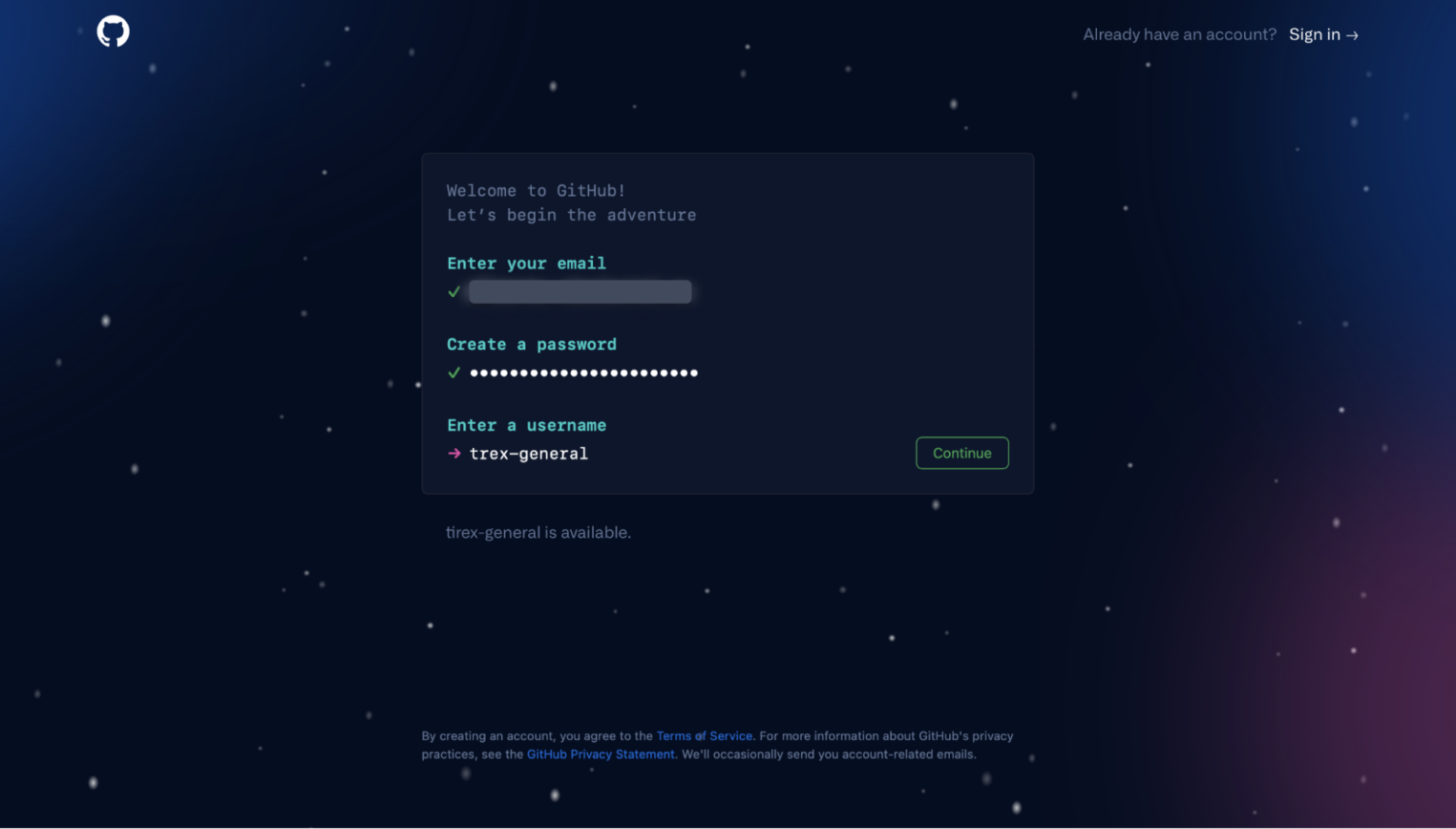
## **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ НА GITHUB**

## **Регистрация на GitHub: создание аккаунта**

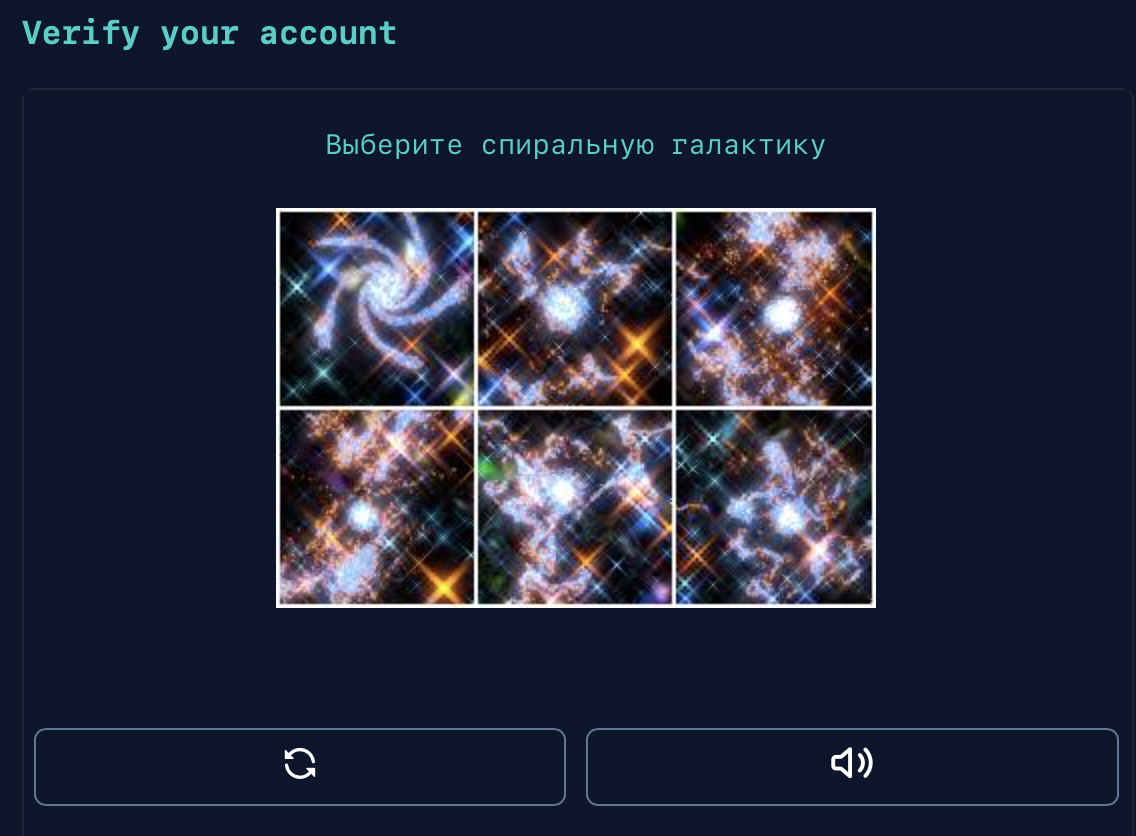
Для работы с платформой нужно создать аккаунт. Для этого переходим по [ссылке](http://github.com/) и тапаем по кнопке **Sign up**.

На [странице регистрации](https://github.com/signup) вводим данные:

1. **Адрес электронной почты**. Если на почту уже был зарегистрирован аккаунт, на странице появится сообщение об ошибке: «Email is invalid or already taken».
2. **Пароль**. Система рекомендует использовать для пароля последовательность из 15 символов или 8, но с использованием хотя бы одной цифры и строчной буквы.
3. **Имя пользователя.**«Юзернейм» должен быть уникальным. При этом он не может начинаться или заканчиваться дефисом.



Теперь нужно нажать кнопку **Continue**, принять или отклонить предложение о подписке на рассылку и пройти экстравагантную валидацию:



Затем подтвердите адрес электронной почты.

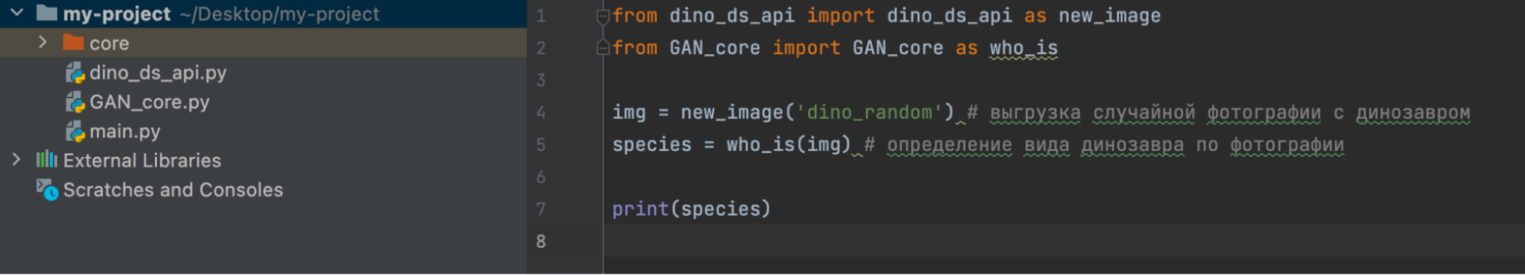
Во всплывающих окнах указывайте настройки на свое усмотрение. Для ознакомления и некоммерческого использования достаточно бесплатного тарифа. Регистрация завершена.

## **СОЗДАНИЕ ПЕРВОГО РЕПОЗИТОРИЯ**

После регистрации и настройки рабочего окружения можно приступить к работе с проектом. Для начала создадим репозиторий на GitHub — облачное пространство, в котором размещаются файлы проекта и его документация. Существует несколько способов.

### **Первый способ — синхронизация с локальным репозиторием**

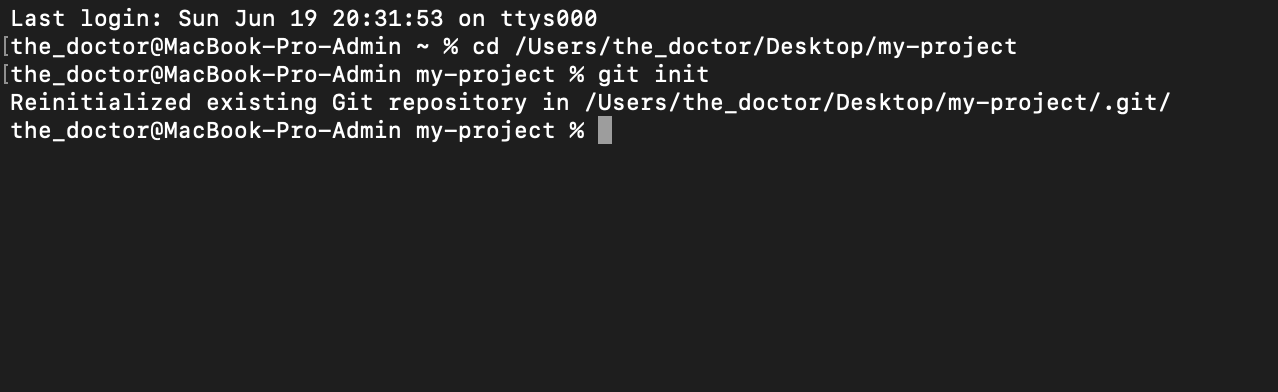
Допустим, нам нужно выложить в открытый доступ код программы Selectel — определитель динозавров по фотографиям.



Перед его загрузкой в глобальный репозиторий можно создать локальный.

Для этого необходимо зайти в терминал, перейти в директорию проекта и ввести команду:

git init



#### **Загрузка файлов в репозиторий. Создание коммитов**

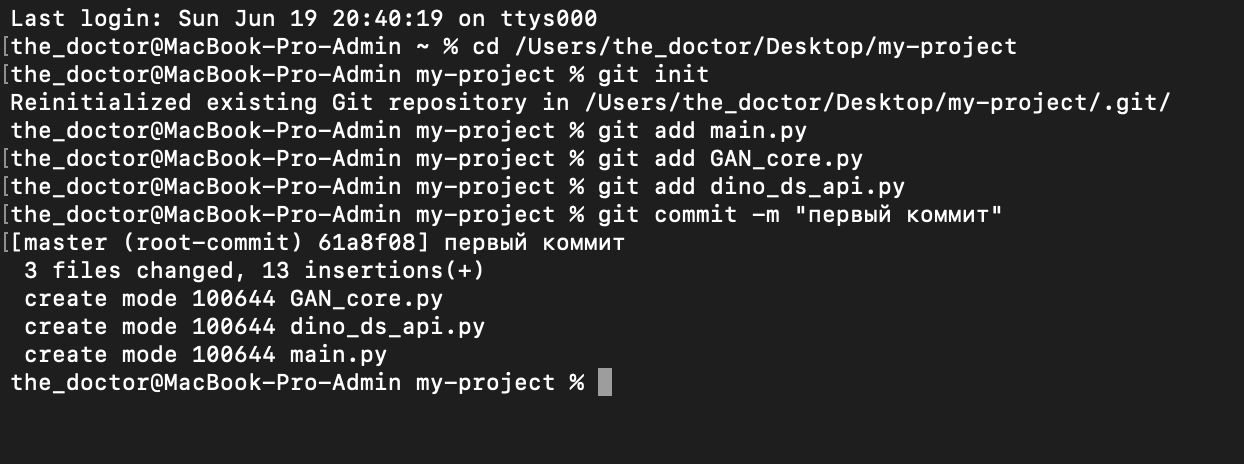
Далее следует добавить все файлы проекта в своеобразный пакет изменений и сделать commit («закоммитить») — загрузить изменения.

git add main.py

git add GAN\_core.py

git add dino\_ds\_api.py

git commit -m “первый коммит”

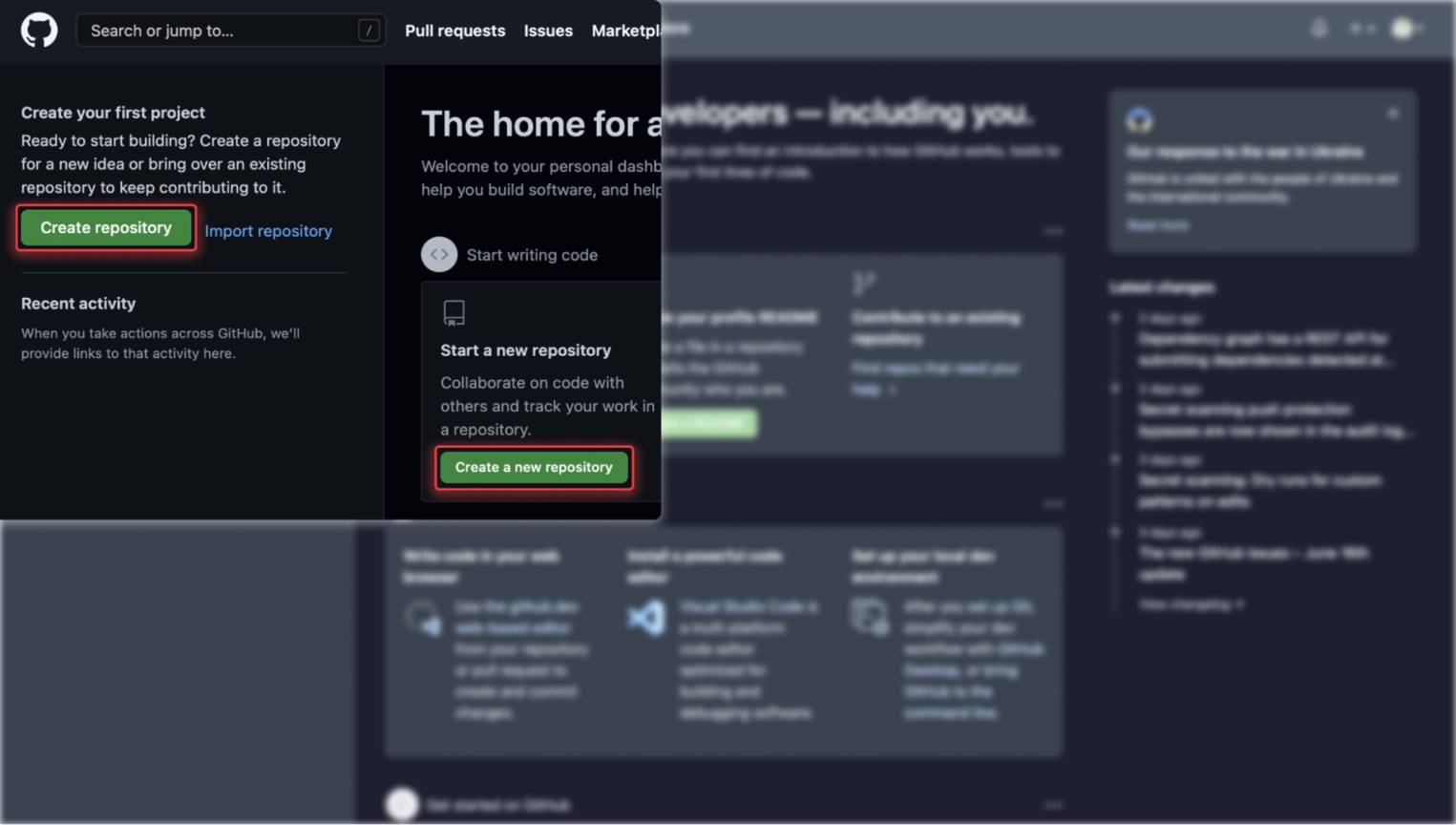


Последняя команда делает сам «коммит», а флаг -m указывает на сообщение «первый коммит».

В примере были «закоммичены» несколько python-файлов: main, GAN\_core и dino\_ds\_api. Если вам нужно добавить в «коммит» все, что есть в директории, — используйте команду:

git add .

Теперь создадим репозиторий на GitHub. Для этого нужно нажать на кнопку **Create repository**.

