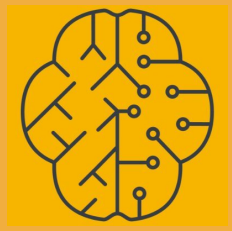


Long Workshop

Version Control with GitHub Desktop

The Agency Fund R Workshop Series

Zezhen (Michael) Wu

Agenda

- Understanding git
- Using the GitHub Desktop software to execute version control for your coding project (whether it is Stata or R)
- Using git in Rstudio

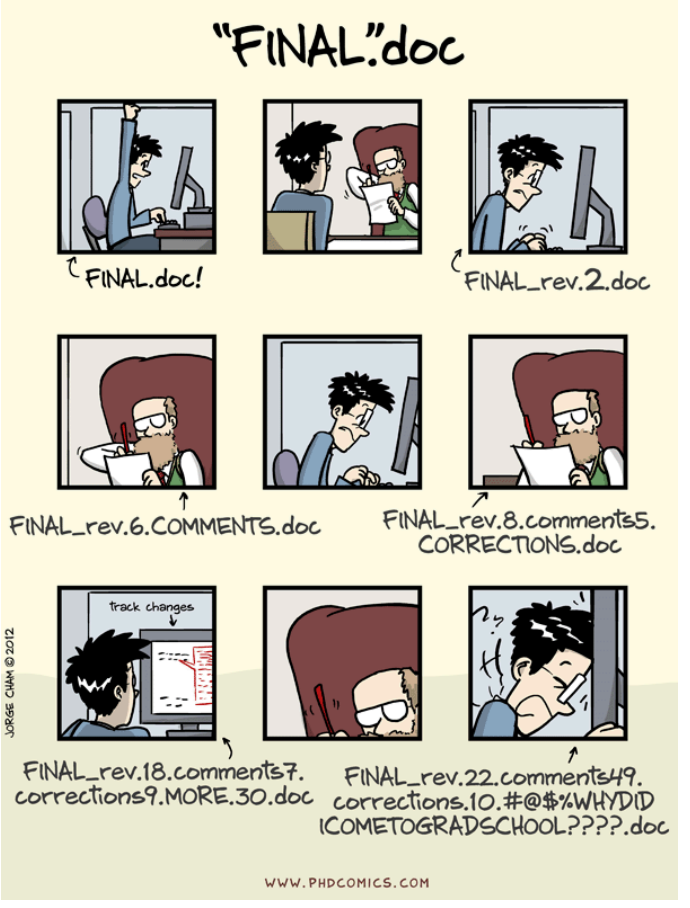
Learning tools

- A Medium post about Stata and GitHub Integration
- A chapter on version control in "An Introduction to R"
- Getting started with GitHub Desktop by GitHub

What is Version Control?

A Version Control System (VCS) keeps a record of all the changes you make to your files that make up a particular project and allows you to revert to previous versions of files if you need to.

Think about a Word Doc that tracks each sentence that you write and allows you to go back to previous versions of the SAME document, WITHOUT the need to create multiple documents.



Why use Version Control?

- Collaboration: Enables multiple teammates to work on a project
- Tracking Changes: Keeps a history of who made what changes
- Reverting: Ability to undo changes, restoring previous versions
- Conflict Resolution: Helps in resolving conflicting changes
- Enhanced Workflow: Tailors to different development workflows

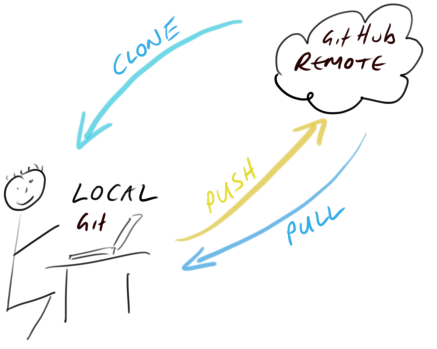
What is Git and GitHub?

Git

- Git is a distributed version control system that tracks changes in any set of computer files.
- All the files that make up a project is called a *repository*.

GitHub

- GitHub is a web-based hosting service for Git repositories which allows you to create a remote copy of your local version-controlled project.



