Основы работы с данными для ИИ

Лекция 1
Основы работы с Git и GitHub

Доц
Екатерина Александровна
Преподаватель кафедры информатики и вычислительного эксперимента

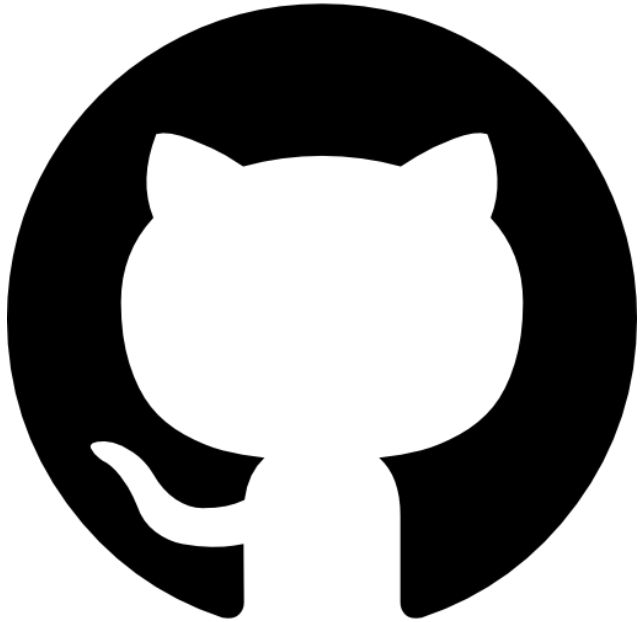
Что такое GitHub?

GitHub — это веб-сервис для хранения Git-репозиториев в облаке и совместной разработки.

Помимо функций Git, GitHub предоставляет инструменты для управления проектами

Возможности GitHub:

- Удалённое хранение репозиториев
- Pull Requests— запрос на внесение изменений в проект
- Issues — система управления задачами и ошибками
- Actions — автоматизация процессов (CI/CD)
- Wiki — документация проекта
- Fork — копирование чужого репозитория для работы над ним
- Stars — отметка «понравилось» для репозитория



Основные определения GitHub

GitHub — это облачная платформа, которая предоставляет удалённое хранение Git-репозиториев, а также инструменты для совместной разработки и управления проектами

Ключевые термины в GitHub:

Repository (репозиторий) — проект, хранящийся в GitHub (код, документация, история изменений)

Fork — копия чужого репозитория в вашем аккаунте для независимой работы.

Pull Request (PR)— запрос на внесение изменений из одной ветки (или форка) в другую

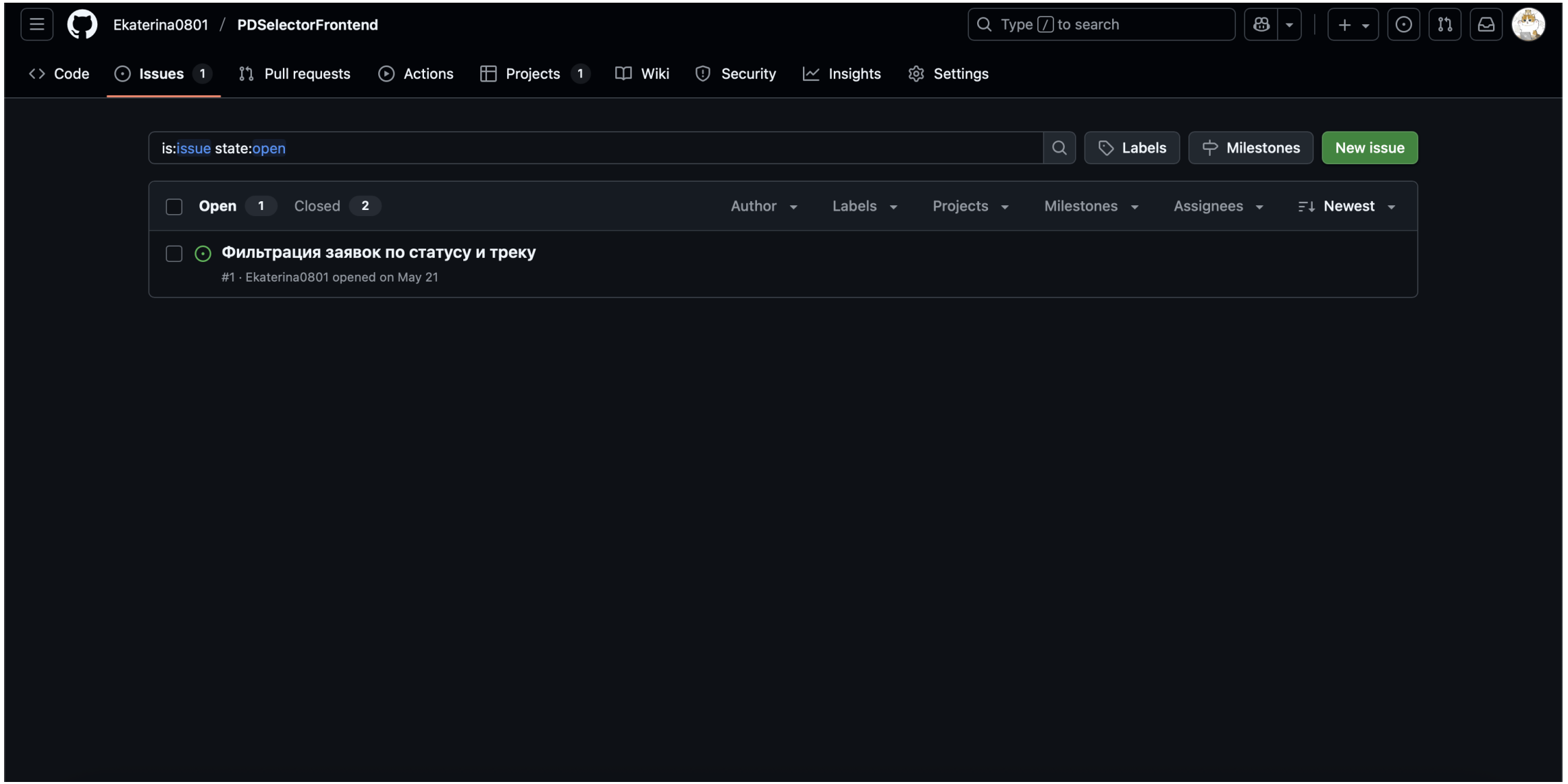
Issues — система отслеживания задач и ошибок

Actions — автоматизация процессов (CI/CD) для тестирования, сборки и деплоя проекта

