ВСР 2.3 №4

Git – система контроля версий, используемая разработчиками по всему миру, и в рамках данного задания будет рассказано об основных принципах работы с Git через терминал. Сам по себе Git без графического интерфейса уже можно использовать как систему для контроля версий и стоит понимать, что Git — это инструмент, а GitHub — конкретная платформа, часто используемая в связке с Git. В рамках задания все команды будут выполняться локально, но для наглядности все изменения будут публиковаться в репозиториях на GitHub.

Прежде чем выполнять все действия, можно убедиться что Git вообще установлен: поможет нам в этом команда

git --versionСнимок экрана 2025-09-19 в 04.09.56.png

(ну и если его нет, то надо скачать его с официального сайта или воспользовавшись нужной командой, в зависимости от ОС). Также возможно, что командами

git config --global user.name «ВашеИмя" и

git config --global user.email "[ВашаПочта@example.com](mailto:%D0%92%D0%B0%D1%88%D0%B0%D0%9F%D0%BE%D1%87%D1%82%D0%B0@example.com)"

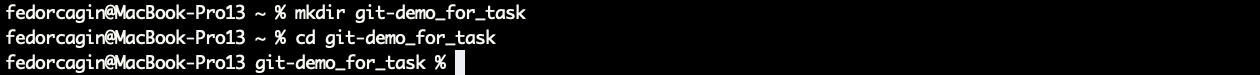
понадобится настроить ваше имя и почту, но в данном случае всё уже готово.

На этом этапе стоит проверить самое важное — подключение к GitHub. Команда

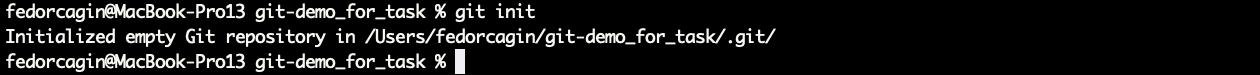
Снимок экрана 2025-09-19 в 04.12.07.pngssh -T [git@github.com](mailto:git@github.com)

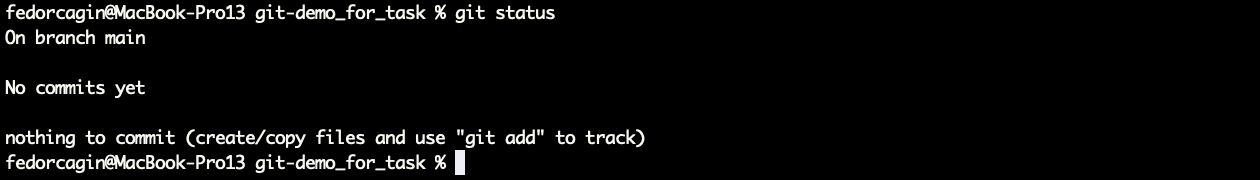
не выполняет никаких действий с репозиториями, а просто проверяет, узнаёт ли GitHub мой компьютер по секретному SSH-ключу. Если настроить Git на своём сервере, то можно отключить это – но GitHub и многие другие аналогичные сервисы не дают сбросить эти меры безопасности. Если SSH ключа нет – то его можно легко добавить или использовать менее безопасные токены. Однако это отдельная тема с подробно расписанными инструкциями на GitHab: в данном случае можно увидеть надпись «Hi Fedor-Chagin! You've successfully authenticated, but GitHub does not provide shell access.» это значит что здесь уже всё настроено, и можно приступать к непосредственной работе с проектом. Для этого создадим папку проекта и перейдём в неё:

mkdir git-demo\_for\_task

cd git-demo\_for\_task

Теперь, когда мы внутри папки нашего проекта, самое время превратить её в репозиторий, за которым будет следить Git. Для этого существует команда

git init

Это значит, что Git создал скрытую служебную папку .git — именно в ней будет храниться вся история изменений, информация о коммитах и настройки именно для этого проекта. Теперь наша папка — не просто папка, а полноценный репозиторий. Можно сразу проверить, что нам об этом говорит Git –

git status

показывает текущее состояние нашего репозитория: какие файлы изменены, какие готовы к сохранению, а какие ещё не отслеживаются. В данном случае мы находимся на ветке main (или master в некоторых старых настройках), и видим что здесь не было ни одного коммита. А главное — ей просто нечего нам показывать в плане изменений, потому что у нас даже нет ни одного файла. Исправим это и создадим самый главный файл любого Снимок экрана 2025-09-19 в 04.19.12.pngпроекта — README.md.

echo "# какой-то текст" >> README.md

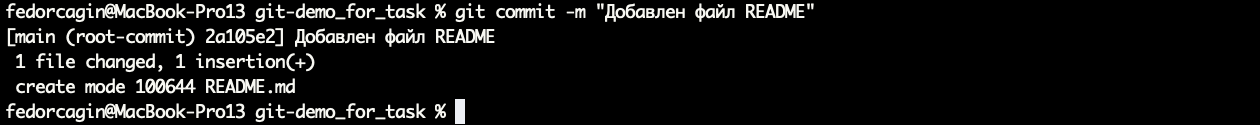
Командой echo создаётся текстовая строка, а стрелочками она направляется прямо в указанный файл. Теперь, если снова проверить статус, то увидим, что Git обнаружил новый файл и сообщает, что он «untracked» – это значит, что за ним ещё не ведётся наблюдение ( т.е. Git видит этот файл в папке, но целенаправленно игнорирует его и не сохраняет его историю изменений). Чтобы Git начал его отслеживать, нужно специально добавить его в так называемый «индекс» или «staging area» — промежуточную область, где собираются вставленный-фильм.pngизменения для следующего сохранения. Делается это командой  
git add README.md  
Снова проверим статус:

git status

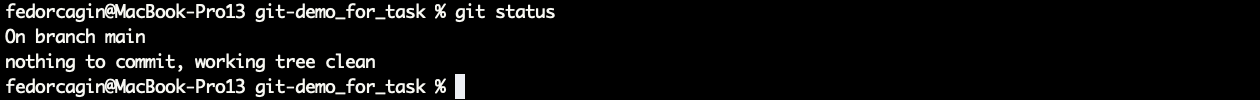
Теперь Git сообщает, что есть новый файл, и он «staged» — то есть готов к сохранению. Он уже в индексе и ждёт своего момента. Это отличие между просто файлом в папке и файлом, готовым к коммиту — одна из ключевых идей Git. Теперь, когда изменения готовы и лежат в индексе, самое время сделать их «снимок» — т.е. зафиксировать текущее состояние проекта. Правильно это называется «сделать коммит». Для этого используется команда

git commit

Но просто сделать коммит — мало. К нему нужно обязательно добавить комментарий, который объясняет, что сохраняется в этом коммите. Это в интересах человека, работающего с Git и в интересах разработчиков, с которыми он работает.

Поэтому припишем флаг -m и добавим комментарий. В результате команда будет выглядеть так:  
git commit -m "Добавлен файл README"

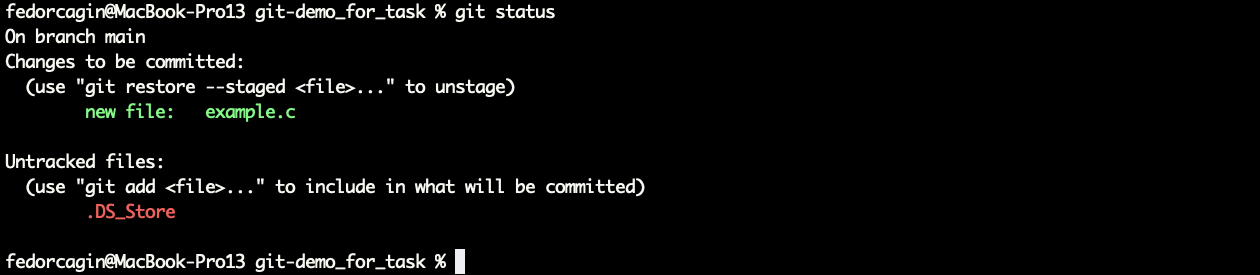
После выполнения команды Git сообщит, что он сделал. В данном случае: «[main (root-commit) 2a105e2] Добавлен файл README». Это значит, что был создан самый первый (root) коммит в ветке main с уникальным идентификатором 2a105e2 и введённым сообщением. Если на этом этапе ввести

git status,

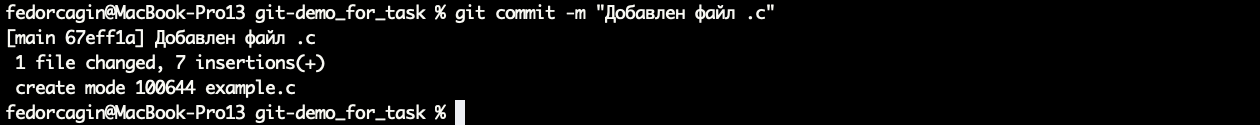
то Git снова скажет «нечего коммитить» — потому что все изменения, которые были (новый файл), уже успешно был закоммичен. Рабочая папка «чистая». Для наглядности можно повторить последние несколько действий и сделать ещё один коммит: для примера можно добавить файл example.c (перемещён вручную) в папку проекта и сохраним эти изменения – всё так же в ветку main. Для этого нужно:

вставленный-фильм.pnggit add example.c

проверить статус

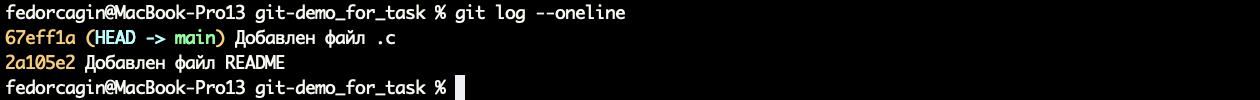
git status

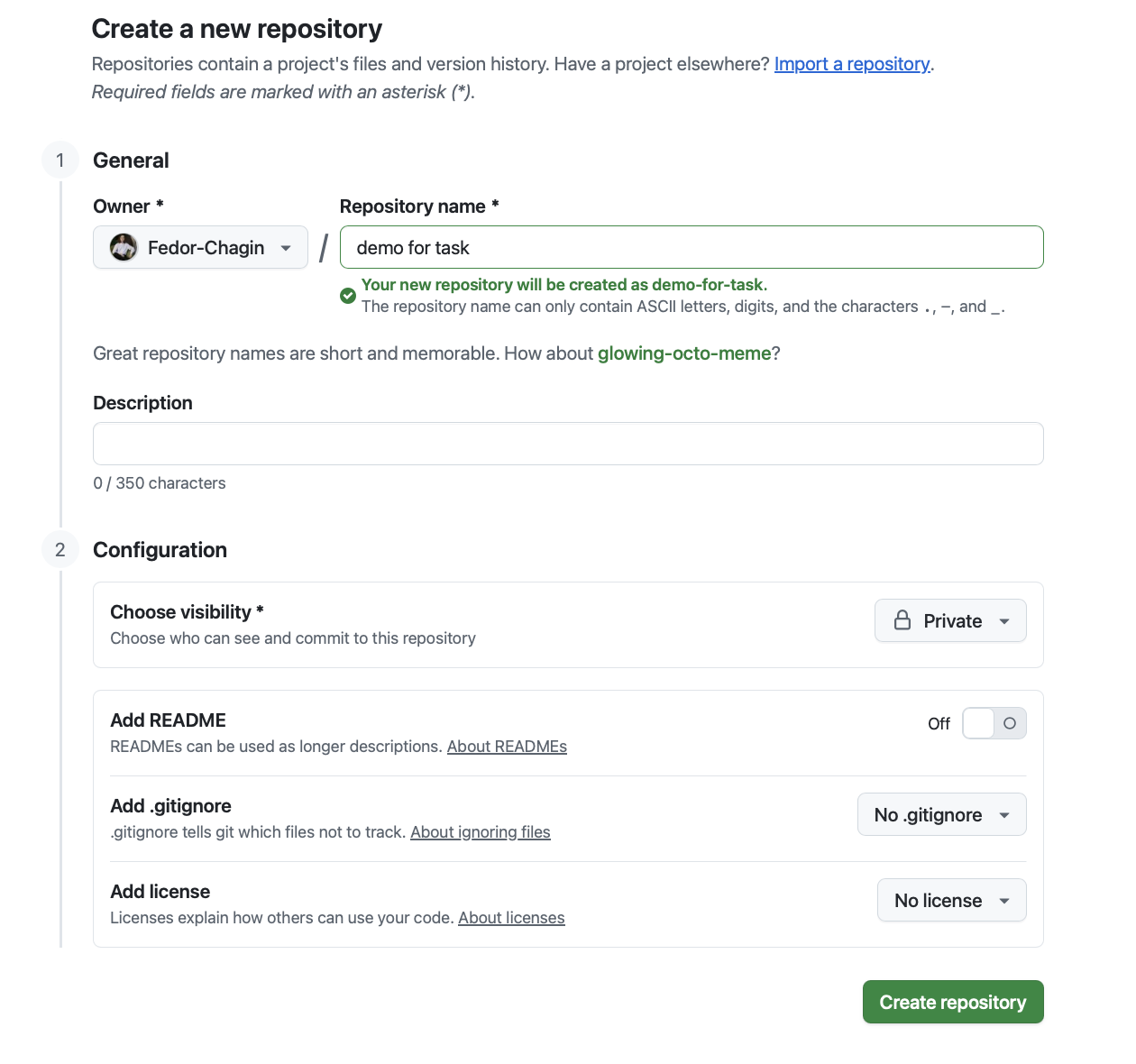
и закоммитить

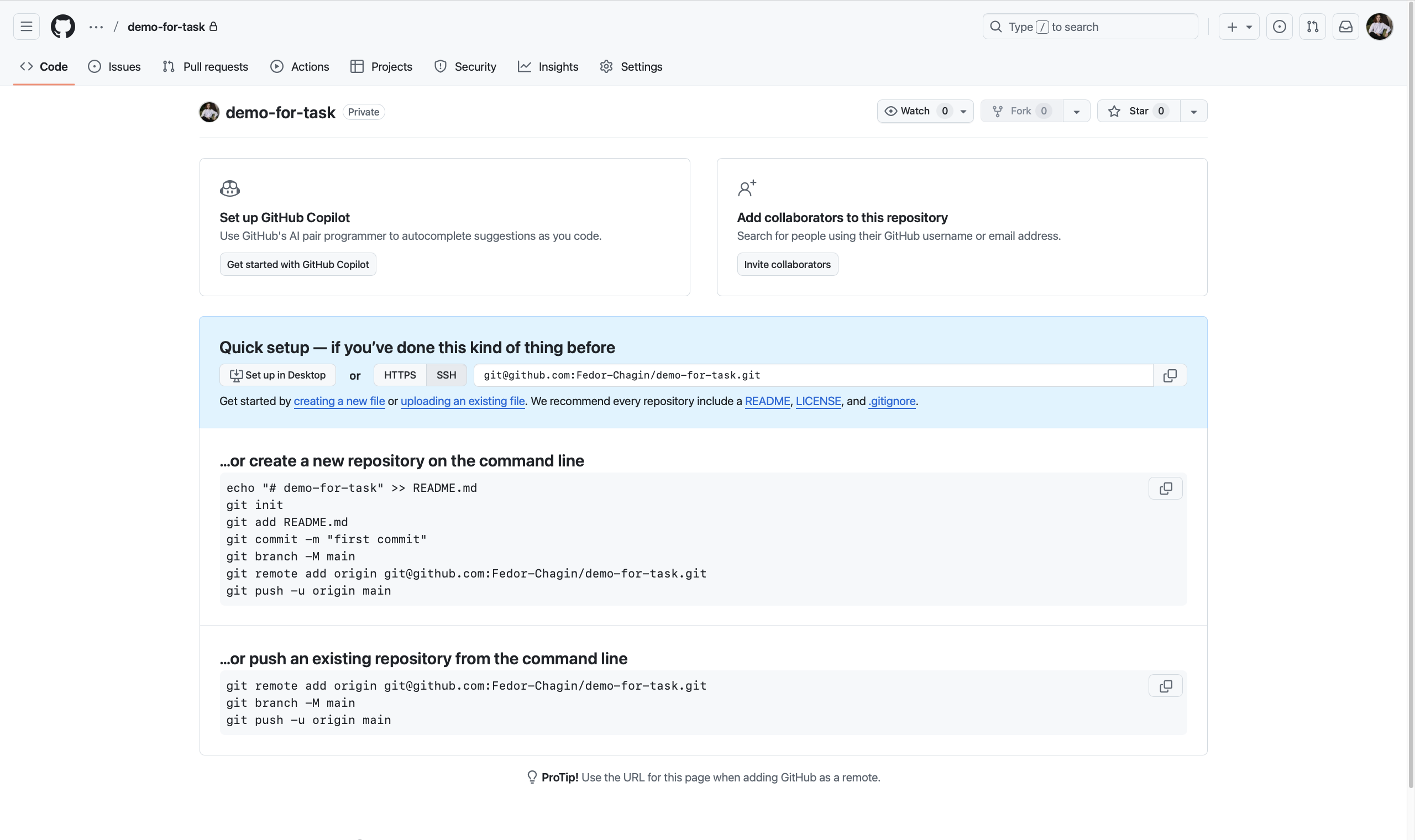
git commit -m "Добавлен файл .c»

А чтобы полюбоваться на свежую историю, можно использовать команду:

git log --oneline

будет показана краткая история коммитов в обратном порядке (т.е. сверху, как видно — самый свежий).

Таким образом был создан готовый локальный репозиторий с историей коммитов: но вся эта история пока что существует только на конкретном компьютере, где выполнялись эти команды. Чтобы поделиться ей с коллегами или просто иметь backup в облаке, нужно запушить эти изменения – эта возможность предусмотрена в Git. Вместе с Git часто используется GitHub, ресурсами которого и будет продемонстрирована и возможность отправки таких изменений на сервера. Для этого понадобится пустой репозиторий на GitHub:



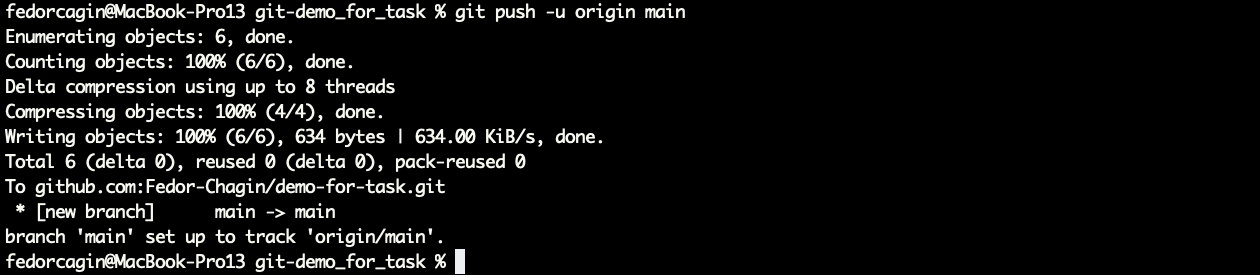
После создания репозитория на выбранном сервисе можно добавить туда все изменения. Удобнее и безопаснее всего использовать вариант по SSH: для этого нужно выполнить 2 команды. Первая команда

git remote add origin git@github.com:ваш\_логин/название\_репозитория.git

cвязывает нужный локальный репозиторий с удаленным на GitHub. Origin — это просто стандартное короткое имя для удаленного репозитория. В данном случае целиком она выглядит вот так:

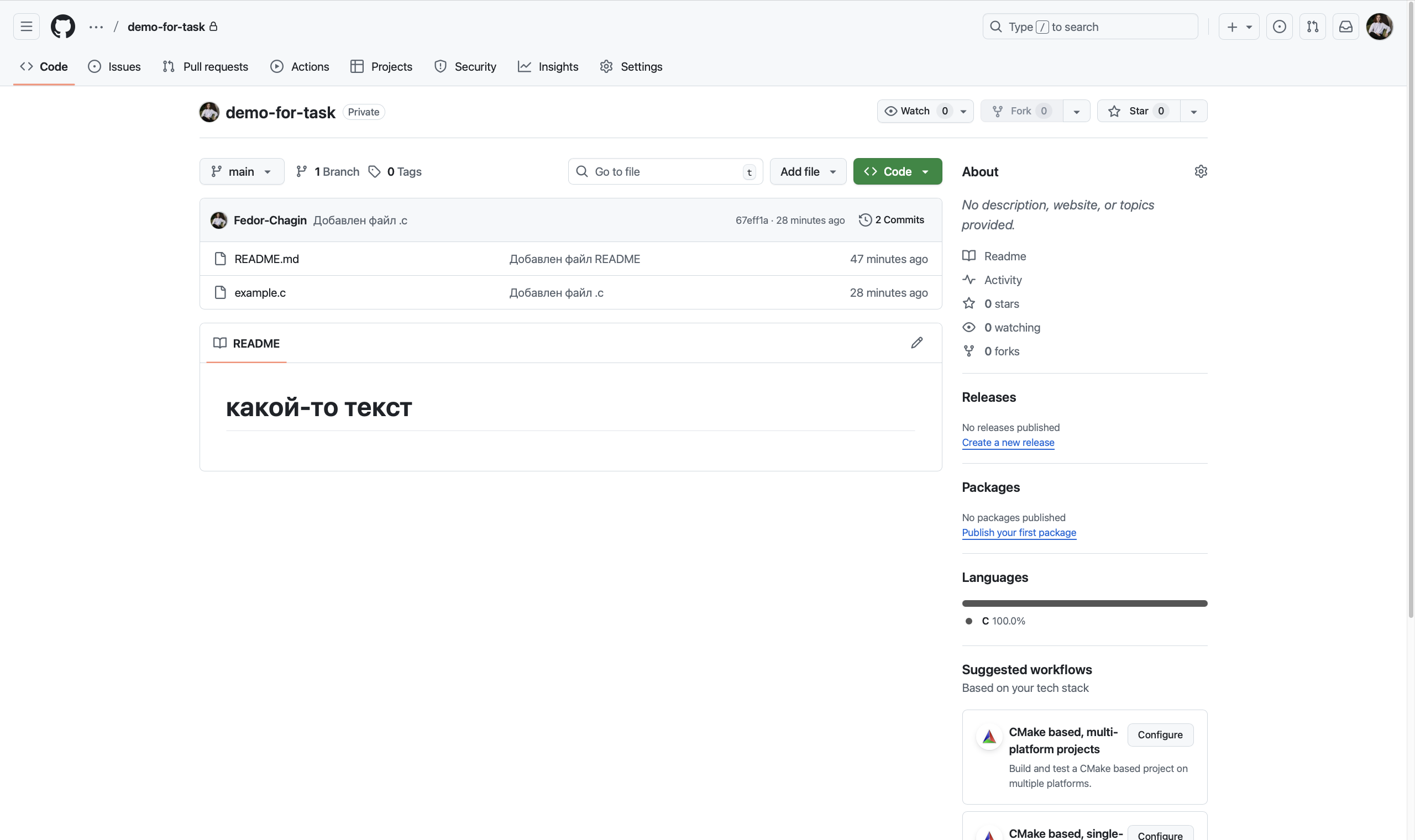
Снимок экрана 2025-09-19 в 05.12.22.pnggit remote add origin git@github.com:Fedor-Chagin/demo-for-task.git