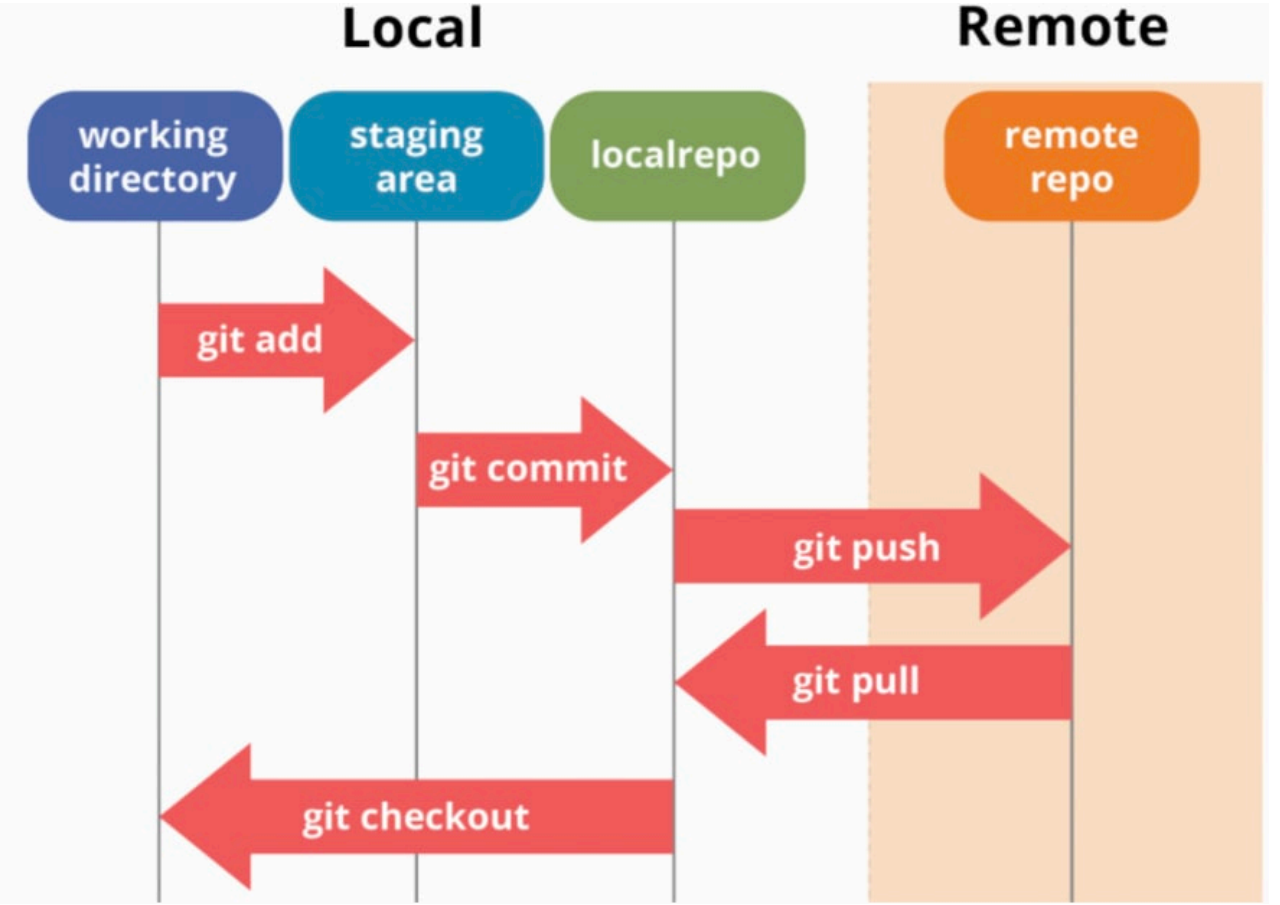
Basic Git Operations

- Committing Changes
- Pushing to Remote Repositories
- Pulling from Remote Repositories
- Branching and Merging

Note:

- You can do the above operations by writing Git commands in a command line tool (e.g., Terminal on Mac or PowerShell on Windows), after installing git, which can be somewhat difficult to learn if you are new to programming.
- An alternative way is to use Github Desktop: https://desktop.github.com/, which allows you to click a few buttons in a visual interface, together with the option to writing commands like `git commit` in a command line tool.

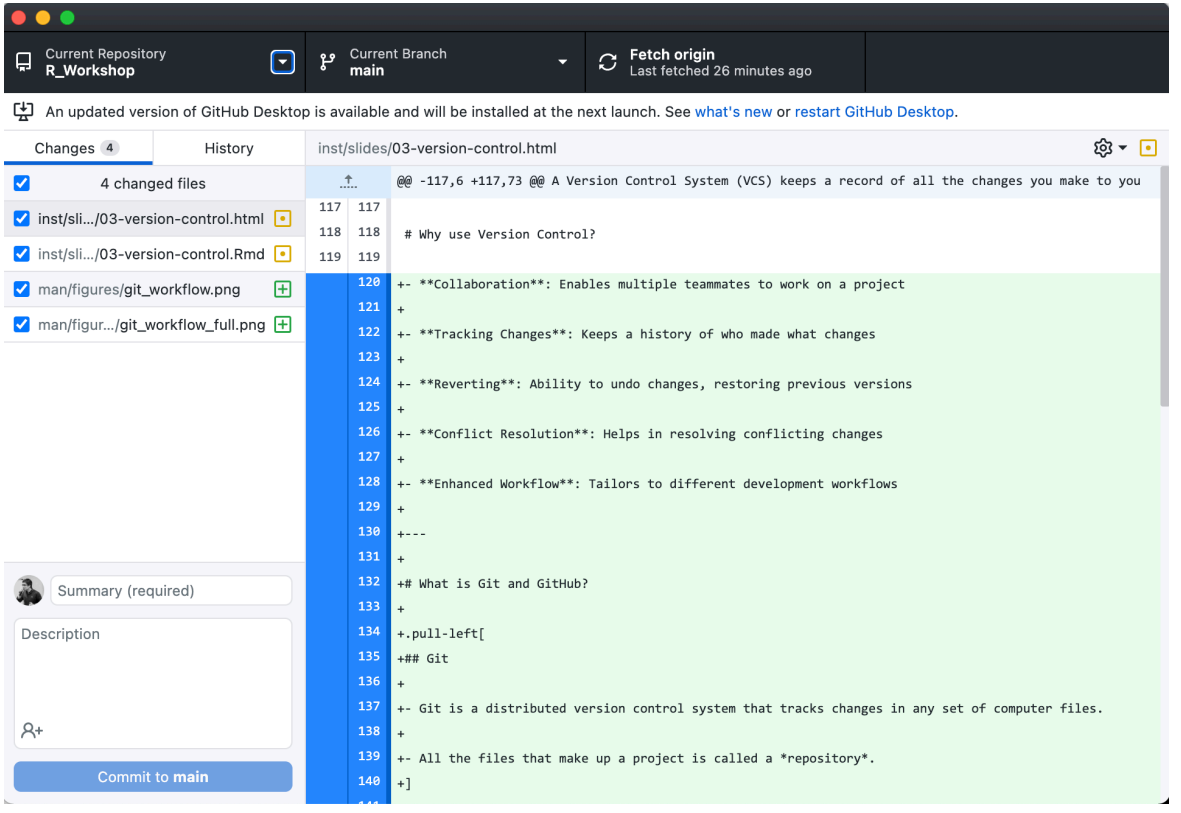
Git Workflow



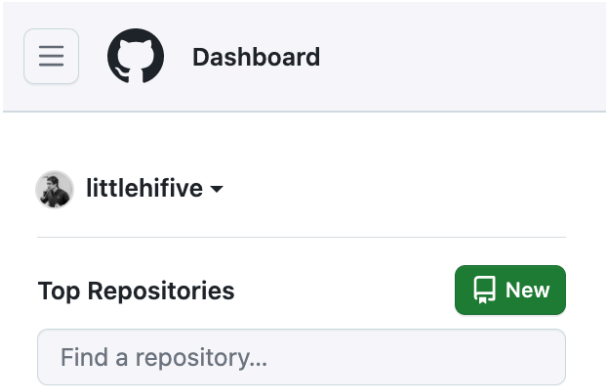
Source: https://dev.to/mollynem/git-github--workflow-fundamentals-5496