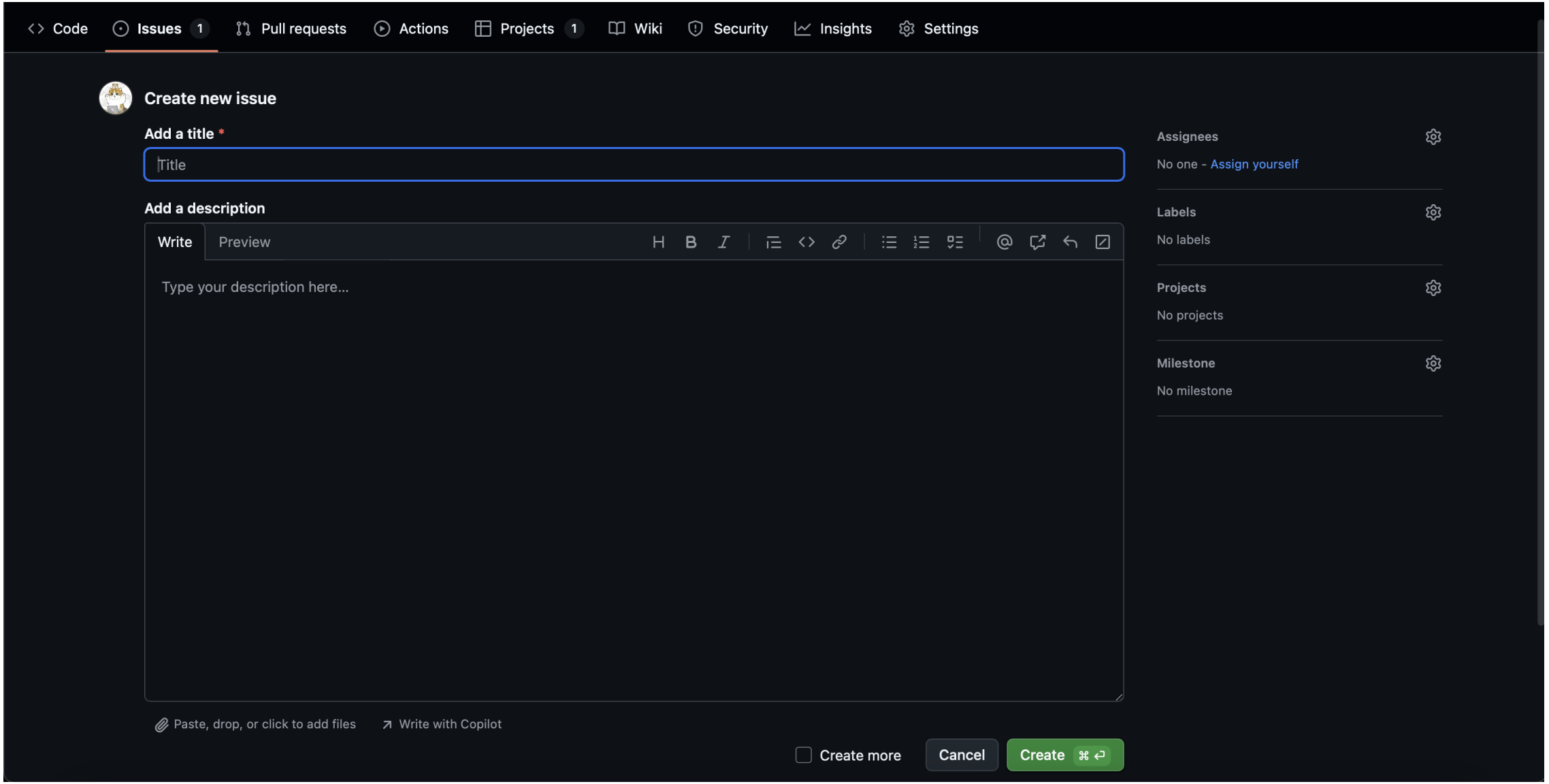
Wiki — встроенная документация проекта

Stars — «лайки» для репозитория, чтобы отметить его как полезный или интересный

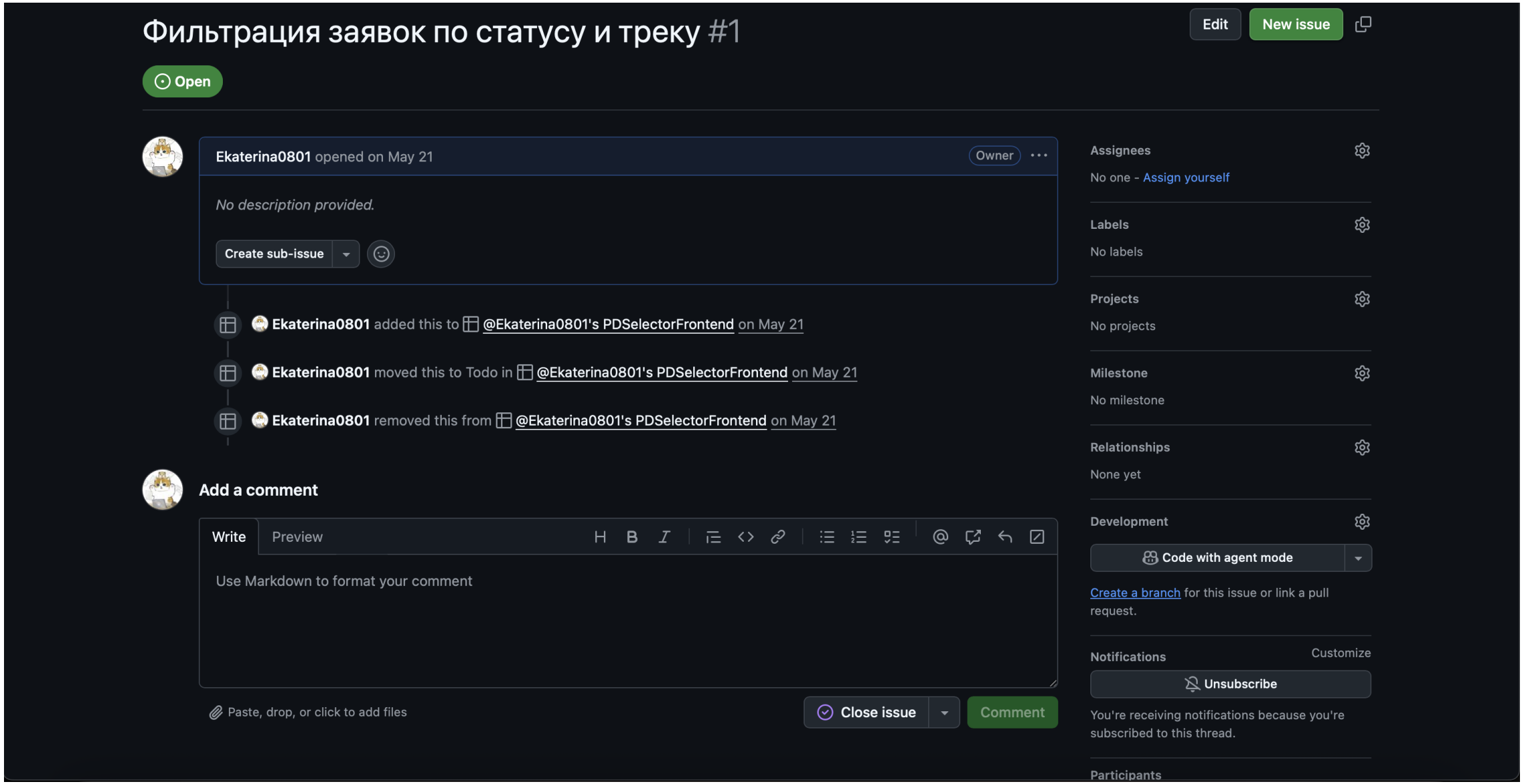
Работа с Issues



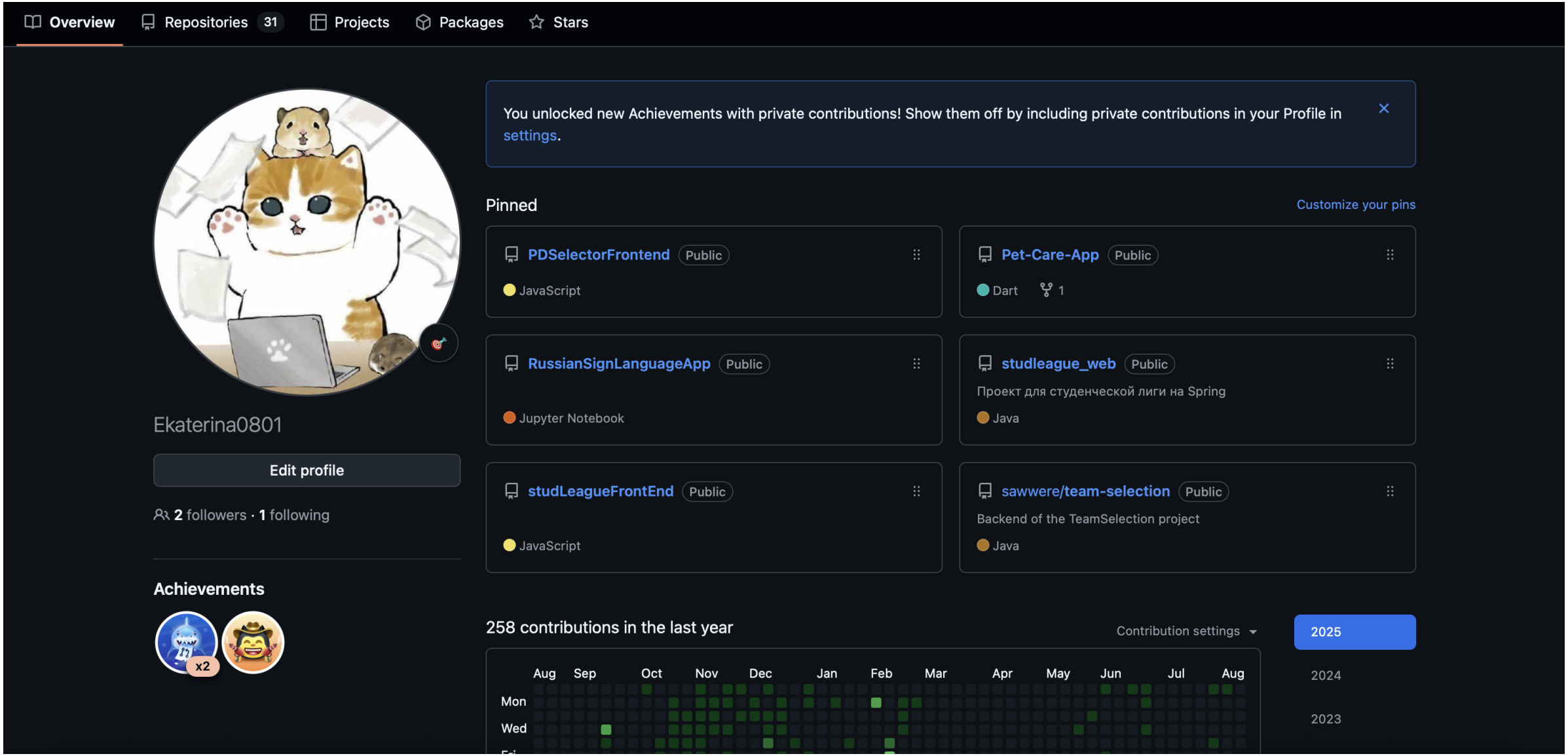
Работа с Issues



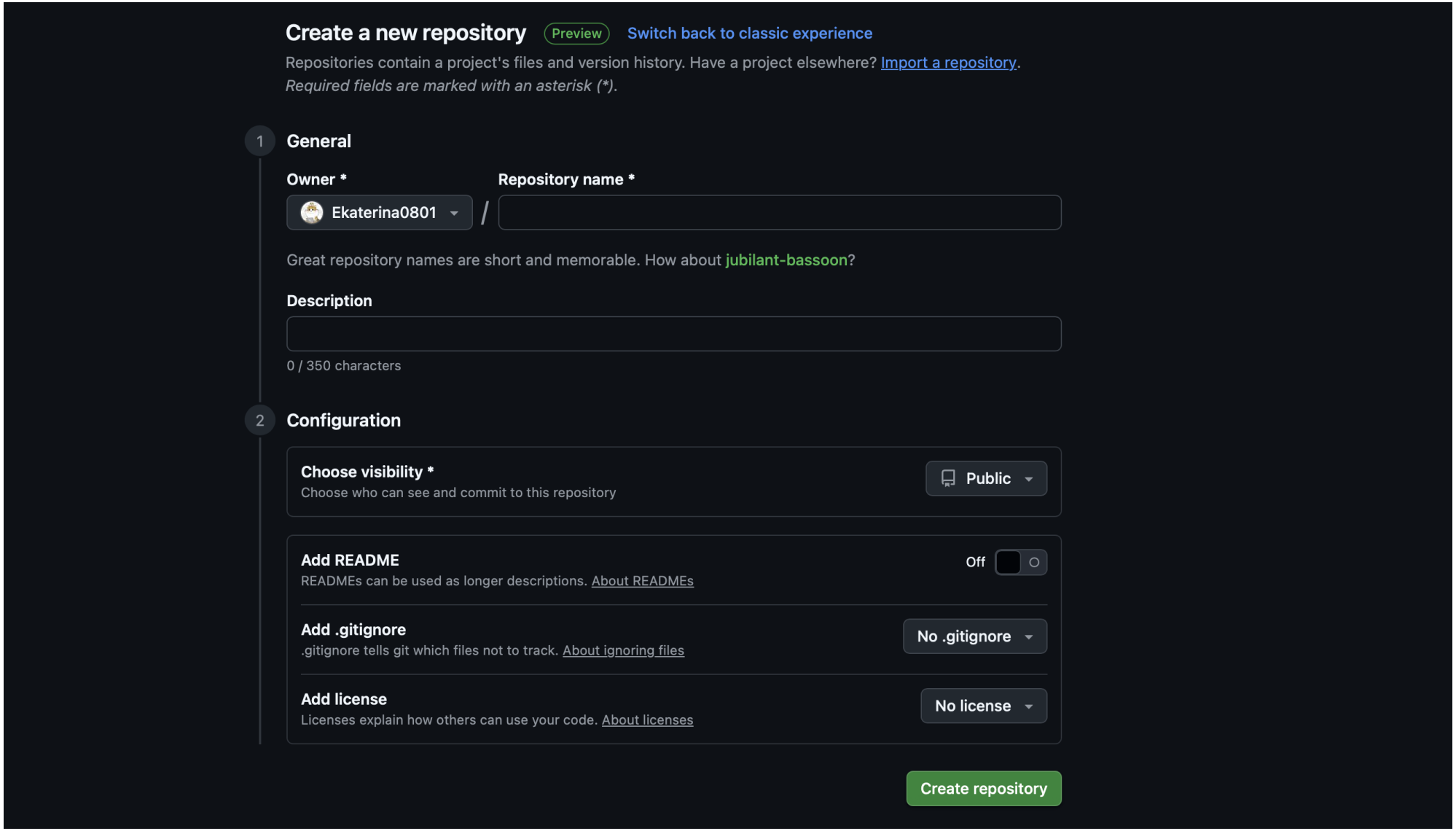
Работа с Issues



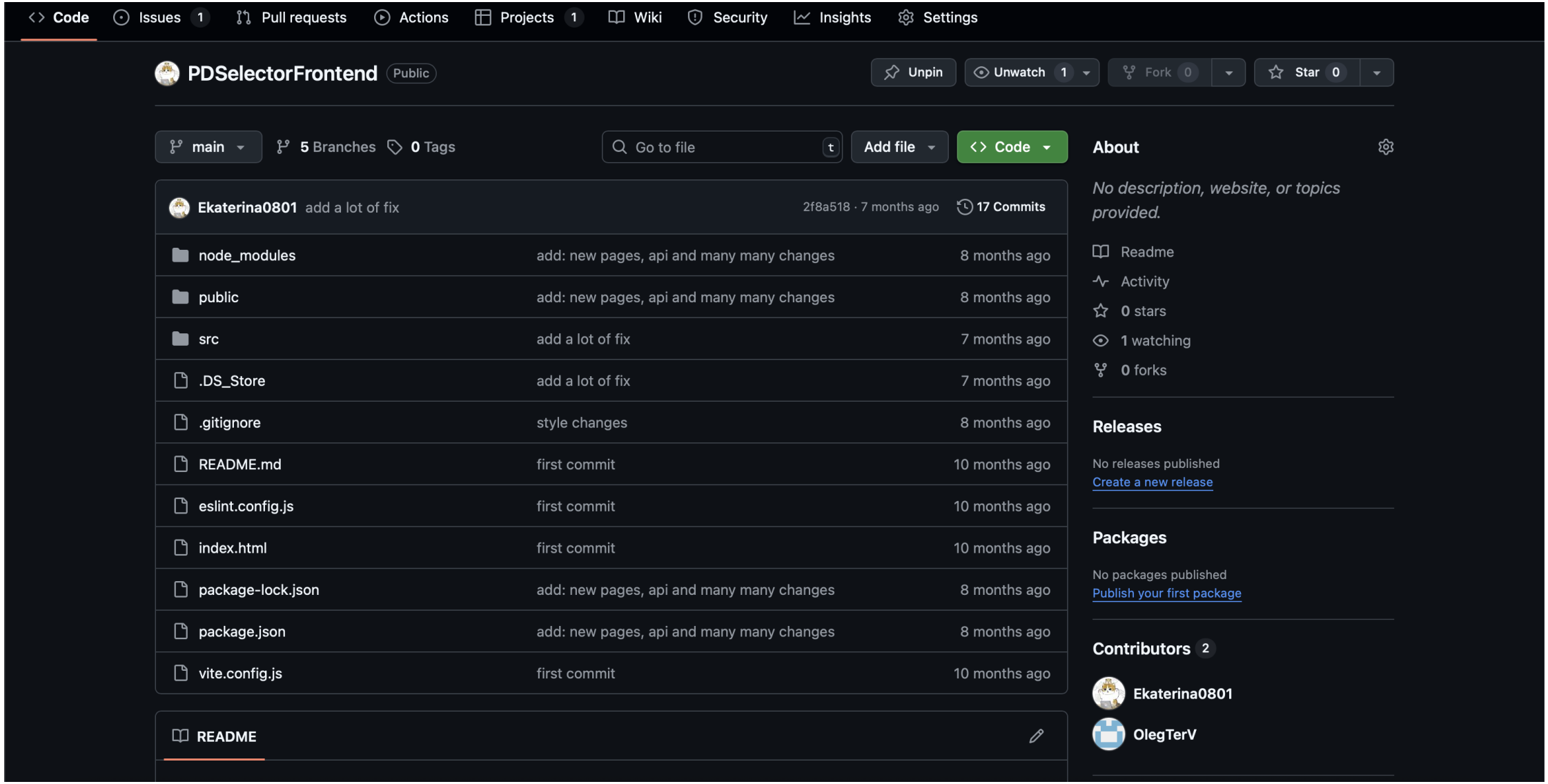
Аккаунт на GitHub



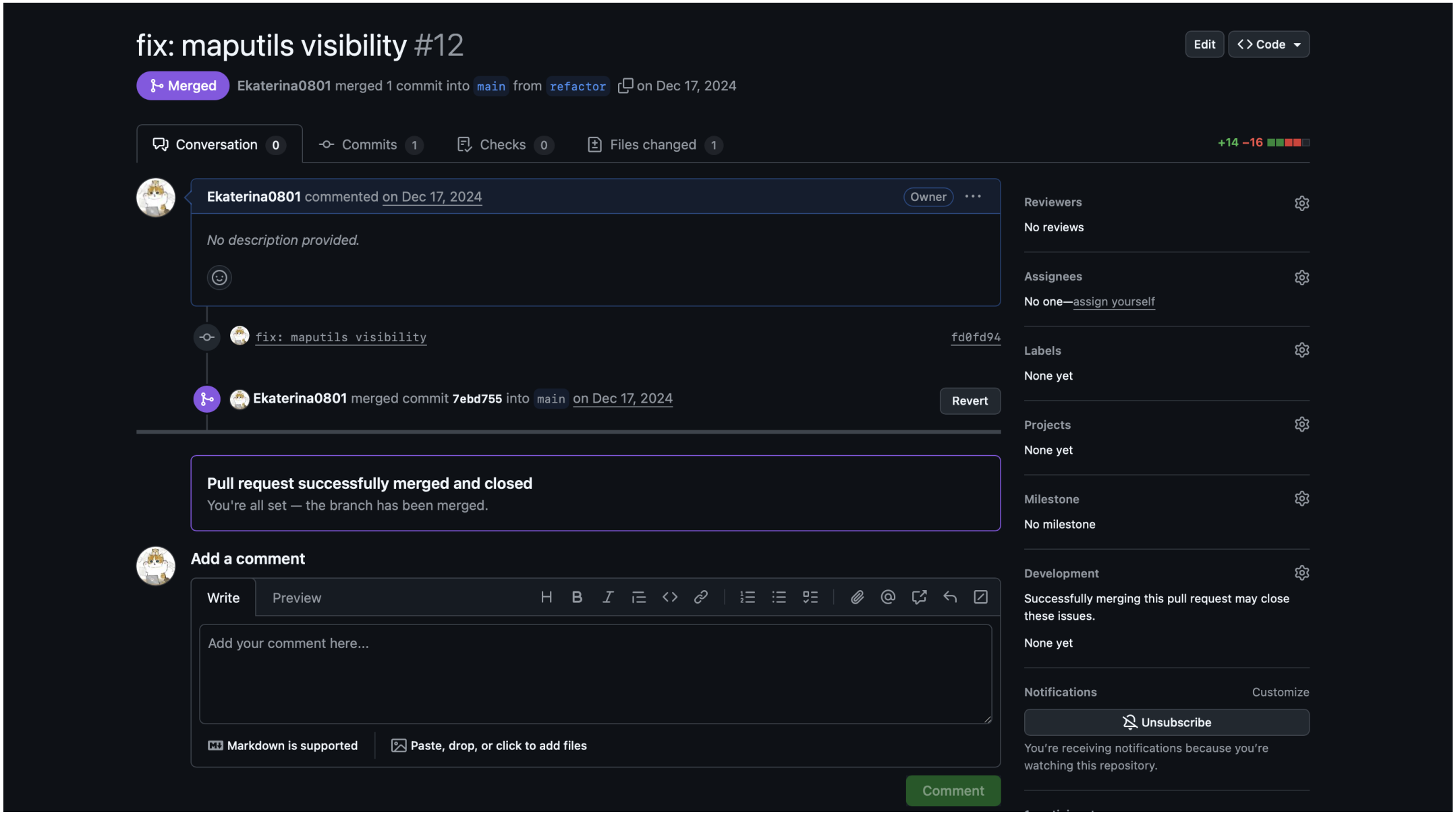
Создание репозитория GitHub



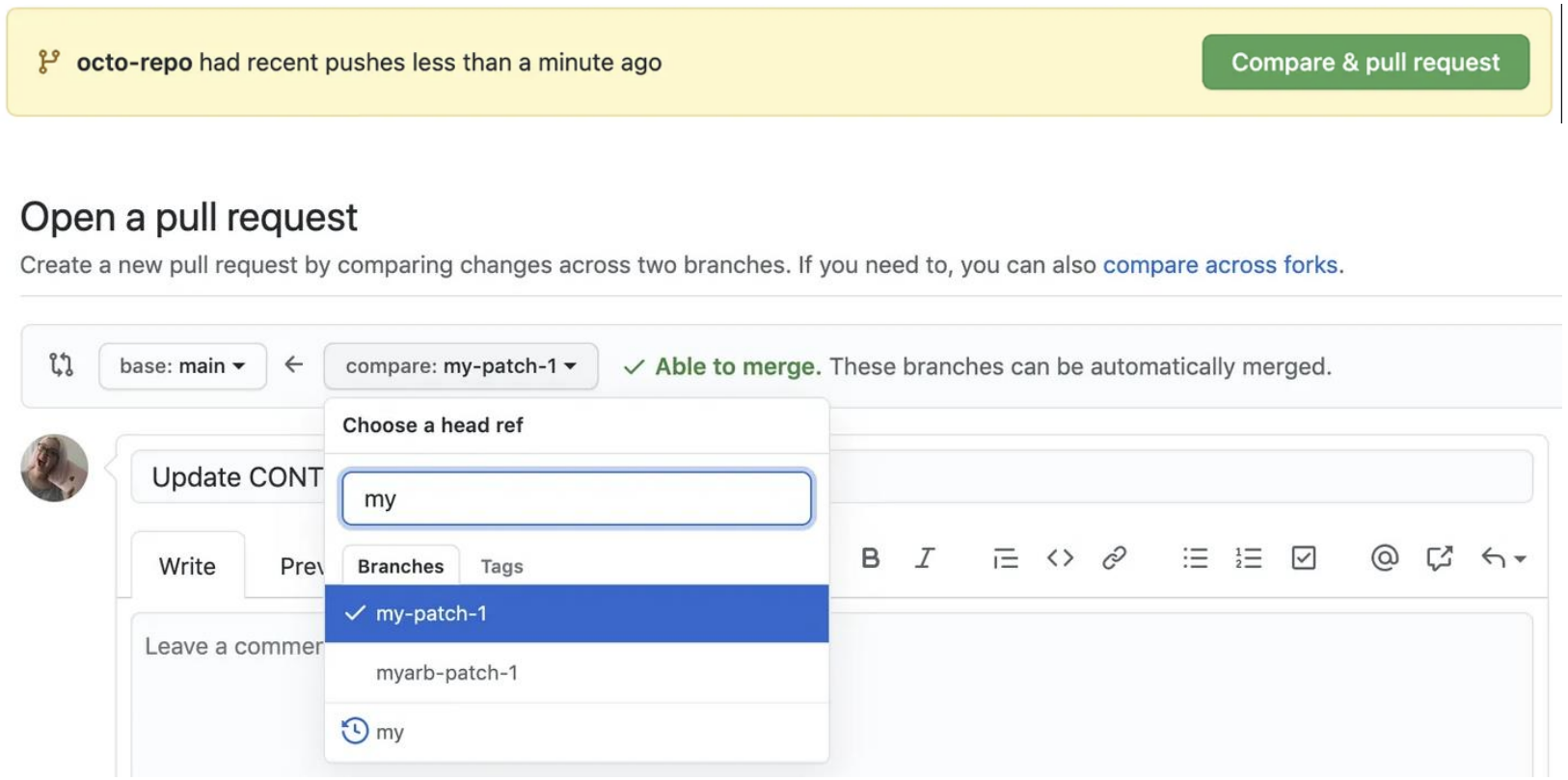
Просмотр файлов в репозитории



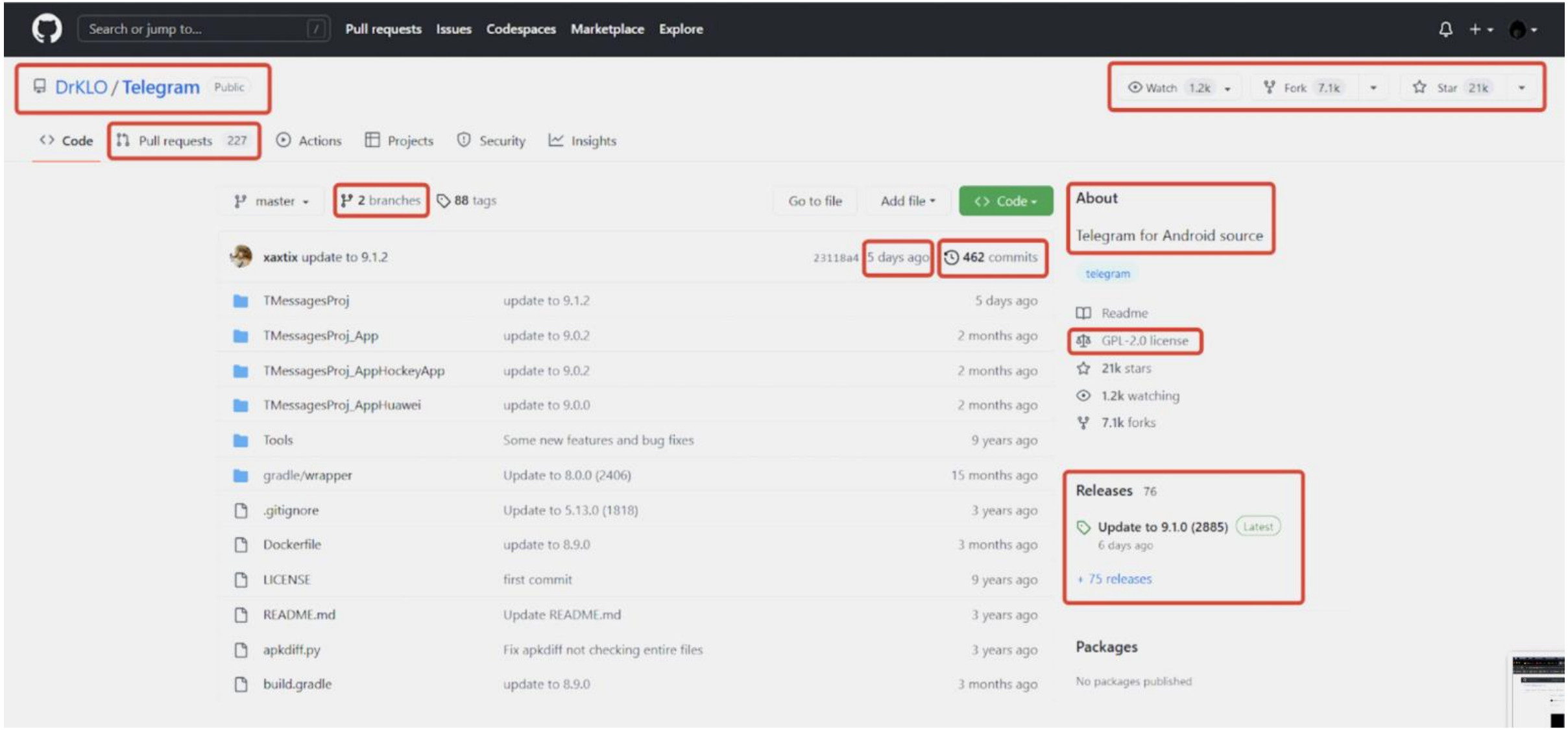
Пулл-реквесты



Пулл-реквесты



Поиск в репозитории



Токены GitHub

Токены на GitHub — общее понятие

