

Simple git workflow for your projects

Create a new private repo on github.com.

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *

 bartdorsey ▾

Repository name *

/ test-repo ✓

Great repository names are short and memorable. Need inspiration? How about solid-invention?

Description (optional)

☐



Public

Anyone on the internet can see this repository. You choose who can commit.

☒



Private

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

Give your repository a descriptive name.

Make sure you don't check the **Initialize this repository with a README** and make sure **Add .gitignore** and **Add a license** are both set to **None**.

If you are working on a project with a pair programming partner you can give them access to the private repo in github.com's Settings... Manage Access section. Then they can clone down a copy of the project as well.

if you are starting with an empty directory:

- `git push -u origin <a name for your branch>` (If you've already pushed once, you can shorten this to a simple `git push`).
- Go to github.com, and create a pull request. Perhaps get your coding pair partner to review your PR. Then merge your PR into master.
- Checkout master now that your feature is merged `git checkout master`.
- `git pull` to update master with the new merged changes.
- Repeat this cycle for other features.

Pair programming with github

You and your pair can both work out of a single repository on github to make sharing code back and forth easier.

- Pick someone to create the initial repo (It doesn't matter who does this, you can both have access to the code in your own repo at the end of the day)
- Add the other programmer as a "collaborator" to this repo.
- Both of you clone the repo to your local repository with the `git clone` command.
- The "Driver" should work on code. When it comes time to swap roles, have the "Driver" commit and push the code to git hub with `git push -u origin master`
- Then the new Driver should `git pull` the code down and start working on it.
- Keep swapping this way with each driver committing code and pushing to github.
- At the end of the day the student that did not own the original github repository can do the following steps to get a copy of all the code in their own repository.
 - Make sure everything is committed and pushed before you start this.
 - Create a new repository on github.
 - Remove the `origin` remote from your existing cloned repo `git remote remove origin`
 - Add your new repository as the new origin `git remote add origin <new repo url>`
 - Push your master branch to your new repo `git push -u origin master`

Note: it's a good idea to have a `README.md` markdown file, github will automatically show it as the main content of your repo below your code. (the `echo` command makes a `README.md` file automatically from the command line, but you can feel free to just create one with VSCode instead)

```
echo "# A Title for my Repo" >> README.md
git init
git add README.md
git commit -m "Initial Commit"
git remote add origin <Put the url to your repo github gives you here>
git push -u origin master
```

if you don't have a repository locally but have code already.

```
git init
git add .
git commit -m "Initial Commit"
git remote add origin <Put the url to your repo github gives you here>
git push -u origin master
```

if you already have a local git repo with code

```
git remote add origin <Put the url to your repo github gives you here>
git push -u origin master
```

Workflow to repeat as you work on code

if you just want to use master

- Edit files
- `git add .` or `git add <filename>`
- `git commit -m "A good descriptive commit message"`
- `git push`
- Repeat this cycle

if you want to practice doing feature branches

Often our projects will have 'phases', you can make a new feature branch for each phase of the project, this is great practice for what it will be like at a real company.

- Make sure you are on master first with `git status`.
- `git checkout -b <a name for your branch>`.
- Edit your files.
- `git add .` or `git add <filename>`.
- `git commit -m "A good descriptive commit message"`.