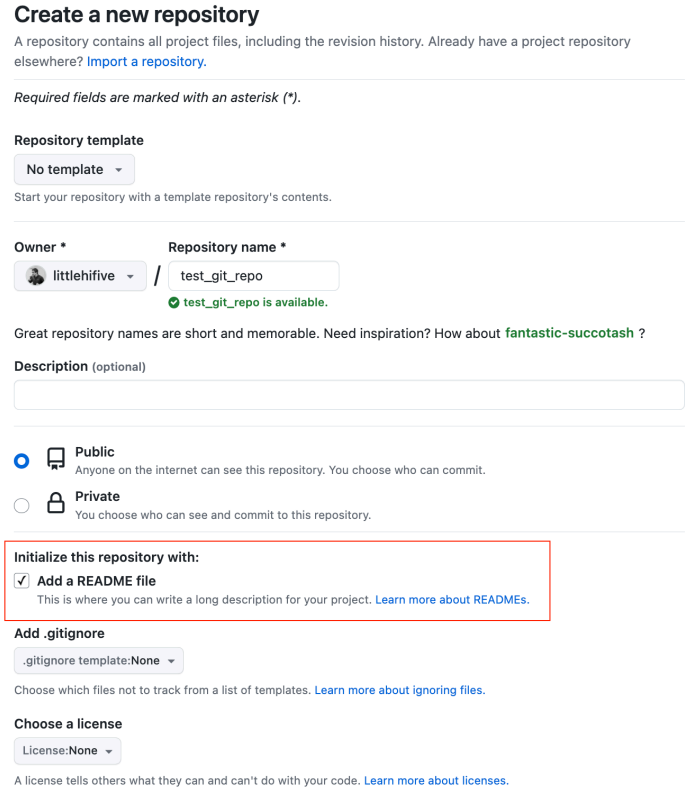
Using GitHub Desktop

1. Create an account on https://github.com/
2. Download GitHub Desktop on https://desktop.github.com/



1. Create your own repository

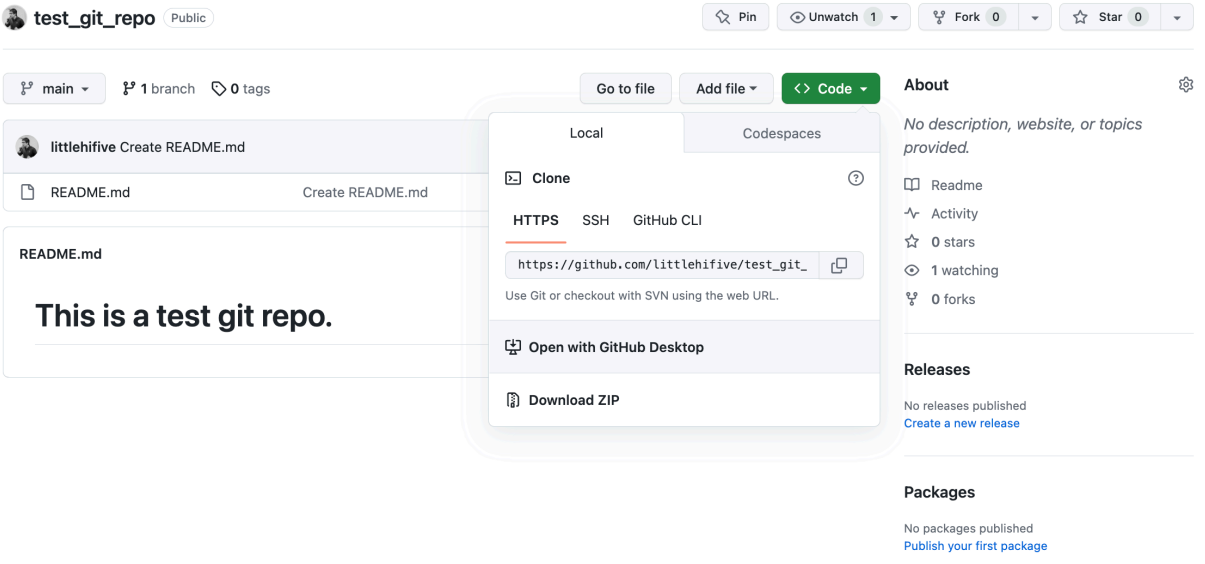


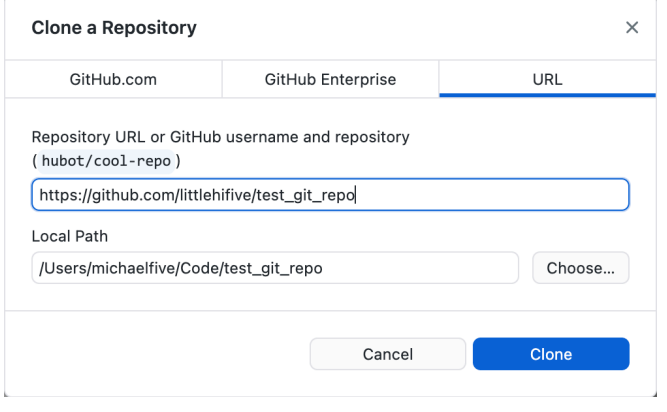


Make sure you check "Add a README file", otherwise the repo is empty and GitHub will ask you to add at least one file to the repo.

A README file is usually a markdown file (README.md) that is used for describing the repo.

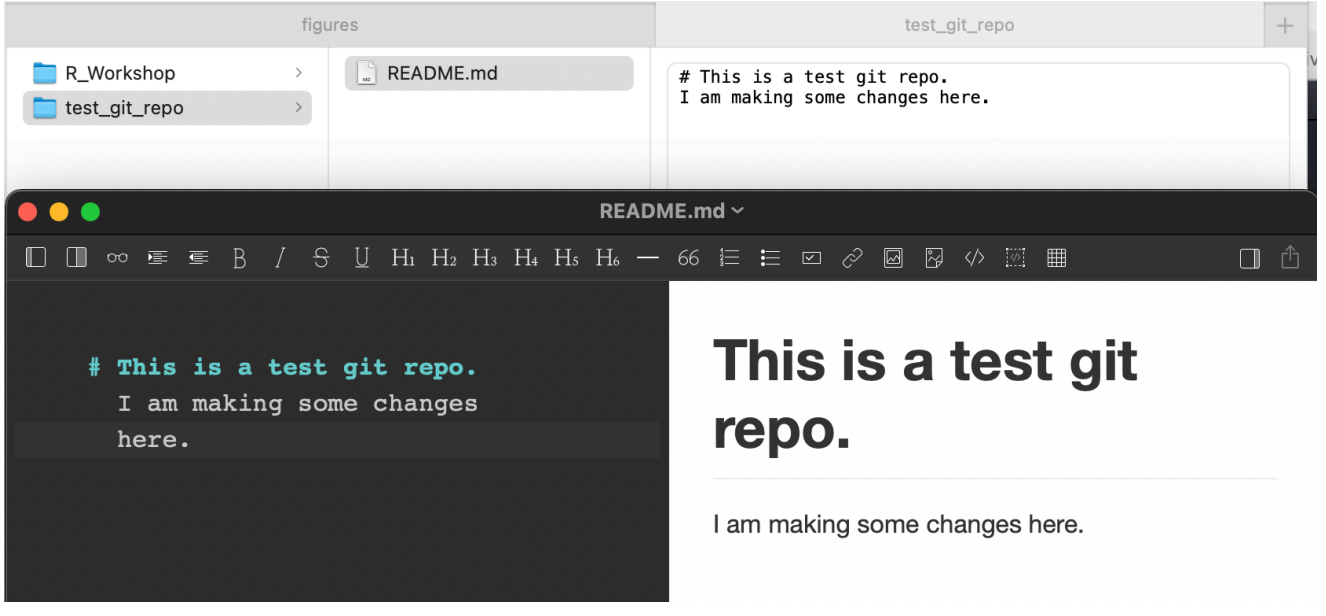
2. Clone the repository



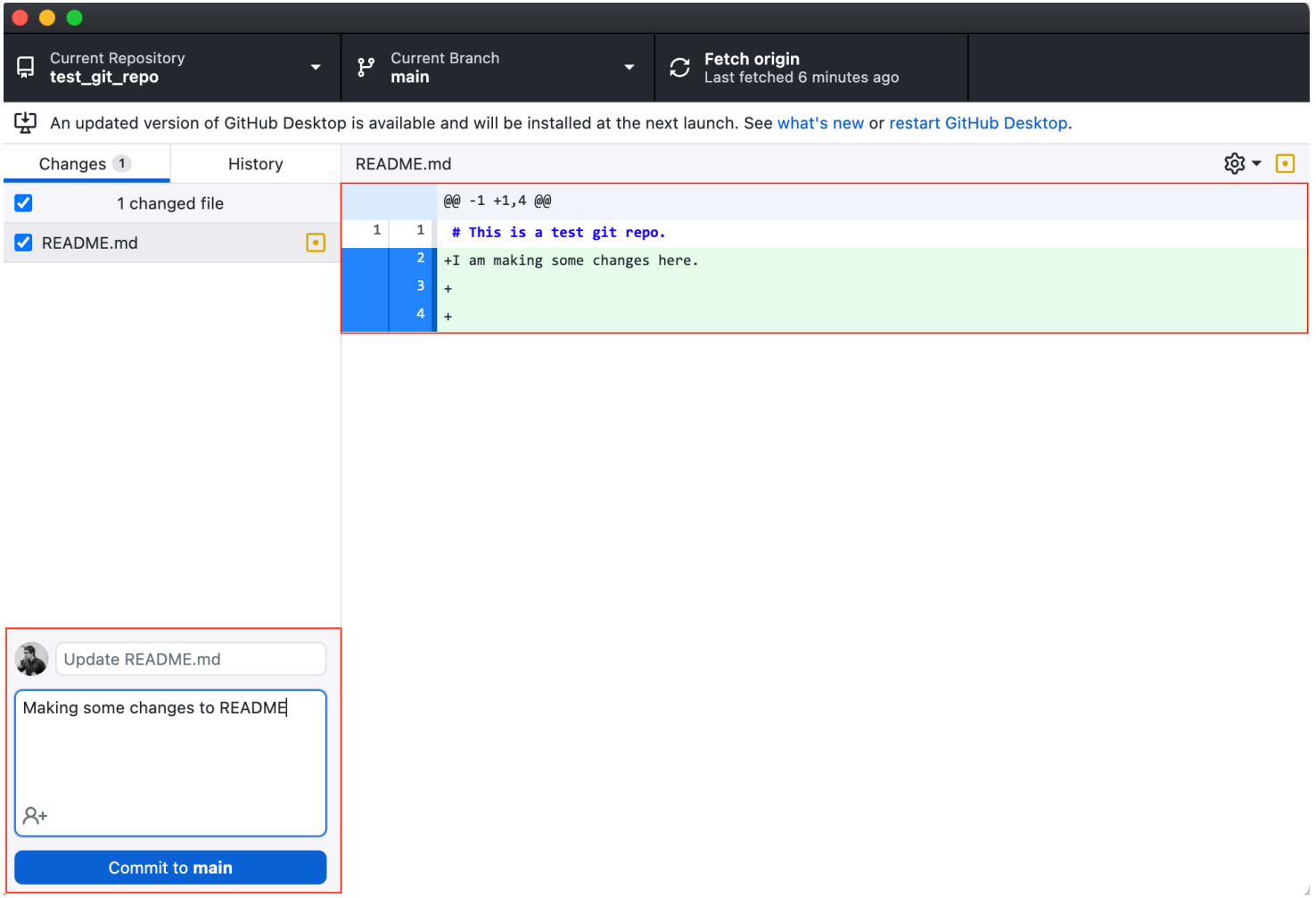


3. Make changes to a file

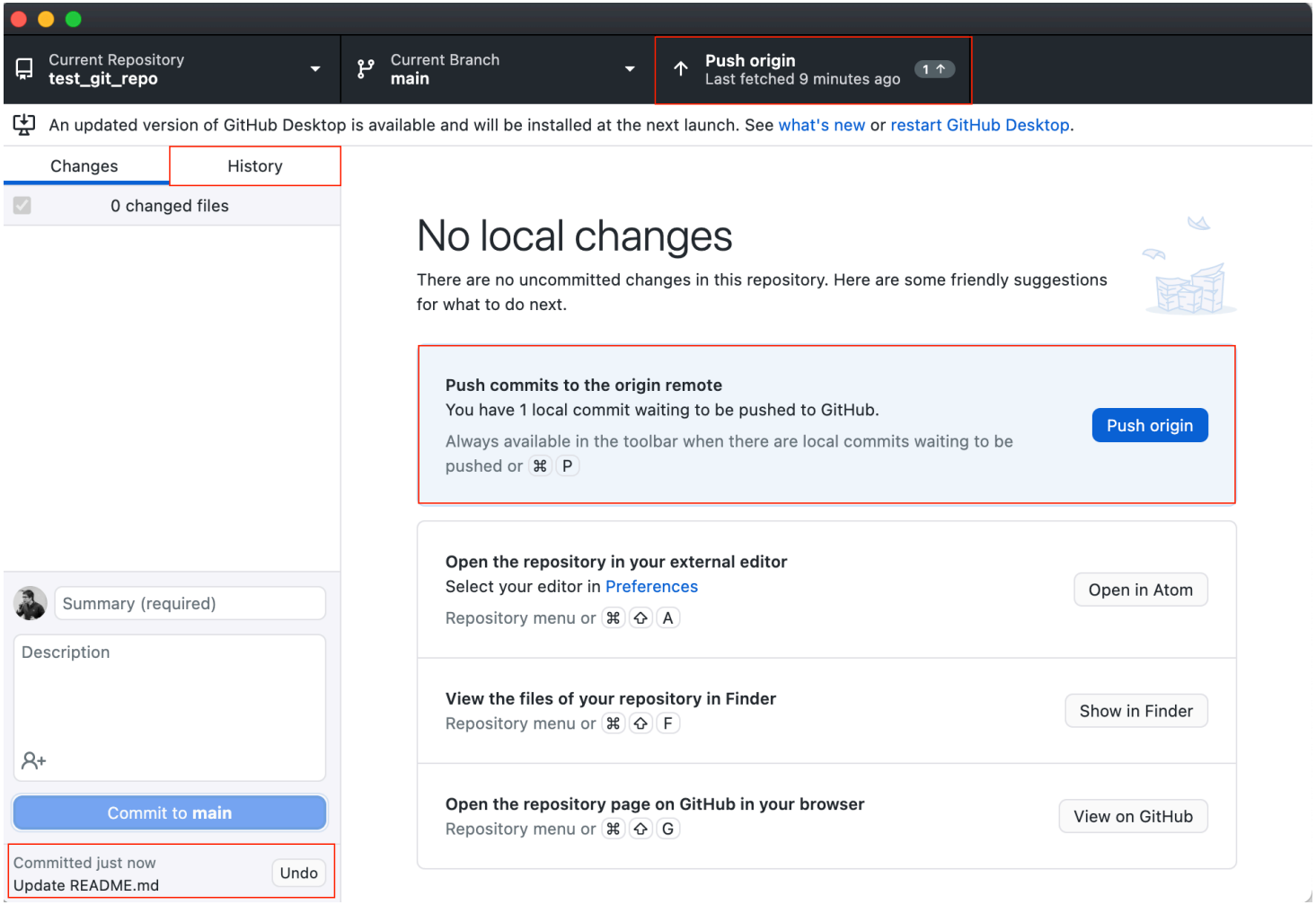
- Open README.md using any text editor on your computer
- Add a sentence and save the document



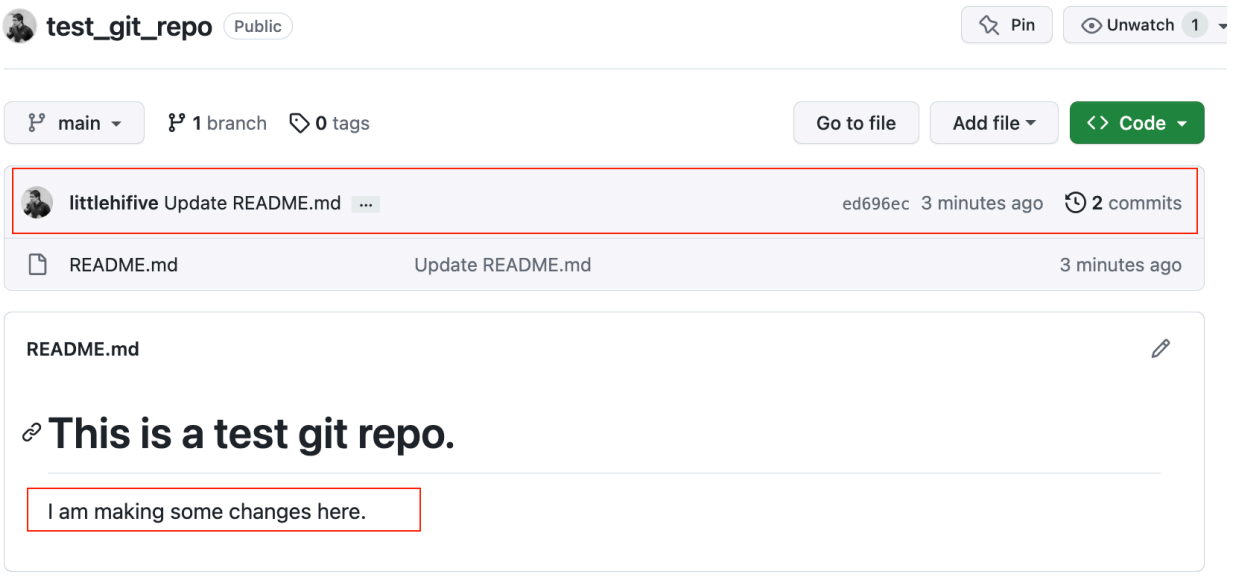
4. Commit changes to main



5. Push changes to origin

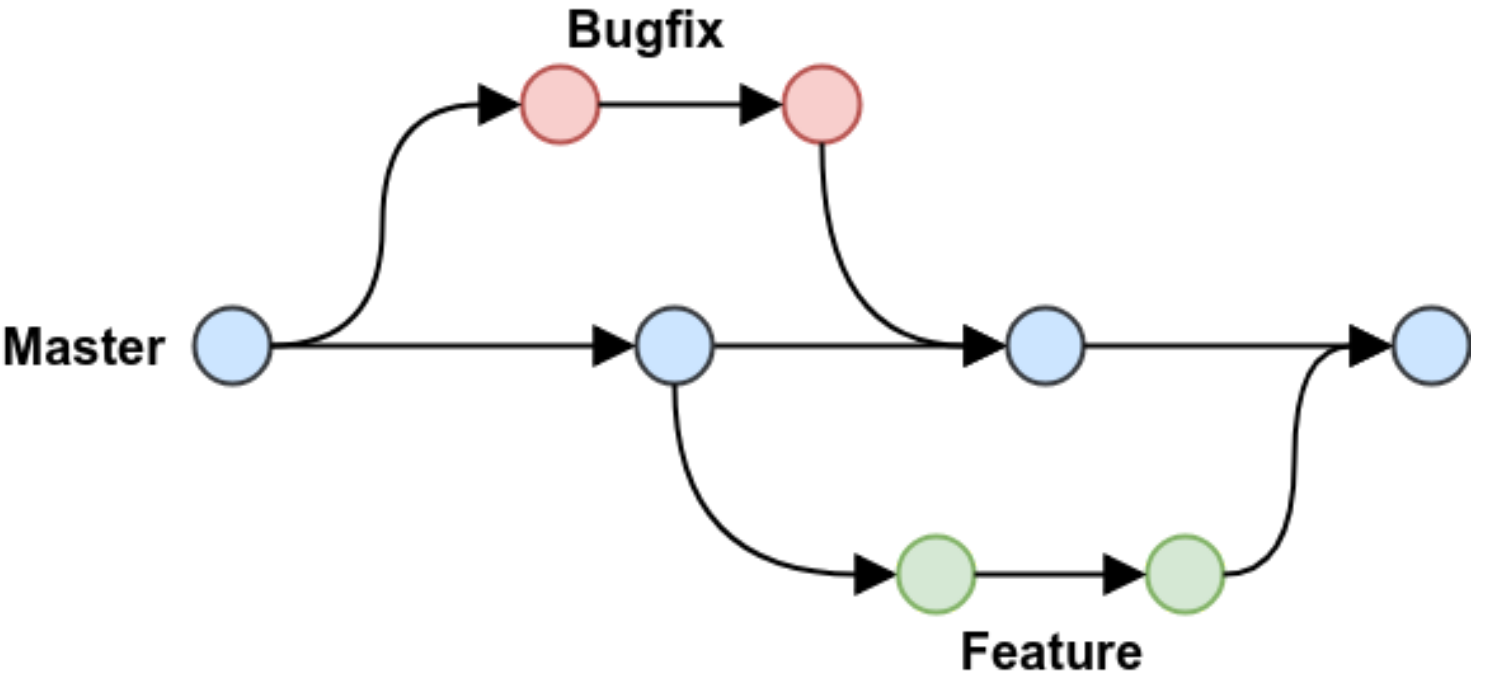


6. See changes on GitHub

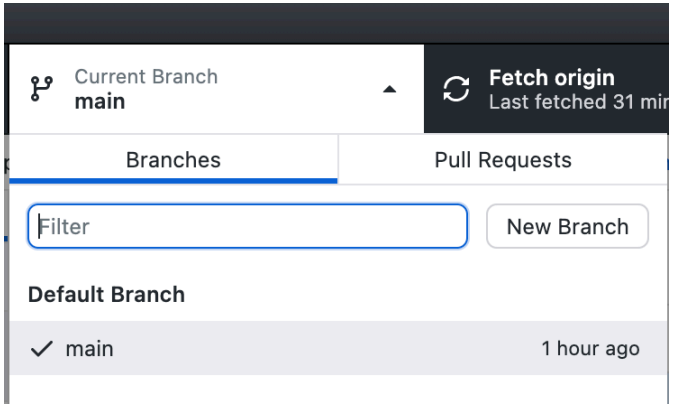


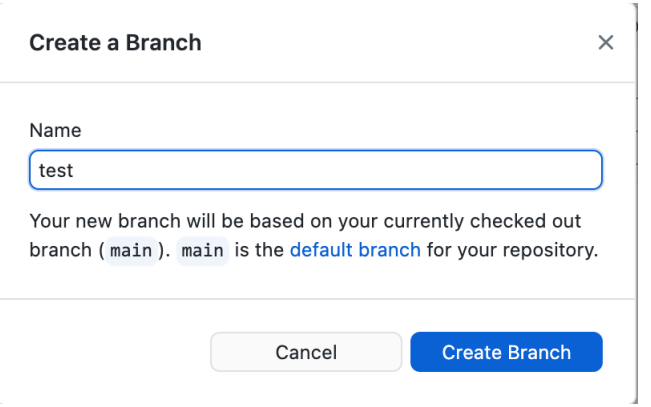
Branching

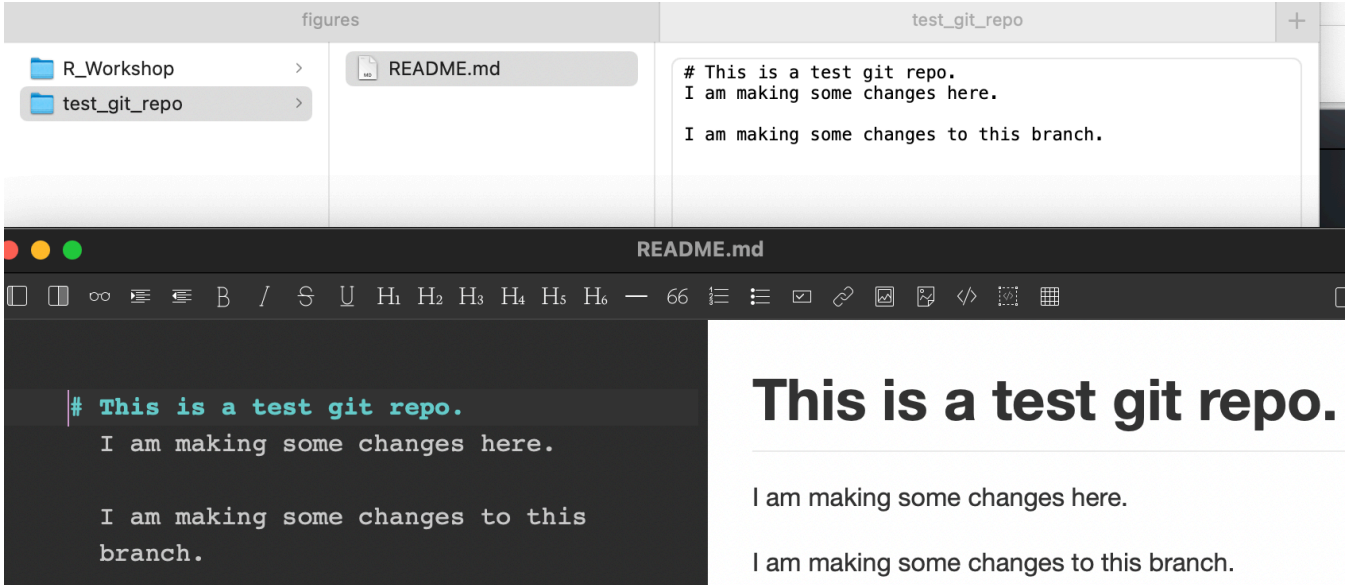
- Branching is about creating parallel changes without affecting the `main`.
- You can use `pull request` to merge branches.