Это общий термин для ключей доступа, которые выдает GitHub. Они используются для **аутентификации** (подтверждения вашей личности) при работе с API GitHub или с репозиториями через Git (например, при пуше или клоне по HTTPS)

Токены GitHub

Personal Access Tokens (PAT) — Классические токены:

Появились раньше и до сих пор широко используются.

Их разрешения привязаны к **scope** (областям) — широким группам прав, например:

- **repo** (полный доступ к репозиториям)
- **admin:org** (доступ к управлению организацией)
- **workflow** (разрешение на управление GitHub Actions).

Минусы: Часто дают избыточные права. Например, токен со scope repo получает полный доступ ко всем вашим репозиториям (публичным и приватным). Это серьезный риск безопасности, если токен утечет

Токены GitHub

Fine-grained Personal Access Tokens — Детализированные токены:

Это новый, более безопасный и современный тип токенов, который GitHub активно продвигает.

Их ключевая особенность — **granular permissions** (детализированные, "мелкозернистые" разрешения).

Токены для установки GitHub Apps:

Используются самими GitHub Apps (интеграциями) для доступа к ресурсам от своего имени.

Токены для Actions (GITHUB_TOKEN):

Специальный токен, который автоматически создается для каждого запуска workflow в GitHub Actions. Он имеет ограниченные права только в рамках того репозитория, где запускается workflow.

Токены GitHub

Какой использовать?

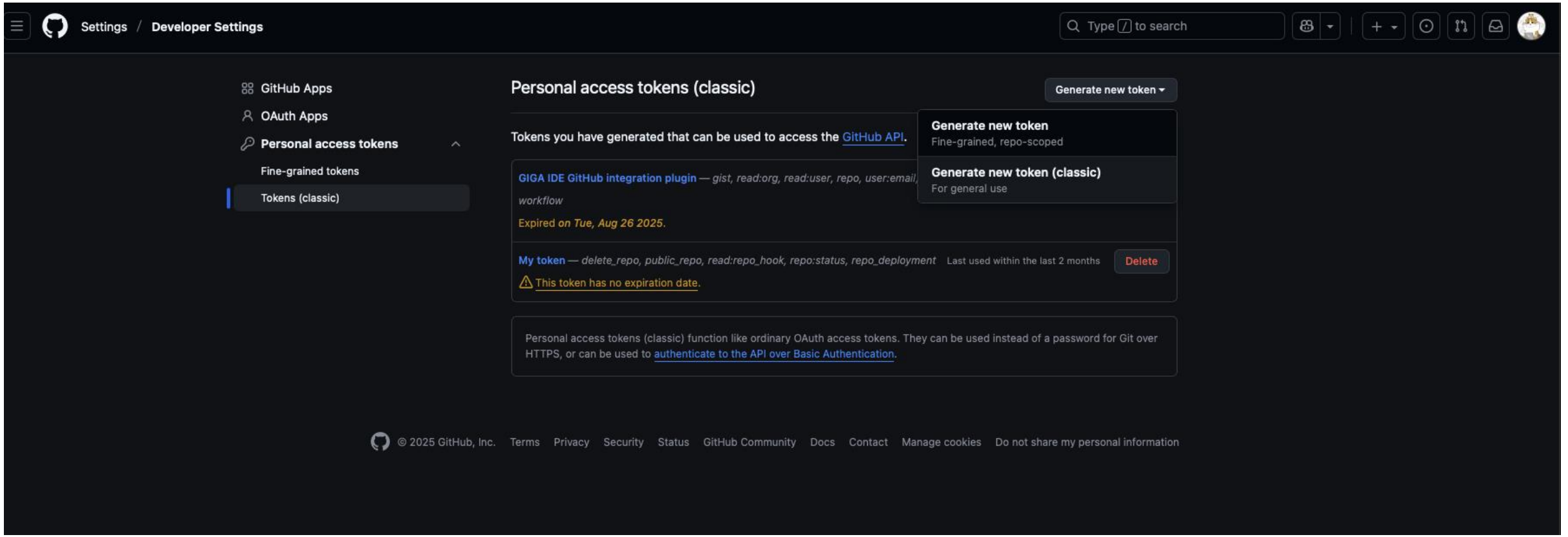
Всегда отдавайте предпочтение Fine-grained tokens. Это современный и безопасный стандарт. GitHub рекомендует использовать именно их

Классические PAT (Personal Access Tokens) стоит использовать только для интеграций со старыми системами, которые еще не поддерживают новый формат fine-grained tokens, или если вам нужны специфические scopes, которых пока нет в новой системе (хотя их список постоянно расширяется

GITHUB_TOKEN — используйте по умолчанию в GitHub Actions, так как он создается автоматически и безопасен по своей природе

Токены GitHub

На Github персональный токен создаётся в разделе Settings - Developer settings - Personal access tokens - Tokens



Авторизация через токен

Созданный персональный токен используется в момент аутентификации для HTTPS репозитория:

```
git clone https://github.com/username/repo.git Username: имя_пользователя
Password: персональный_токен
```

Или в момент авторизации на Github API, например:

```
curl -u имя_пользователя: персональный_токен https://api.github.com/user
```

Что такое Git?

