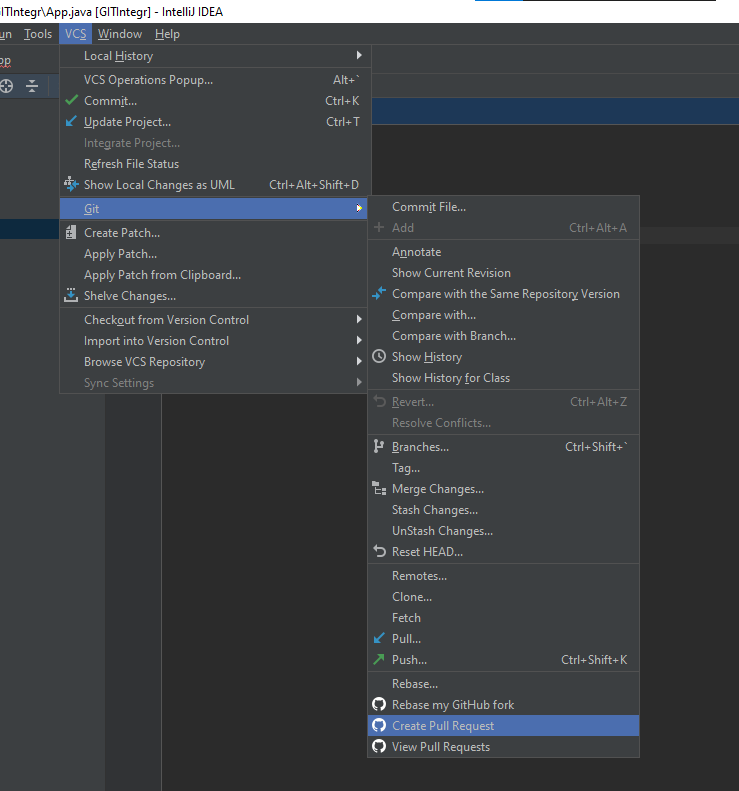


Далее следуем инструкциям:



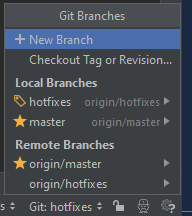
Для поддержки совместной работы есть команды: Create pull request

Rebase



На одном ПК сделать сложно, но можно:

1. На первом аккаунте создаем проект и клонируем его к себе на ПК. (регистрация внутри idea так же под первым аккаунтом)
2. Входите на второй аккаунт(и в idea так же перерегистрируйте аккаунт). Ищите проект на гитхабе у первого профиля и нажимаете fork. После чего у вас появится проект на вашем профиле. Открываете новый проект (лучше пустой) и клонируете себе проект из своего репозитория форкнутый у первого репозитория. VCS – checkout version control – git.
3. Для получения всех обновлений каждый раз делаете VCS – Git – rebase my GIT. И отправляете в свой репозиторий VCS – git – push
4. Создаете свою ветку в проекте

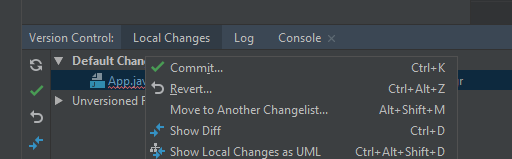


У меня это hotfixes



1. Работает в проекте, что то добавляете.

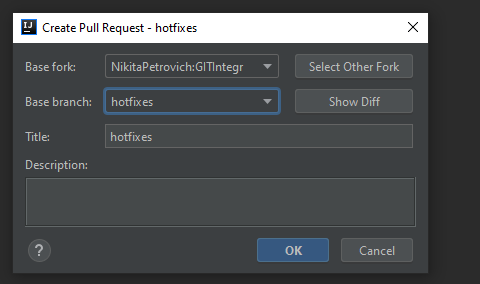
Далее добавляем в код изменения и делаем коммит:



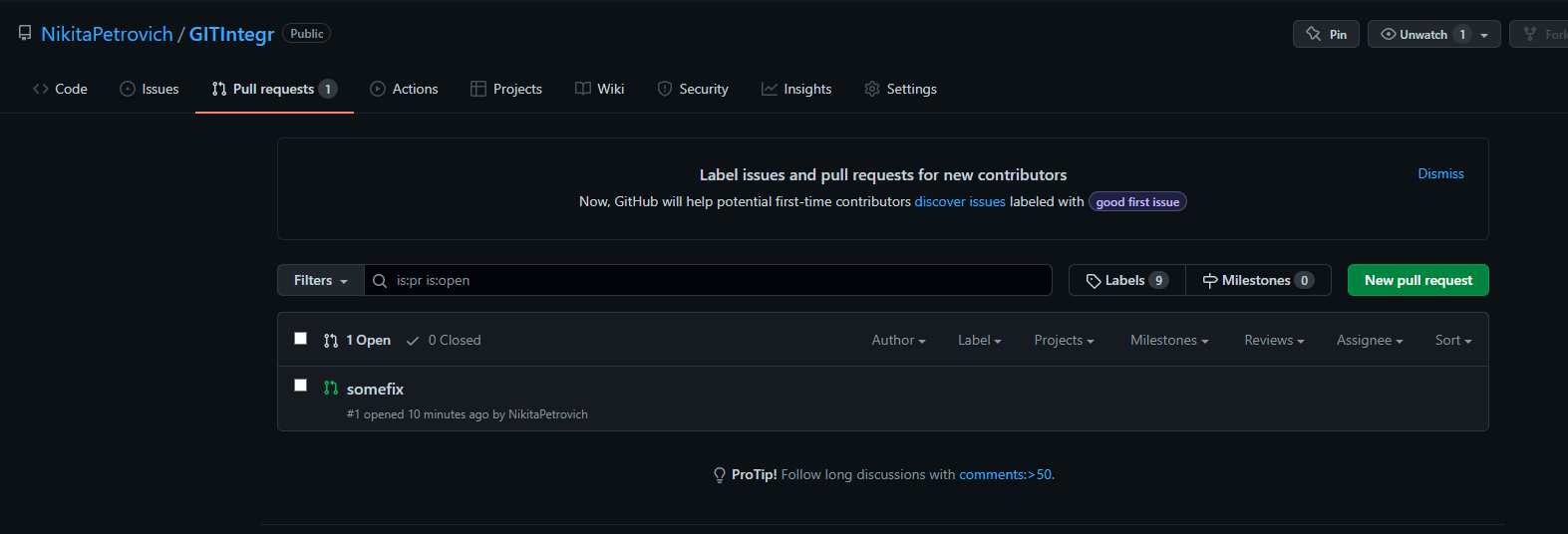
Важно. Вы должны быть зареганы от пользователя, который зависим от Владельца репозитория. Т.е. того пользователя, кто будет делать запрос на изменение.

1. Делаем коммиты.
2. Далее PUSH
3. После чего VCS – git – Pull reqest и выбираем уже первый аккаунт

Выбираем ветку нашу



После чего мы сможем с вами увидеть запрос и на сайте:



И в системе:

