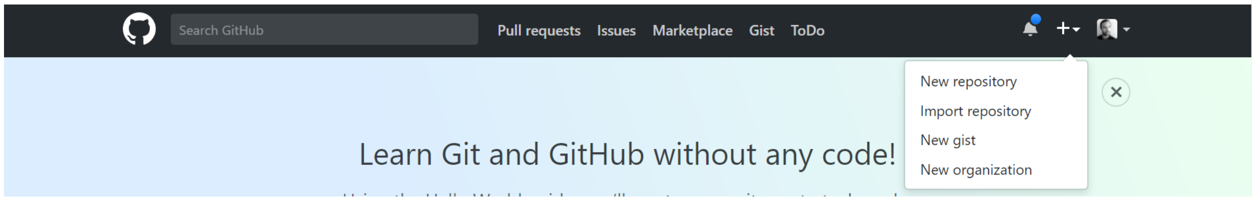
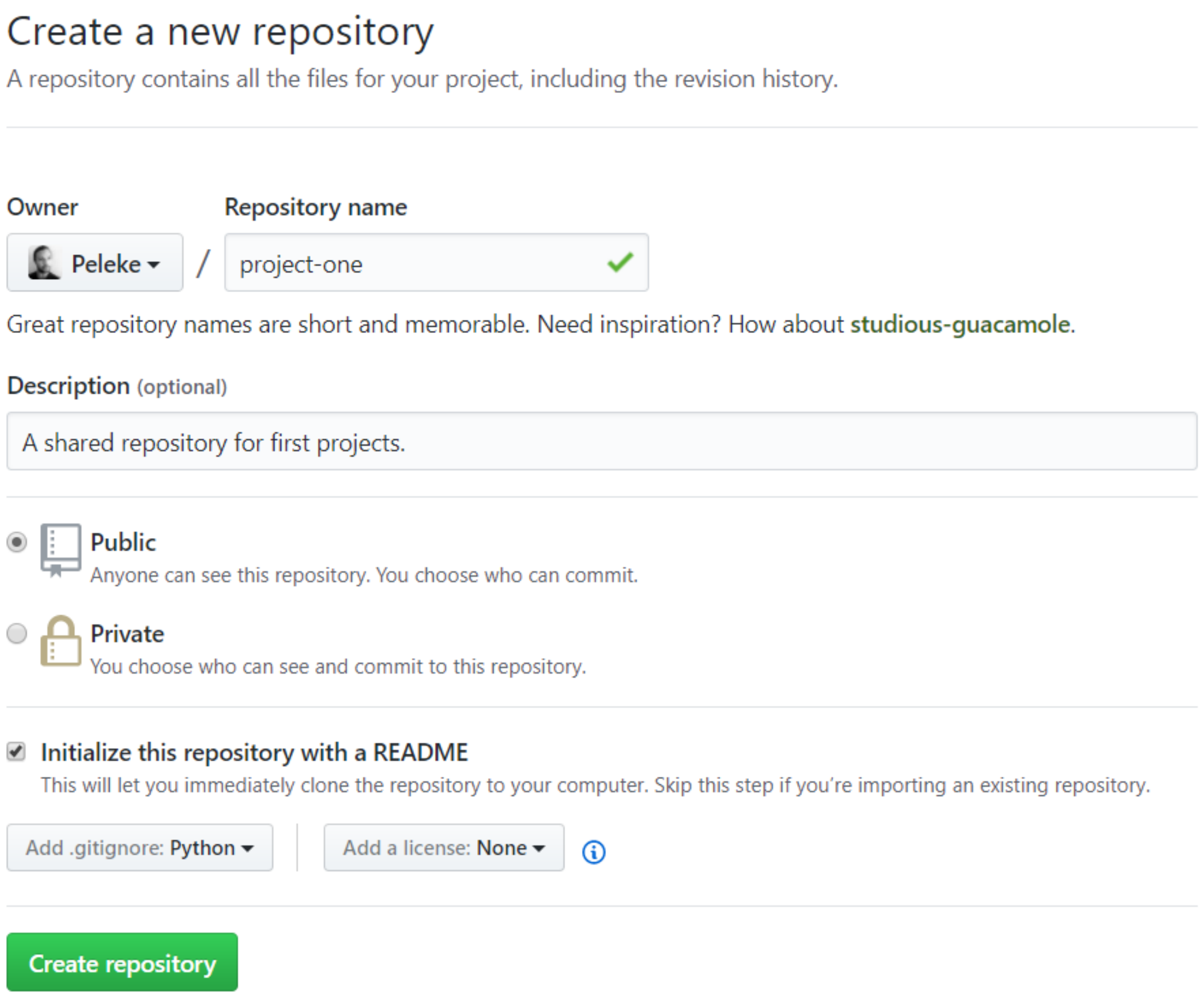
Creating a Project Repo



Learn Git and GitHub without any code!

- Fill out the fields on the new repo page.
- Initialize with a `.gitignore`.
- Choose `Python` in the gitignore dropdown.



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: Peleke / Repository name: project-one

Great repository names are short and memorable. Need inspiration? How about **studious-guacamole**.

Description (optional): A shared repository for first projects.

[✓ Public] Anyone can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

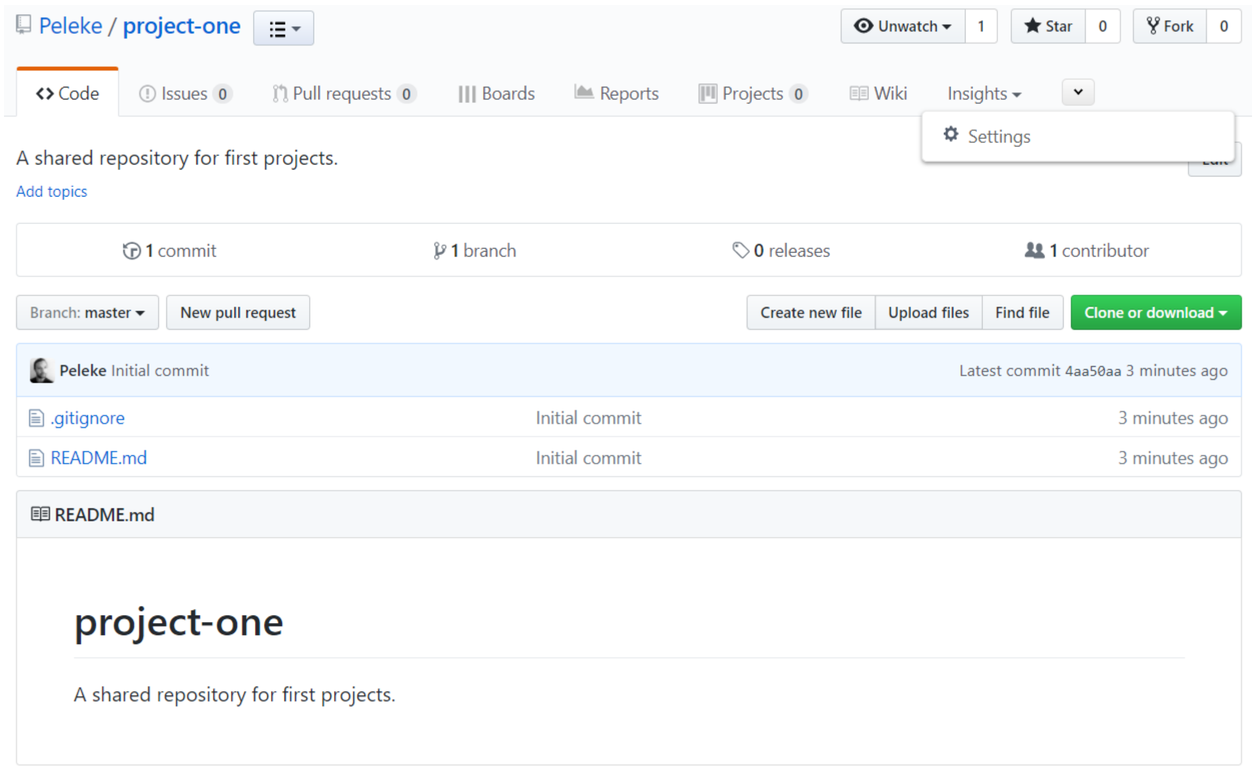
[✓ Initialize this repository with a README]
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **Python**

Add a license: **None**

- Slack the remote URL (i.e., the link to the repo) to their teammates.
    - Team members will `git clone` this link.

- by default, only the creator of the repo can push changes.
- "open up" the repo by adding **collaborators**.



A shared repository for first projects.