Git — это распределённая система контроля версий, созданная Линусом Торвальдсом в 2005 году

Её основная задача — отслеживать изменения в проекте и обеспечивать совместную работу разработчиков

Особенности Git:

- **Распределённость** — каждый разработчик имеет полную копию репозитория с историей изменений
- **История изменений (commit history)** — хранит все версии проекта
- **Ветки (branches)**— позволяют параллельно работать над разными функциями
- **Слияния (merge)**— объединяют изменения из разных веток
- **Гибкость** — поддержка разных рабочих процессов (workflow)

Git и GitHub - не одно и то же

С помощью Git программисты и разработчики ориентируются в коде и отслеживают изменения. Git помогает **вернуть файлы в исходное состояние и видеть изменения**, внесённые в определённый период. Разработчик выполняет разные команды (например, commit, push), а все изменения синхронизируются с центральным репозиторием.

Git — это система контроля версий, а **GitHub — онлайн-сервис**, по сути социальная сеть. Одна из основных целей GitHub — **быть единым местом для проектов с исходным кодом.** Предполагается, что пользователь делится чем-то полезным, а другие люди смогут участвовать в разработке.

Ещё один вариант — использовать GitHub как хранилище проектов для портфолио: легко дать на них ссылку

Связь Git и GitHub

Git управляет версиями проекта **локально**

GitHub хранит репозиторий **удалённо** и упрощает командную работу

Рабочий процесс:

- **Clone** — клонируем удалённый репозиторий
- **Commit** — фиксируем изменения локально
- **Push** — отправляем изменения на GitHub
- **Pull Request**— предлагаем изменения в общий проект
- **Merge** — объединяем изменения в главную ветку

Преимущества использования Git и GitHub

- Безопасность данных — вся история хранится у каждого участника
- Совместная работа над одним проектом
- Возможность отката изменений
- Автоматизация тестов и деплоя
- Удобная визуализация истории изменений

Основные определения

Репозиторий – это просто **корневая папка с файлами и вложенными директориями вашей программы —** и одновременно **её страница на GitHub**. Загрузить в репозиторий можно всё что угодно, но предполагается, что вы будете хранить в нём файлы с исходным кодом и какие-нибудь дополнительные материалы — допустим, необходимую для GUI или вёрстки графику (картинки, иконки и тому подобное).

Репозитории могут быть **публичными** и **приватными**, в них можно создавать другие папки и отслеживать изменения версий. Управлять своими репозиториями можно прямо через интерфейс сайта, командную строку, десктопное приложение GitHub или различные средства разработки (IDE), поддерживающие интеграцию с сервисом

Основные определения Git

Repository (репозиторий)— хранилище проекта с историей изменений.

Commit (коммит)— зафиксированное изменение в проекте с описанием (сообщением).

Branch (ветка)— отдельная линия разработки, которая позволяет работать над новыми функциями без изменения основной версии.

Merge (слияние)— объединение изменений из одной ветки в другую.

Clone (клонирование)— создание локальной копии удалённого репозитория.

Pull — получение и объединение изменений с удалённого репозитория в локальный

Push — отправка локальных изменений в удалённый репозиторий

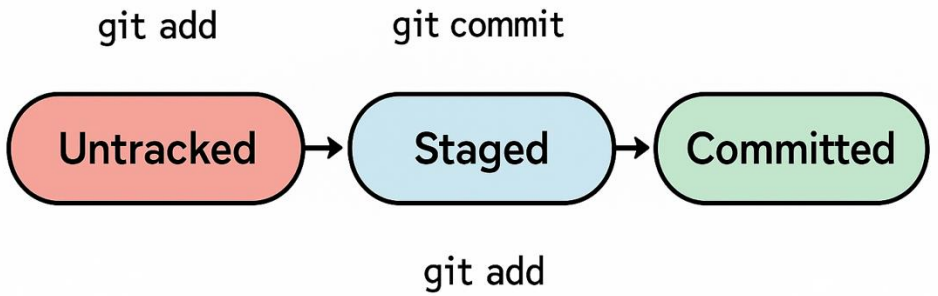
Remote — ссылка на удалённый репозиторий

Жизненный цикл файла в Git

Untracked — файл не отслеживается

Staged — файл добавлен в индекс (git add)

Committed — файл сохранён в истории (git commit)



Работа с Git через консоль. Установка

Windows

Перейти по ссылке и скачать:
https://git-scm.com/downloads/win

Mac OS

```
brew install git
```

Linux
Открываем терминал и вводим следующую команду:

```
# Debian или Ubuntu
sudo apt install git

# CentOS
sudo yum install git
```

Настройка и создание репозитория

```
$git config --globaluser.name"<ваше_имя>"
```
задание имени

```
$git config --globaluser.email"<адрес_почты@email.com>»
```
задание email

#Для Linux и MacOS путь может выглядеть так /Users/UserName/Desktop/MyProject
#Для Windows например C://MyProject

```
$  cd <путь_к_вашему_проекту>
```
переход в папку

```
$  gitinit
```
инициализация репозитория локально

```
$ git clone https://github.com/repo
```
клонировать репозиторий

```
$ git remote add origin <https://github.com/repo>
```
подключиться к внешнему репозиторию

Ветки в Git

Ветка — это просто указатель (ссылка) на коммит. Переключение — смена «куда смотрит» HEAD

`$ git branch` узнать текущую ветку

`$ git switch <branch>` переключиться на другую ветку

`$ git switch -c <new-branch>` создать новую ветку и переключиться на нее

`$ git checkout <branch>` переключение на другую ветку

`$ git merge <branch>` сделать слияние в текущую ветку (создаётся новый merge-коммит с двумя родителями, история сохраняет разветвления)

`$ git rebase <target>` сделать коммиты поверх ветки <target>

`$ git branch -d awesome_new_feature` удаление ветки

Коммиты в Git

Коммит – сохранение состояния проекта в определенный момент времени

`$ git add .` добавление всех файлов в коммит

`$ git add –all` тоже добавление всех файлов в коммит

`$ git add <имя_файла>` добавление конкретного файла в коммит

`$ git commit -m"<комментарий>"` создание коммита с комментарием

`$ git reset css/style.css` удаление файла из коммита

`$ git log` просмотр истории коммитов

`$ git checkout <hash коммита>hello.txt` возврат состояния файла на состояние из коммита

Коммиты в Git

Решение конфликтов

```
$ # правим файлы, убираем <<<<<<< =======
>>>>>>>
$ git add <исправленный_файл>
$ git merge --continue        # если был merge
# или
$ git rebase --continue       # если был rebase
```

Отправка изменений и получение изменений с сервера

`$ git push origin <branch>` отправка (пуш) изменений на сервер

`$ git pull origin <branch>` получение изменений с сервера

`$ git fetch origin <branch>` добавление конкретного файла в коммит

`$ git revert HEAD` отмена последнего коммита

`$ git revert b10cc123` отмена изменений до конкретного коммита (могут быть конфликты)

Основные конвенции для коммитов

Предлагаемый формат коммитов

type(scope): subject

Где:

- Type — тип изменения, fix/feat/build/etc
- Scope — область, которую затронуло изменения, либо номер заявки. Если несколько, писать через запятую. Если вне заявок, пропустить
- Subject — краткое описание проделанной работы. С заглавной буквы. Описание коммитов должно отвечать на вопрос: «Что делает коммит?»

Основные конвенции для коммитов

Тип изменения:

- feat - используется при добавлении новой функциональности приложения
- fix - если исправили какую-то багу
- refactor - рефакторинг кода приложения
- style - исправляем опечатки, исправляем форматирования
- chore - обычное обслуживание кода
- revert - отмена изменений
- perf - оптимизация кода, улучшение производительности
- docs - всё, что касается документации
- test - всё, что связано с тестированием
- build - всё, что связано с билдом
- ci - всё, что связано с процессом деплоя
- any - не входящее ни в одну из категорий

Настройка .gitignore

В большинстве проектов есть файлы или целые директории, в которые мы не хотим (и, скорее всего, не захотим) коммитить. Мы можем удостовериться, что они случайно не попадут в git add -A при помощи файла **.gitignore**

1. Создайте вручную файл под названием .gitignore и сохраните его в директорию проекта.

2. Внутри файла **перечислите** названия файлов/папок, которые нужно игнорировать, каждый с новой строки.