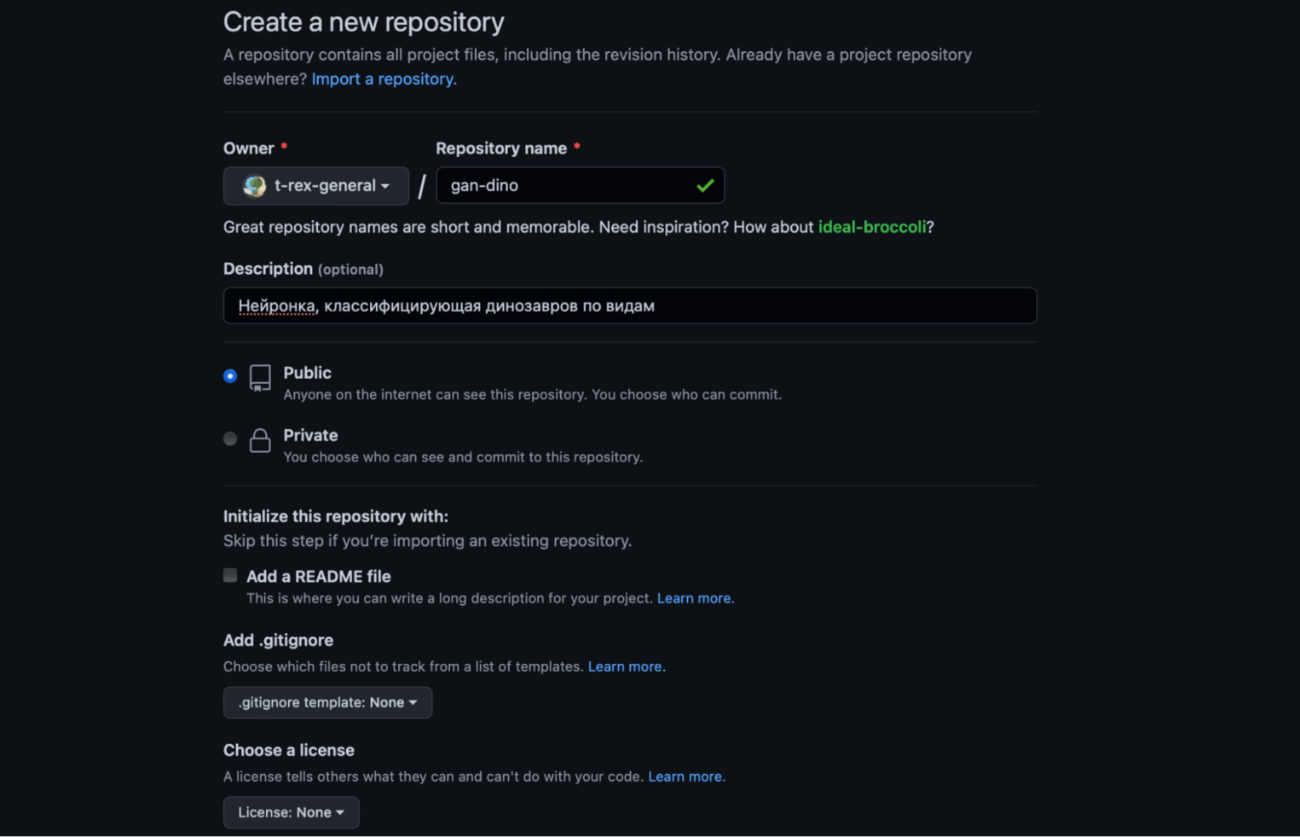


В открывшемся окне обязательно к заполнению только поле с названием проекта. Оно должно быть кратким, но понятным. В нашем примере это gan-dino (gan от [generative adversarial networks](https://habr.com/ru/post/352794/" \t "_blank) и dino от dinosaur).

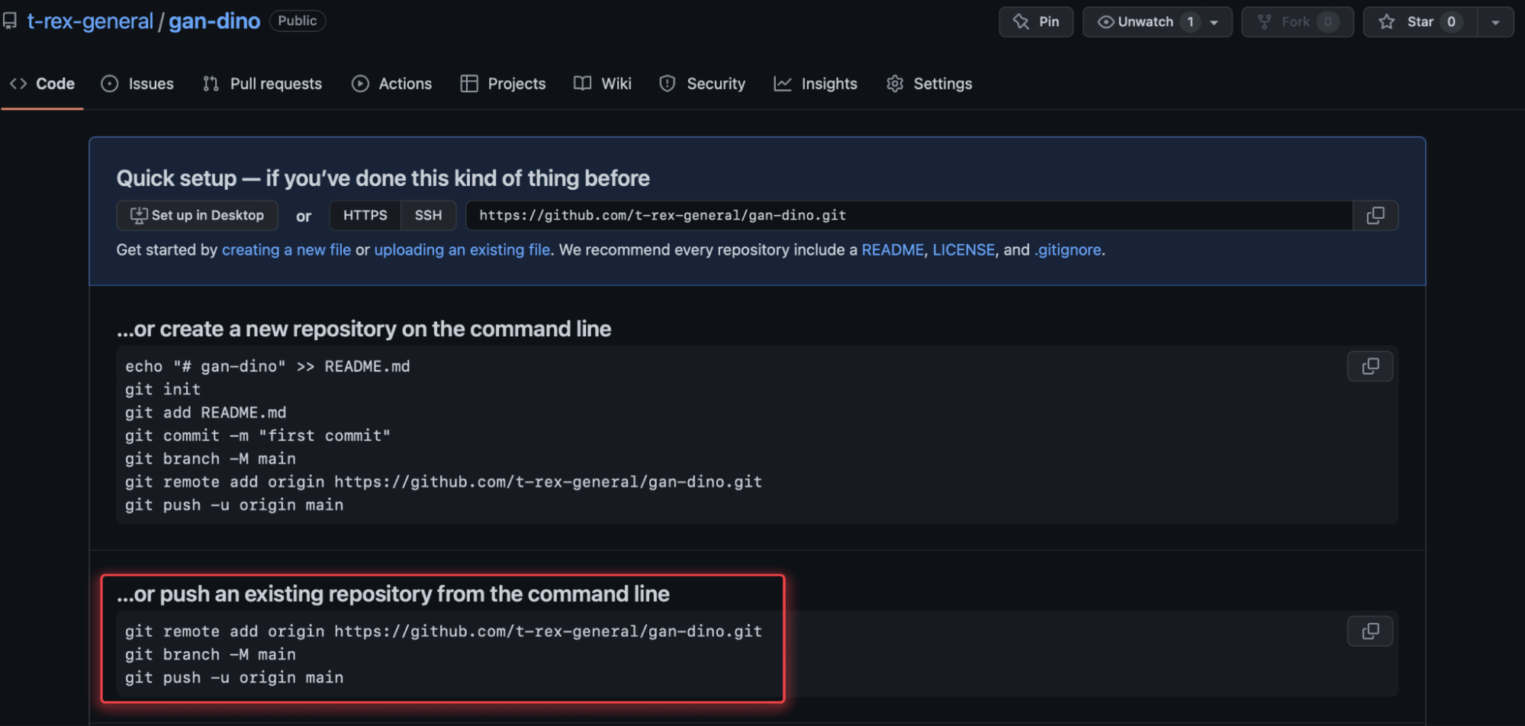
Все остальное опционально:

* **Описание**. Поле с кратким описанием проекта.
* **Режим доступа**. Для коммерческих или корпоративных продуктов обычно устанавливается режим private (репозиторий доступен ограниченному кругу лиц). В остальных случаях — public (доступно по ссылке).
* **Файл README**. Если в репозитории нужно подробное описание проекта — поставьте галочку рядом с **Add a README file**. Но есть нюанс: для первого способа создания репозитория галочки быть не должно.
* **Конфигурация .gitignore**. Бывает, что в проекте нужно разместить невидимые для Git файлы. Чтобы как-то их обозначить, придумали конфигурацию .gitignore, в которой их можно [перечислить](https://git-scm.com/docs/gitignore).
* **Лицензия**. Чтобы никто не использовал ваш код в коммерческих целях без спроса, необходимо добавить файл с лицензией. В нем правообладатели прописывают правила использования своей интеллектуальной собственности.

Создадим public-проект gan-dino, без файла README и конфигурации .gitignore.



Далее GitHub показывает наборы команд, необходимые для загрузки исходного кода в репозиторий. Нас интересует второй блок.