Peleke Initial commit — Latest commit 4aa50aa 3 minutes ago

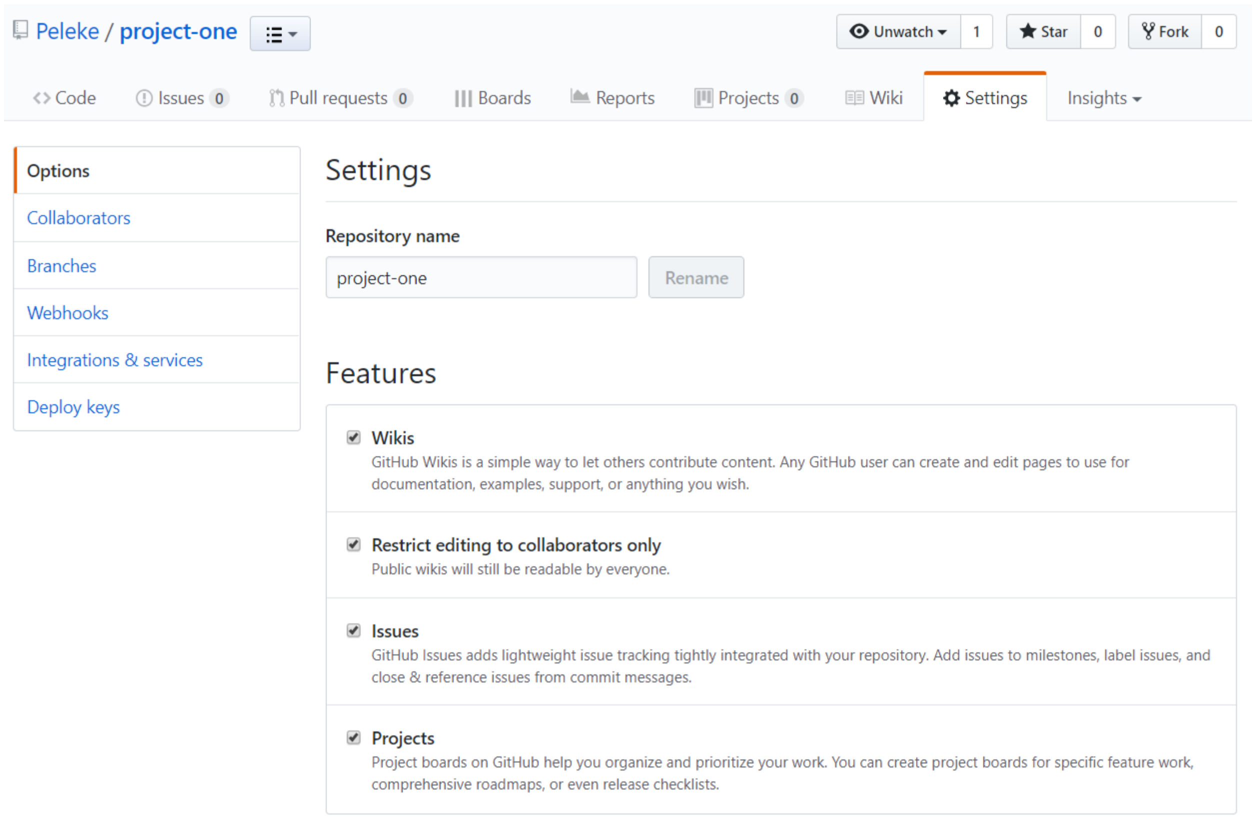
.gitignore	Initial commit	3 minutes ago
README.md	Initial commit	3 minutes ago

README.md

project-one

A shared repository for first projects.

- Navigate to the repository settings.
- Navigate to the collaborators tab, and enter your password when prompted.



Settings

Repository name

project-one

Features

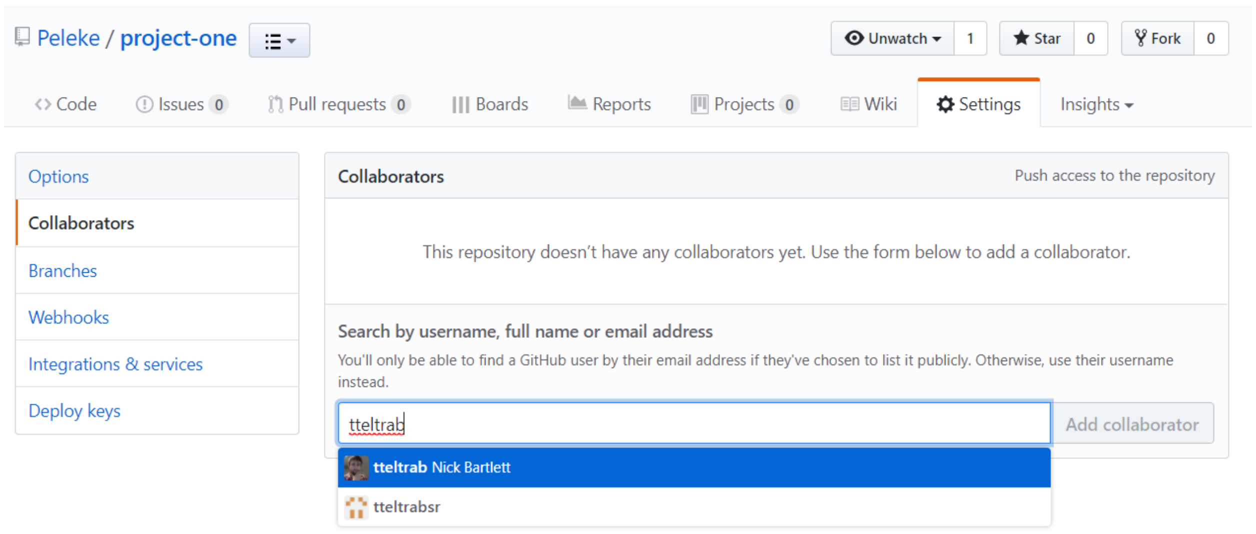
☑ Wikis
GitHub Wikis is a simple way to let others contribute content. Any GitHub user can create and edit pages to use for documentation, examples, support, or anything you wish.