3. Файл .gitignore **должен быть добавлен, закоммичен и отправлен на сервер**, как любой другой файл в проекте.

Вот хорошие **примеры файлов**, которые **нужно игнорировать**:

- Логи
- Артефакты систем сборки
- Папки node_modulesв проектах node.js
- Папки, созданные IDE, например, Netbeansили IntelliJ
- Разнообразные заметки разработчикаили .envфайлы

Самые частые ошибки при работе с Git

fatal: Not a git repository

Ошибка возникает при выполнении команд git в каталоге, который не был инициализирован как репозиторий Git.

Что делать

Инициализируйте новый репозиторий Git или перейдите в существующий

```
# Инициализировать новый репозиторий
$ git init

# Или перейти в существующий репозиторий
$ cd path/to/git/repo
```

Самые частые ошибки при работе с Git

error: failed to push some refs

Локальный репозиторий не синхронизирован с удаленным репозиторием

Что делать

Получите последние изменения из удаленного репозитория и объедините их с вашим локальным репозиторием

```
$ git pull origin master (или main,если
главная ветка- main)
```

Самые частые ошибки при работе с Git

error: Your local changes to the following files would be overwritten

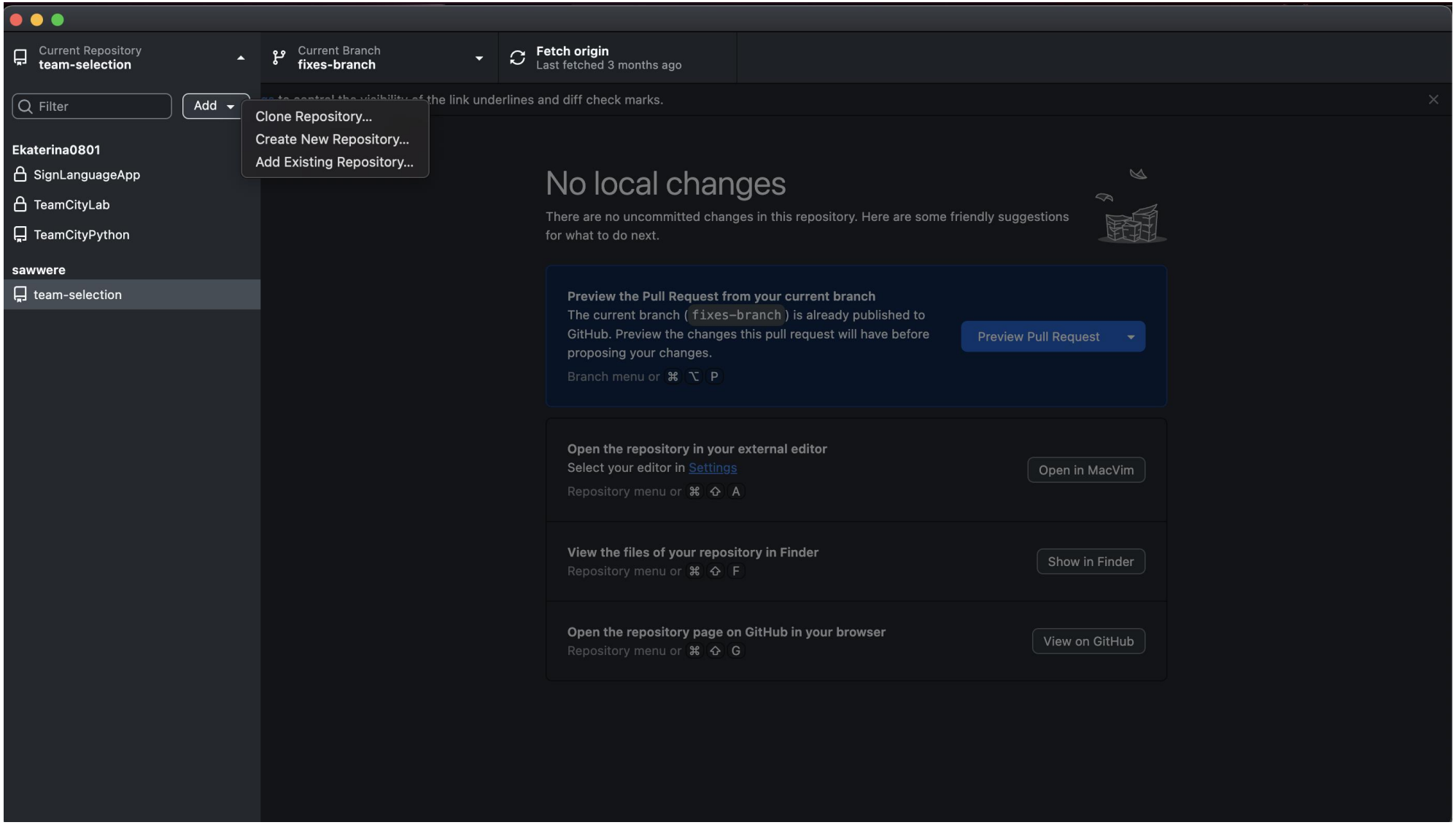
У вас есть локальные изменения, которые не были зафиксированы, и вы пытаетесь получить данные из удаленного репозитория

Что делать?
Либо зафиксируйте свои изменения, либо отложите их

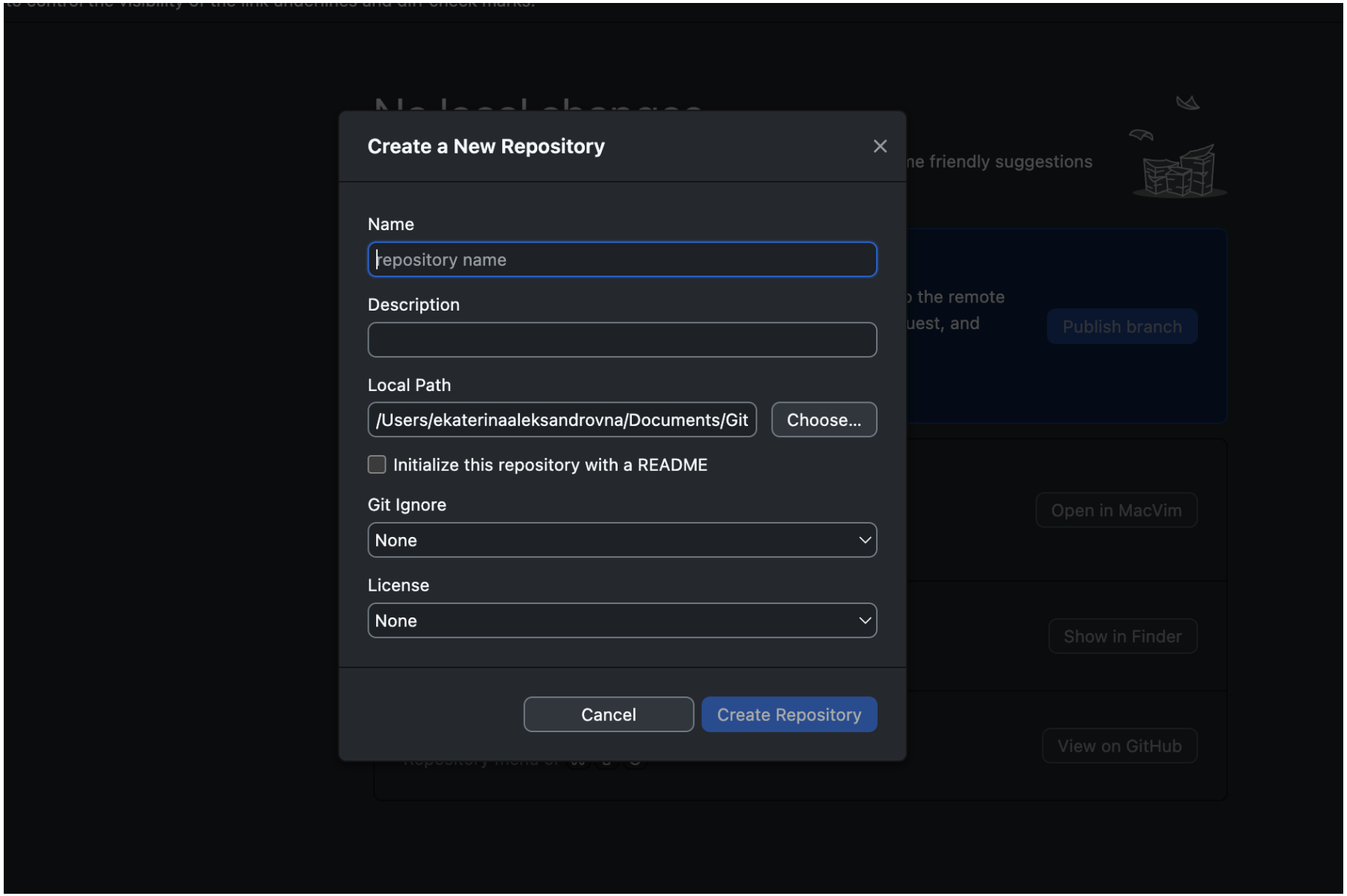
```
# Зафиксировать свои изменения
$ git add . git commit -m "Ваше сообщение о коммите"

# Или отложить свои изменения
$ git stash
```