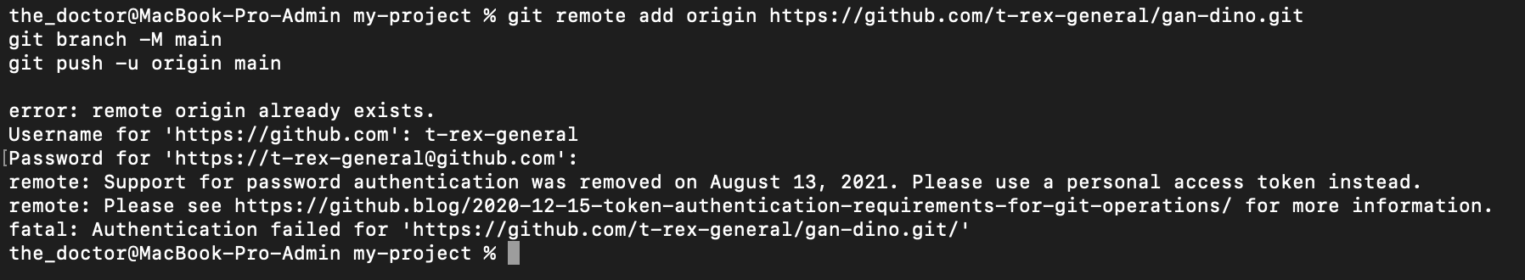
git remote add origin https://github.com/t-rex-general/gan-dino.git

git branch -M main

git push -u origin main

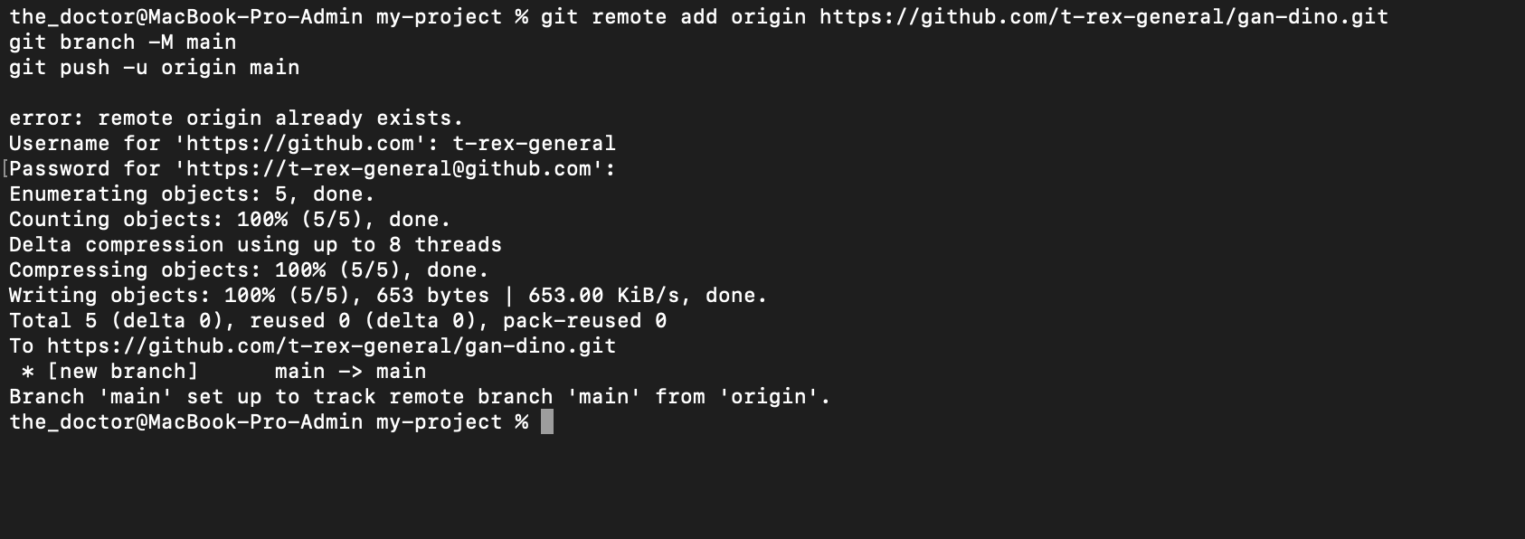
Первая строка загружает origin —  прообраз нашего проекта в глобальном репозитории. Со второй командой мы познакомимся позже. Третья команда загружает (пушит) изменения в GitHub-репозиторий.

После ввода команд система попросит авторизоваться с помощью пароля и названия профиля.

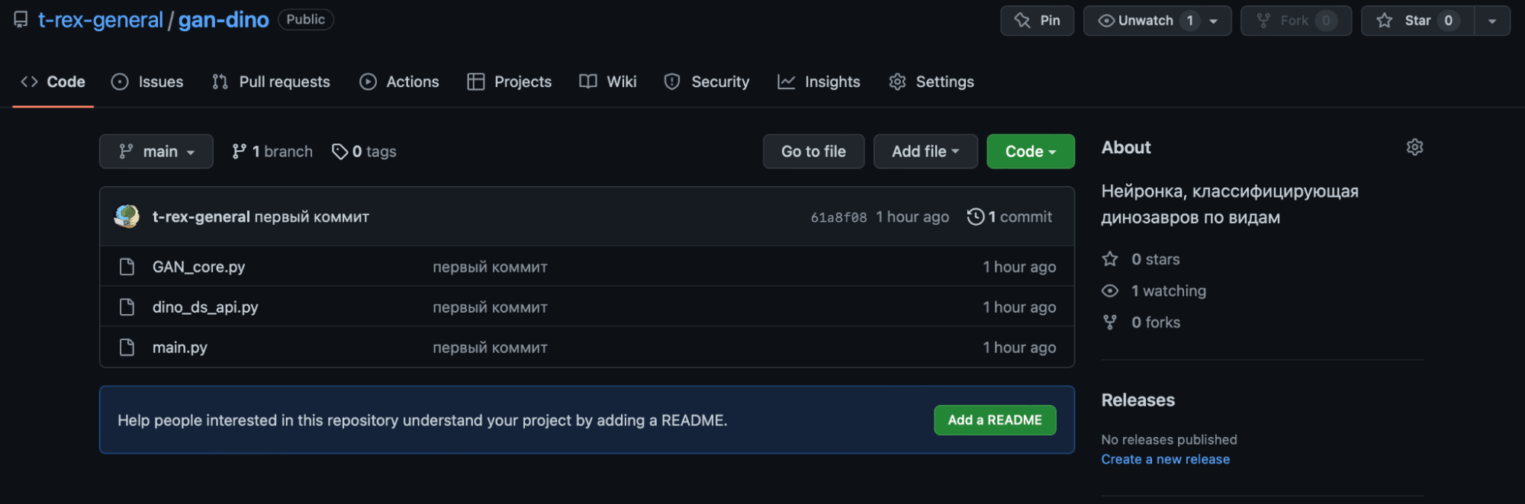


После 13 августа 2021 года вместо пароля нужно вводить [токен](https://github.com/settings/tokens" \t "_blank).

Откройте настройки вашего аккаунта, выберите пункт меню **Developer settings**, кликните по **Personal access tokens** и **generate new token**. А затем повторите попытку.



Получилось! В репозиторий загрузились нужные файлы. Можно смело делиться ссылкой на него.