Вторая команда

git push -u origin main

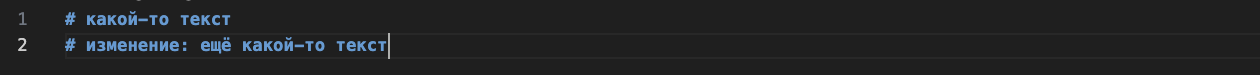
отправляет всю локальную историю коммитов (ветку main) на сервер GitHub. Ключ -u нужен только один раз для каждой ветки. Он запоминает, куда были отправлены изменения, и в дальнейшем для отправки новых коммитов из этой же ветки будет достаточно просто набрать git push. Изменить репозиторий (например, если изначально был указан не верный), можно удалив старый выбор командой

git remote remove origin.

После выполнения команды можно обновить страницу репозитория на GitHub и увидеть, что файлы успешно загрузились – теперь наш код не только на компьютере, но и в безопасном месте на GitHub.

Однако может появиться необходимость что-то изменить в проекте – для этого нужно создать новую ветку.

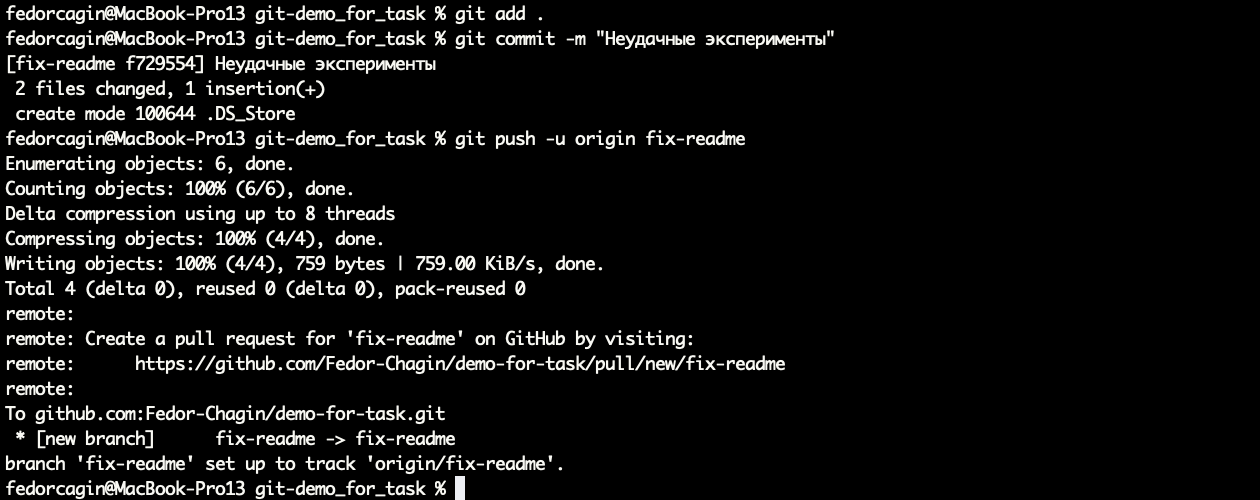
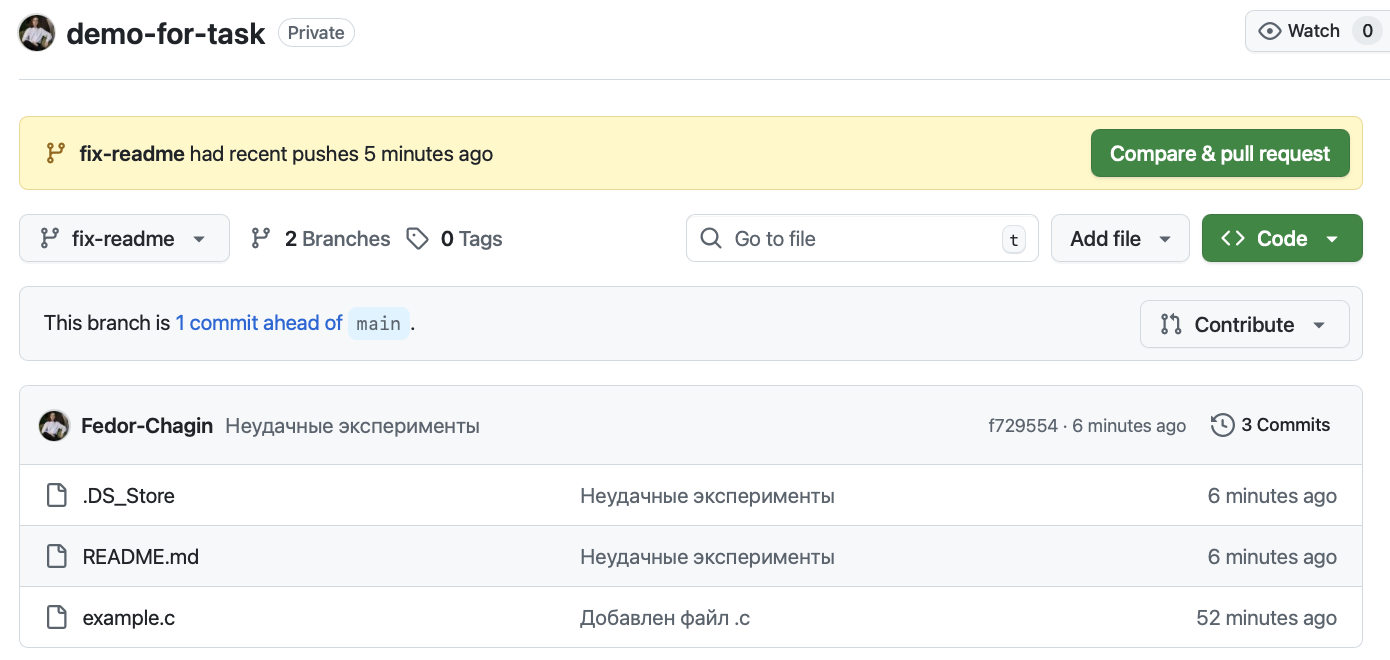
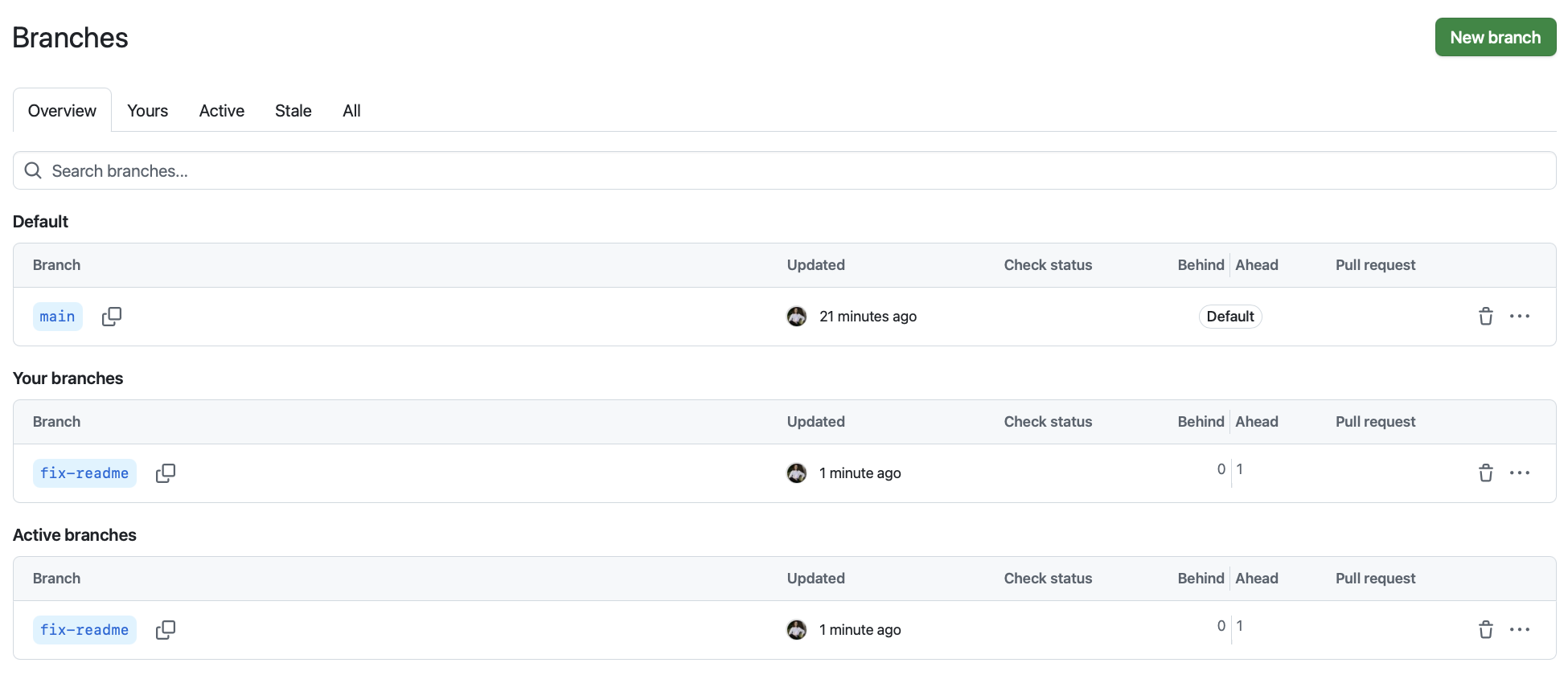
git switch -c fix-readme