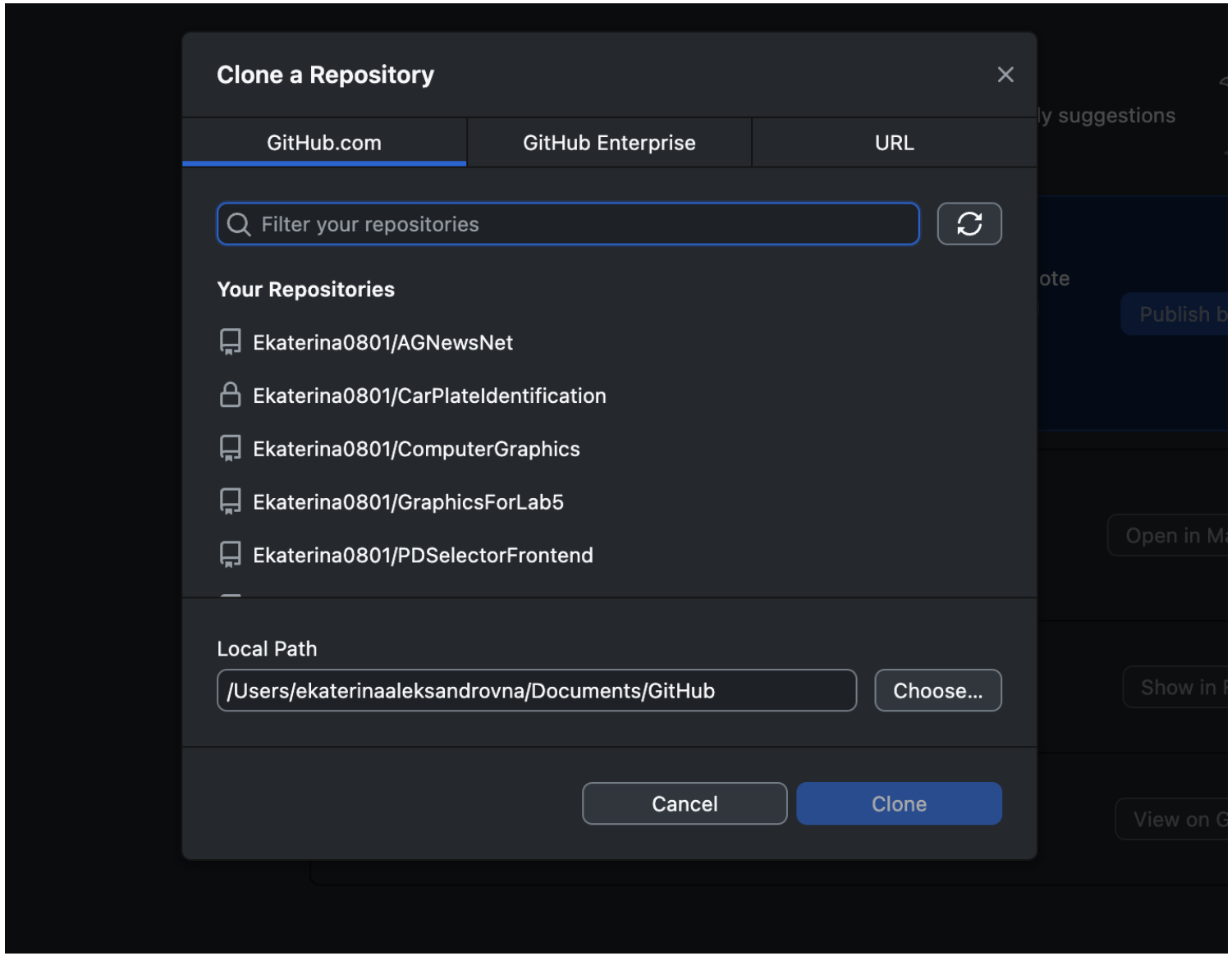
Работа через Github Desktop



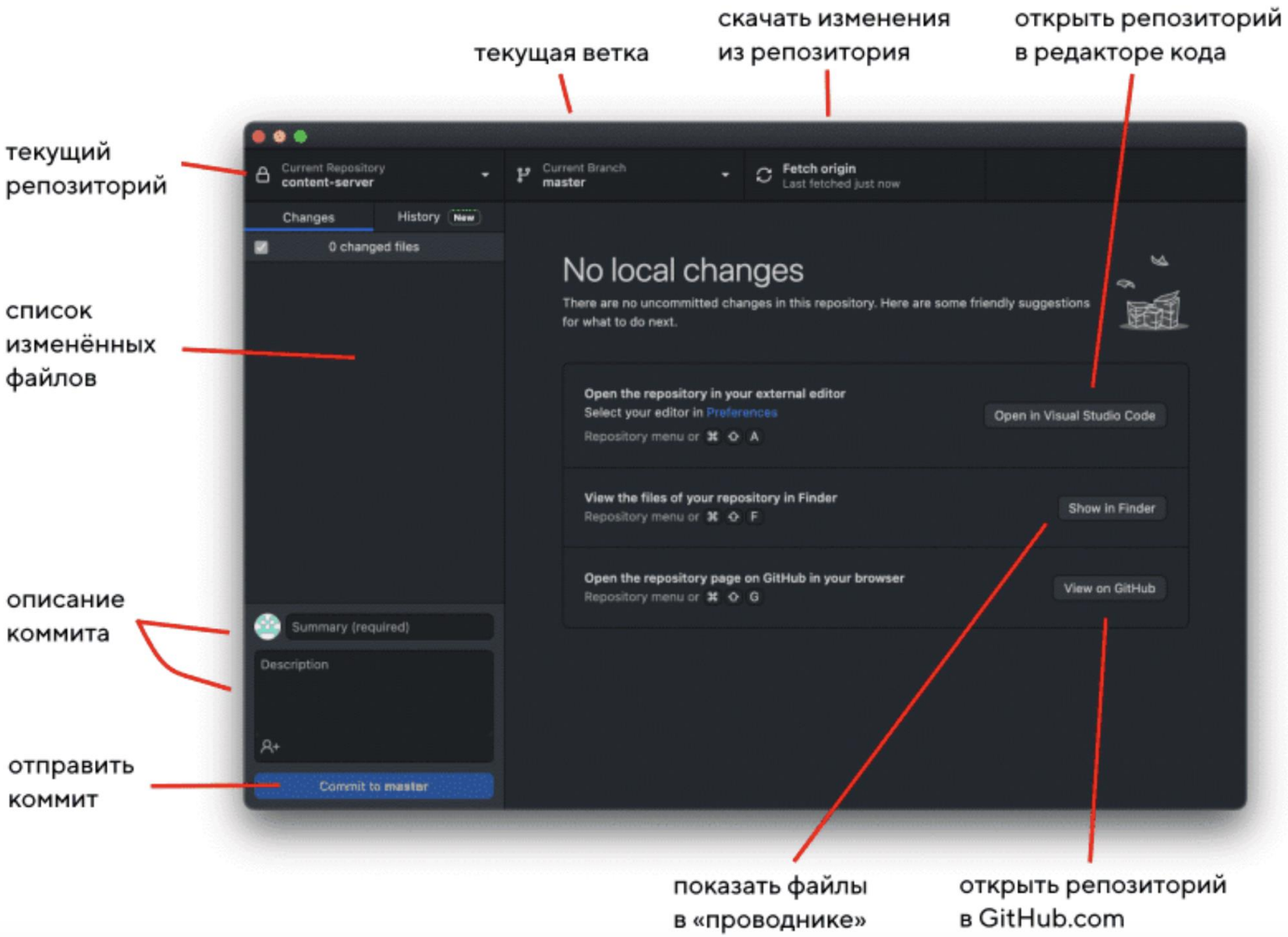
Работа через Github Desktop



Работа через Github Desktop



Работа через Github Desktop



Работа через Github Desktop