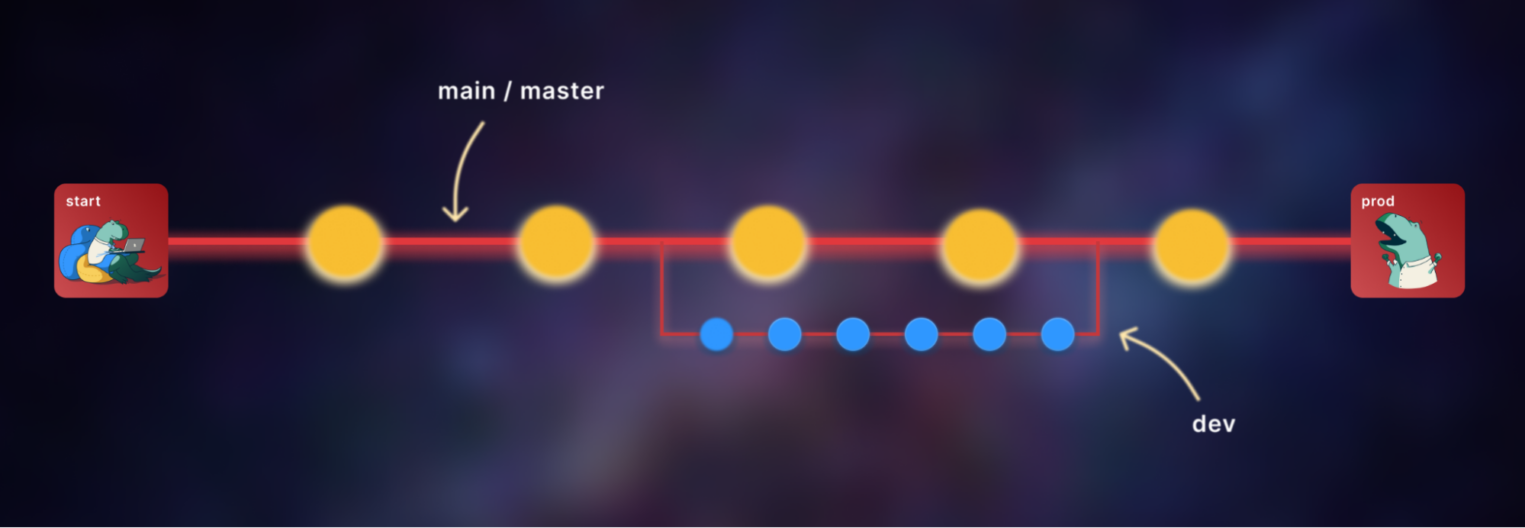


## **РАБОТА С ВЕТКАМИ**

С точки зрения Git, весь процесс разработки — это история коммитов. Такие истории называются ветками — своеобразными указателями на последний коммит.

Представьте ситуацию:

Два программиста работают над одним контроллером для авторизации. Первому нужно написать шифрование пароля по заданному «юзер-токену», а второму —  запрограммировать регистрацию информации в базу данных. Разработчики обнаружили, что у токена не тот тип данных, и решают преобразовать его, используя новую переменную. Они это сделают по-разному. Но ничего страшного: каждый из них работал в своей ветке и заметит конфликт при их слиянии.



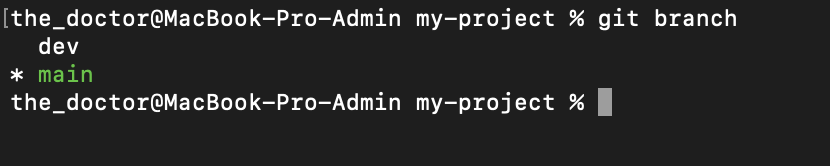
### **Создание веток через Git**

Чтобы создать ветку (например, dev) в проекте, нужно ввести команду:

git branch dev

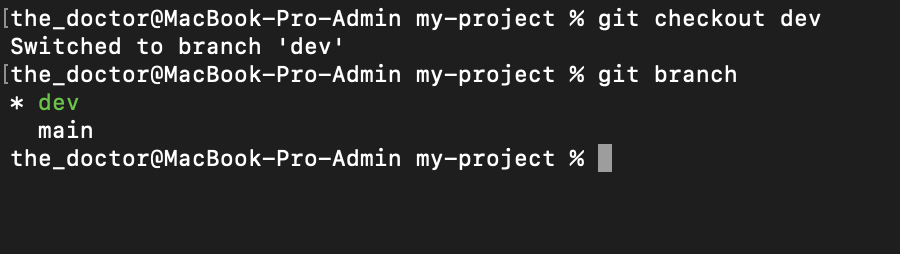
После ветка появится в общем списке.

git branch

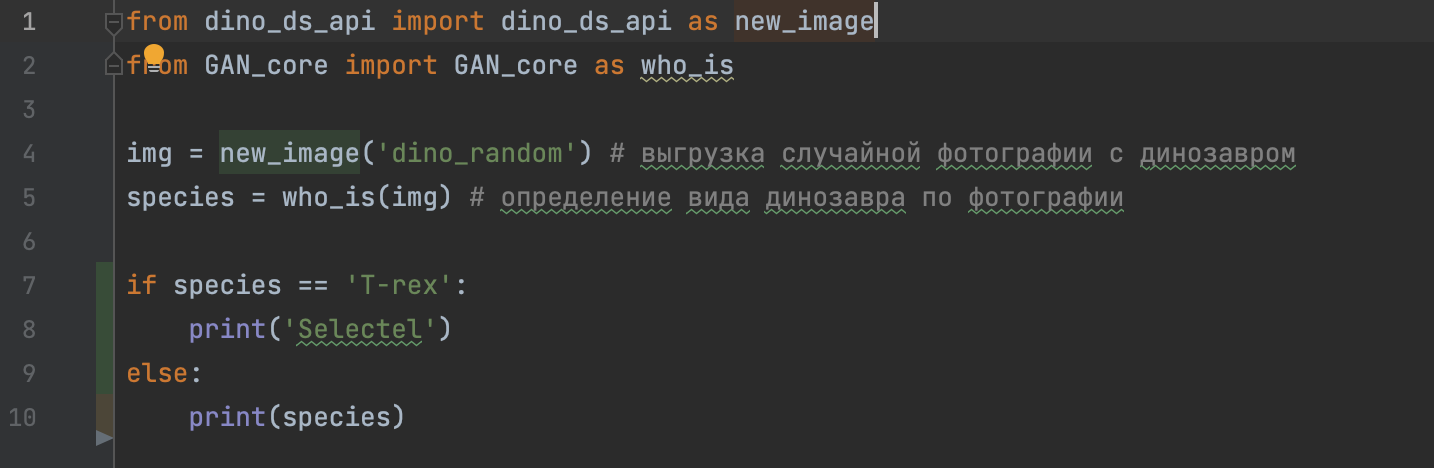


Видно, что выбрана ветка main, то есть все коммиты загружаются в нее. Чтобы работать с веткой dev, нужно переключиться.