Для примера можно зайти в проект и изменить файл readme, а так же (вручную) добавить папку exampe\_№1.

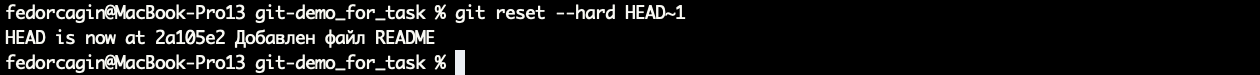
И сохранить + запушить эти изменения:

git add .

git commit -m "Неудачные эксперименты"

git push -u origin fix-readme

Но если изменения не оправдали результата, появились новые баги или просто было принято решение не вносить правки – то можно откатиться к прошлой версии. В данном случае можно просто откатиться на 1 версию назад:

git reset --soft HEAD~1

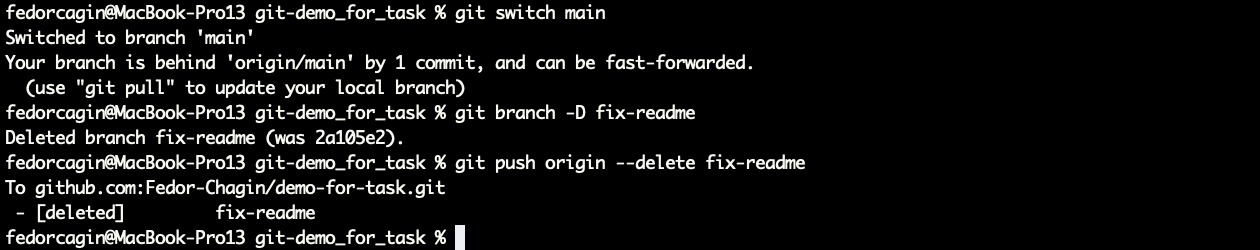
После такого мы увидим старую версию проекта: с 1 строчкой в readme и без добавленной папки. Но если изменения не устраивают до такой степени, что их не хочется оставлять как неудачные версии – поможет полное удаление ветки: для этого также нужно переключиться на main

git switch main

Удалить ветку fix-readme локально

git branch -D fix-readme

а затем и в GitHub

git push origin --delete fix-readme

После выполнения ветка исчезнет даже на GitHub и не будут засорять процесс разработки.