☑ Restrict editing to collaborators only
Public wikis will still be readable by everyone.

☑ Issues
GitHub Issues adds lightweight issue tracking tightly integrated with your repository. Add issues to milestones, label issues, and close & reference issues from commit messages.

☑ Projects
Project boards on GitHub help you organize and prioritize your work. You can create project boards for specific feature work, comprehensive roadmaps, or even release checklists.

- From here, you can search for their teammates by username.



Collaborators — Push access to the repository

This repository doesn't have any collaborators yet. Use the form below to add a collaborator.

Search by username, full name or email address

You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

tteltrab

tteltrab Nick Bartlett

tteltrabsr

- Everyone in each group should now be able to make changes to the shared repo.

- **Reminder:** *everyone in the group must clone the new repository*.

0. Getting the Repo

Note that, in the examples below, we use `git status` before every `git commit`. This is a best practice that helps ensure a deliberate commit history.

but assume we've always run `git status`before any `git commit`.

Clone from GitHub

If someone has already shared a repository on GitHub, you can **clone** it to your local machine with `git.

```
# Clone an existing repo.
git clone <repo_url>
# Navigate into newly created repo directory
cd <repo_name>
```

1. Add Files

Next, we simply develop as normal, and `commit` our changes whenever we make significant progress.
In general, it's best to **commit early** and **commit often**. Frequent snapshots ensure you'll never be far away from a "last working version".

```
# Create a file, called clean_data.py
touch clean_data.py

# Add and commit clean_data.py...
git add clean_data.py
git status
git commit -m "First commit."

# Add cleanup code to clean_data.py...
git add clean_data.py
git status
git commit -m "Clean up provided data."

# Add code to export clean data...Note that `add .` adds
# everything in the current folder
git add .
git status
git commit -m "Export clean data as CSV."
```

2. Create Branches

To create a new, isolated development history, we must create **branches**.

```
# Create new branch and switch to it
# Long form: `git checkout --branch data_analytics`
git checkout -b data_analytics
```

Alternatively, we can create a branch and then switch to it as two separate steps, though this is uncommon.

```
git branch new_branch_name
git checkout new_branch_name
```

Once we've created a new branch, we can develop as normal:

```
# Create file to contain data analysis
git add analysis.ipynb
git status
git commit -m "Add Jupyter Notebook for data analysis."

# Add notebook cells summarizing data
git add analysis.ipynb
git status
git commit -m "Add summary tables to Jupyter Notebook."

# Export analyzed data and/or plots
git add .
git commit -m "Export analysis results and save plots as PNG files."
```

3. Merge

Once we've developed and tested the changes on our `data_analysis` branch, we can include them in `master` by **merging** the two branches.

```
# Move back to master
git checkout master

# Merge changes on data_analysis with code on master
git merge data_analysis

# Delete the data_analysis branch
git branch -d data_analysis
```

N.b., deleting the `data_analysis` branch isn't necessary, but it's best practice to prune unneeded branches.

Pushing to GitHub

- Up until now, your `data_analysis` branches aren't visible to their teammates—there's no way for their group members to see the work they've done.
- **Push** code to from our computers to GitHub, after which our teammates can **pull** it from GitHub to their computers.
- Two steps to push our local branch to GitHub.
- First, checkout the branch we want to push to GitHub
- Then, run: `git push origin <branch_name>`
  - Run this line to push their local branches to their shared repository.
- Pushing your local branch to GitHub, allows your teammates to get access to it later.
- After you have pushed to GitHub, now it is time to checkout master, and then:
  - First, run `git pull`
  - Then, run `git checkout <branch_name>`, where `<branch_name>` is the name of one of their teammates' branches.
  - Verify that the code they checked out does indeed come you're your teammate's branch.
- This allows us to easily share different versions of our code across workstations, and allows us to easily test those versions on our local computers.