git checkout dev



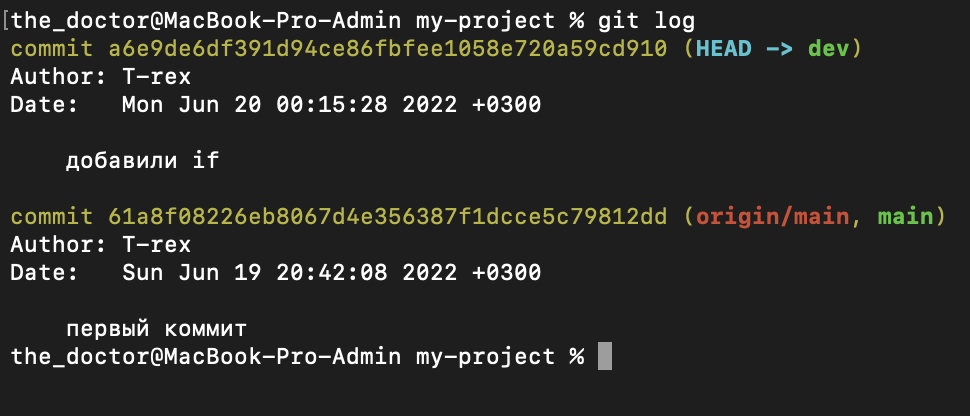
Попробуем изменить файл проекта и загрузить коммит.

Добавили в программу конструкцию if

git add main.py

git commit -m “добавили if”

Теперь можно посмотреть логи — историю добавления коммитов.



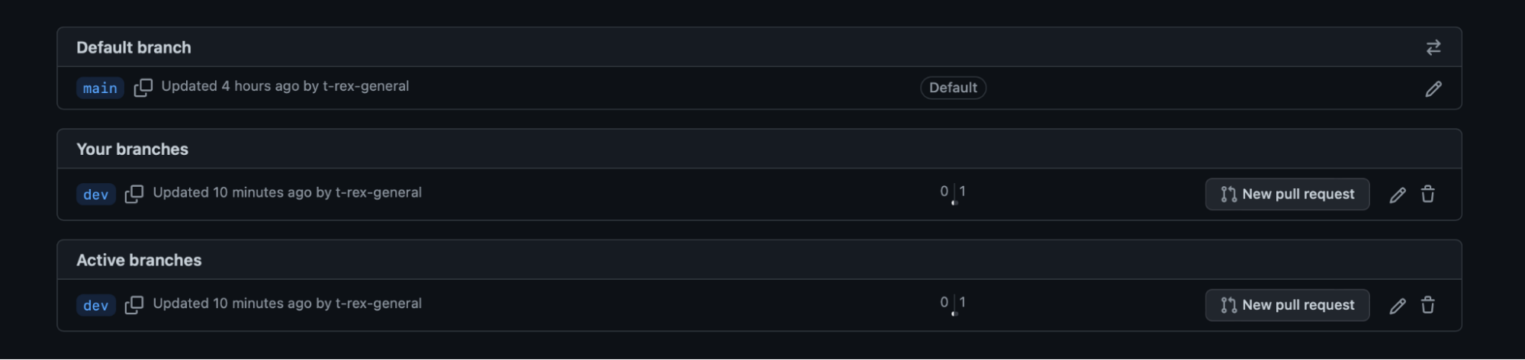
Действительно, второй коммит «улетел» в ветку dev. Если нас не устроили изменения, можно откатиться до предыдущего (любого) коммита по его номеру.

git checkout 61a8f08226eb8067d4e356387f1dcce5c79812dd

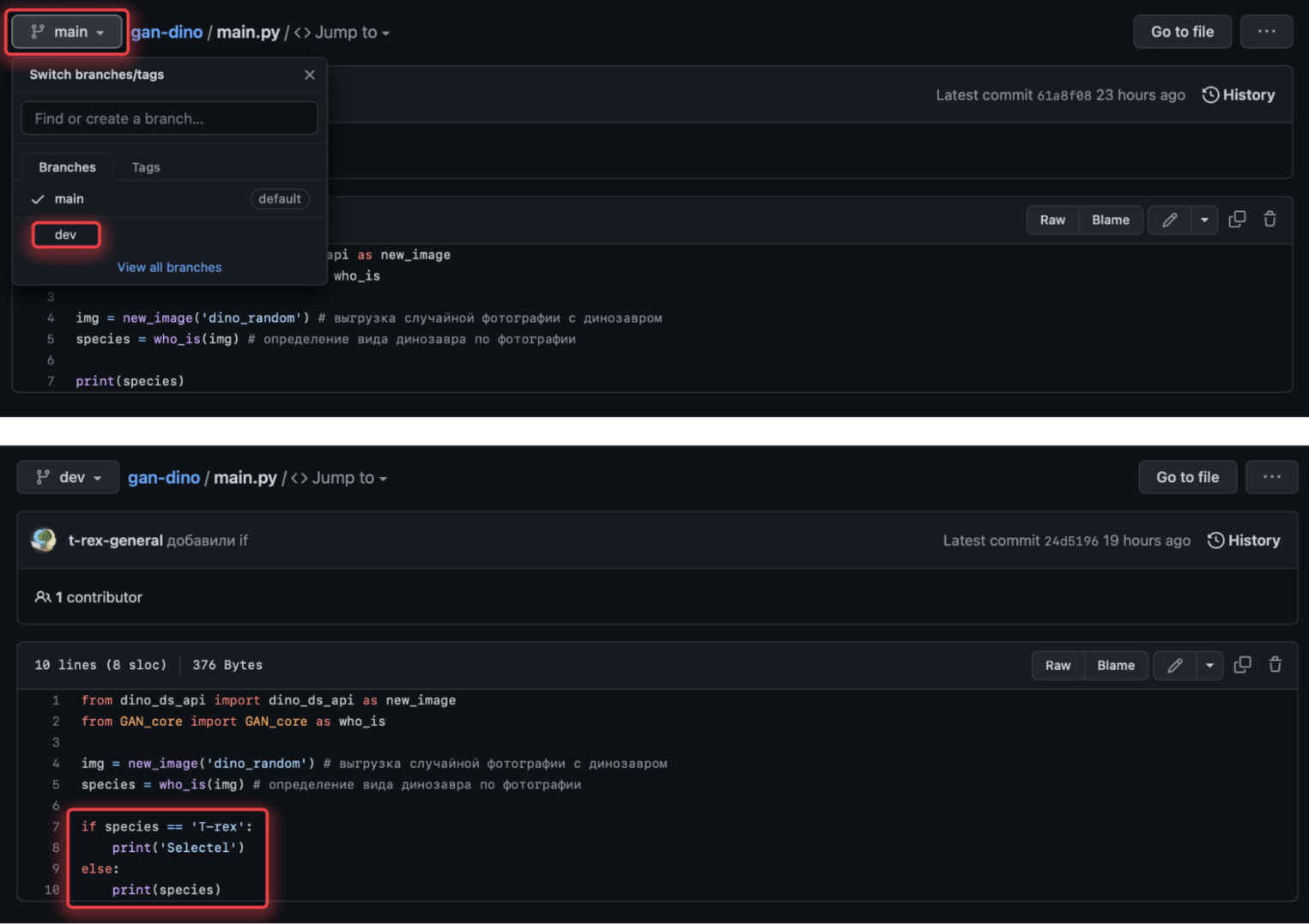
Чтобы запушить ветку в онлайн-репозиторий введем команду:

git push --set-upstream origin dev

Открываем репозиторий в GitHub и видим, что добавилась ветка dev:



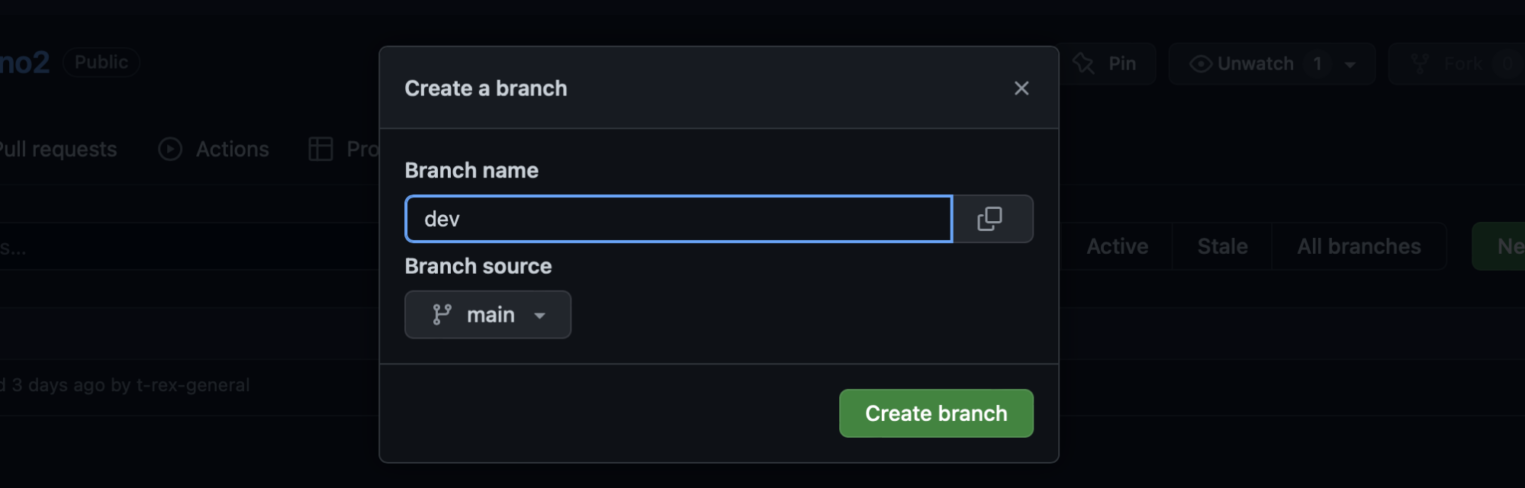
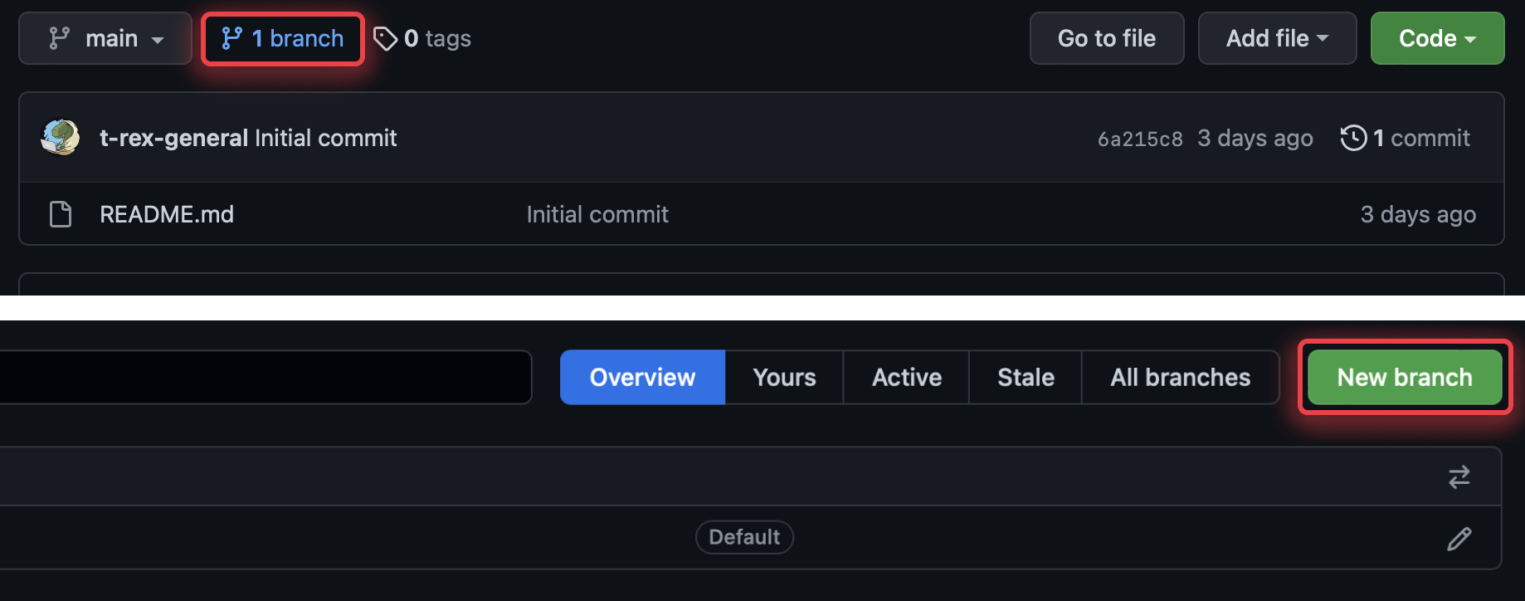
Но если мы зайдем в main.py, то никаких изменений там не обнаружим, пока не выберем ветку dev.



Чтобы изменения затронули и main-ветку, нужно сделать merge — слияние веток.

### **Создание веток через GitHub**

Как и в случае создания репозитория, можно быстро создавать новую ветвь в GitHub и переключаться между существующими ветками.



В рамках веток можно также вносить изменения — механизм работы не меняется.

### **Слияние веток проекта**

Мы почти разработали свой проект. Самое время объединить ветки dev и main.

Первым шагом необходимо переместиться в ветку main:

git checkout main

Вторым шагом — сделать merge с веткой dev и запушить изменения:

git merge dev

git push

Теперь в GitHub-репозитории отображается актуальная информация.