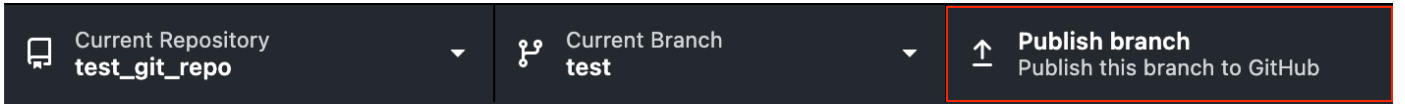
7. Create a branch

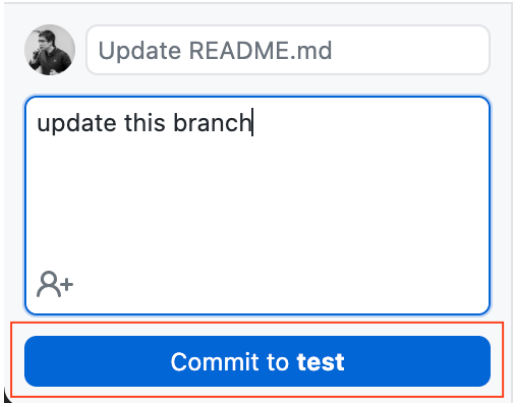


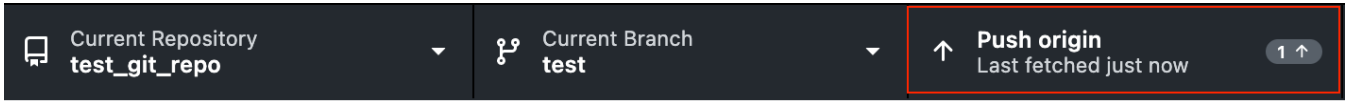




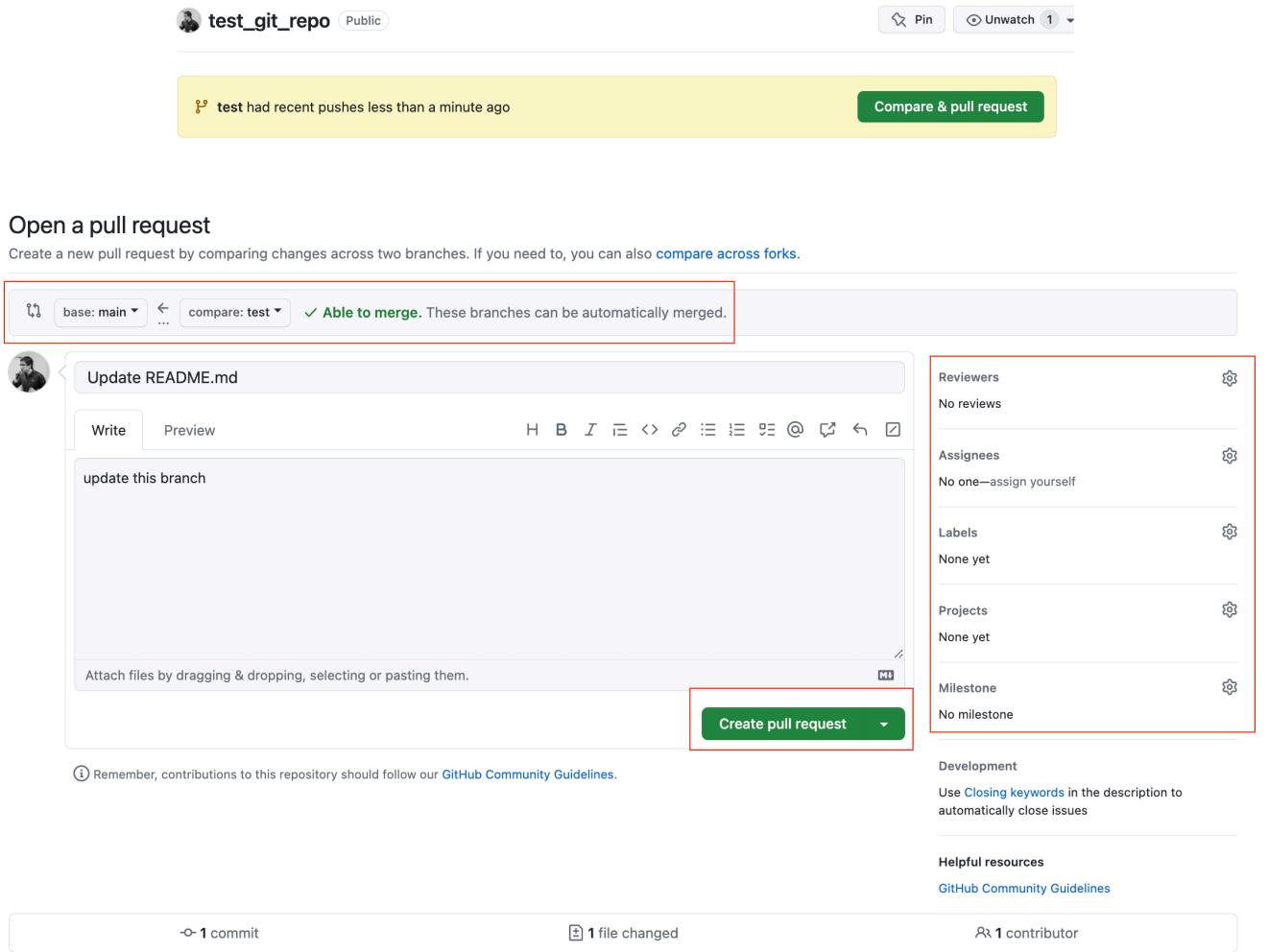
8. Publish the branch



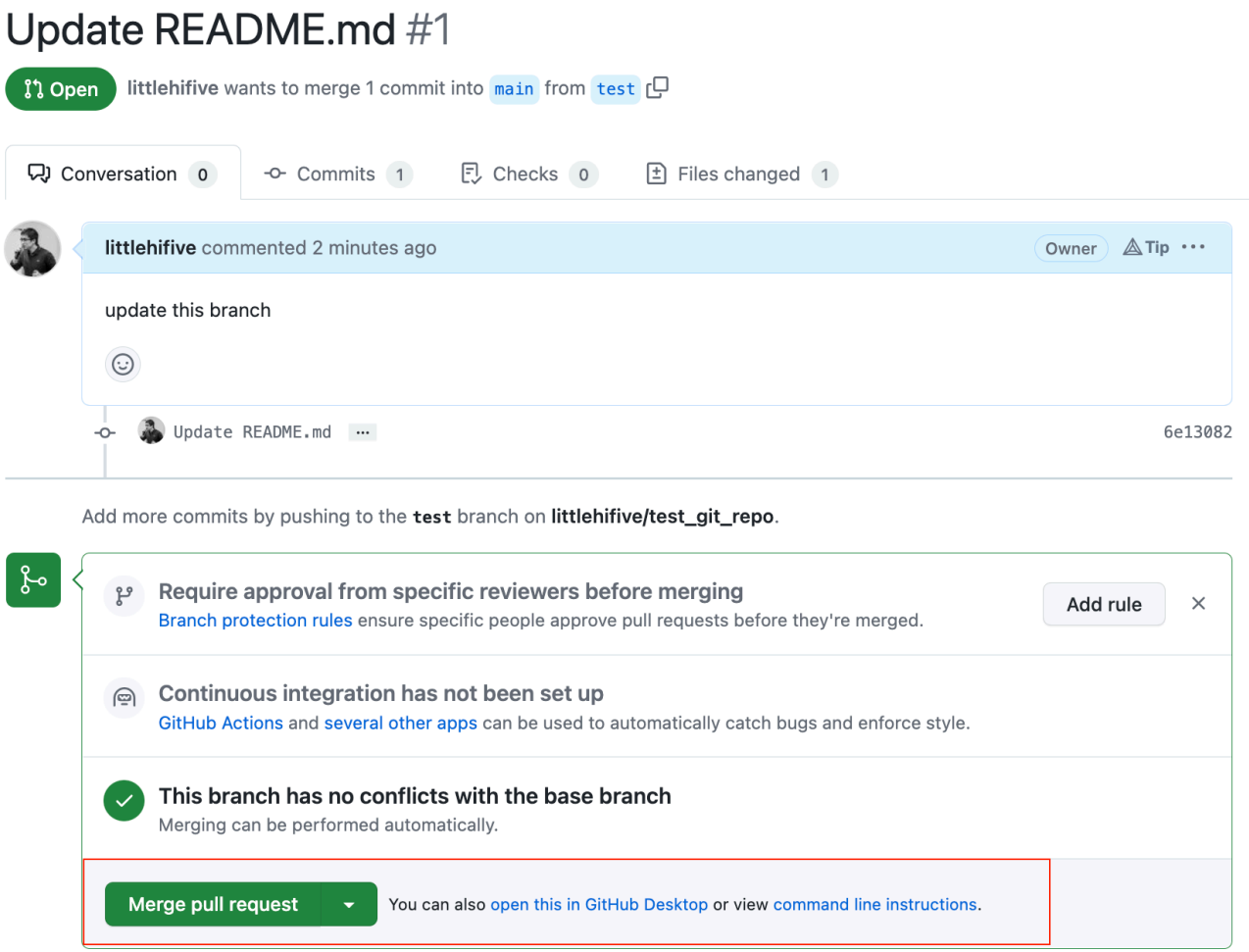




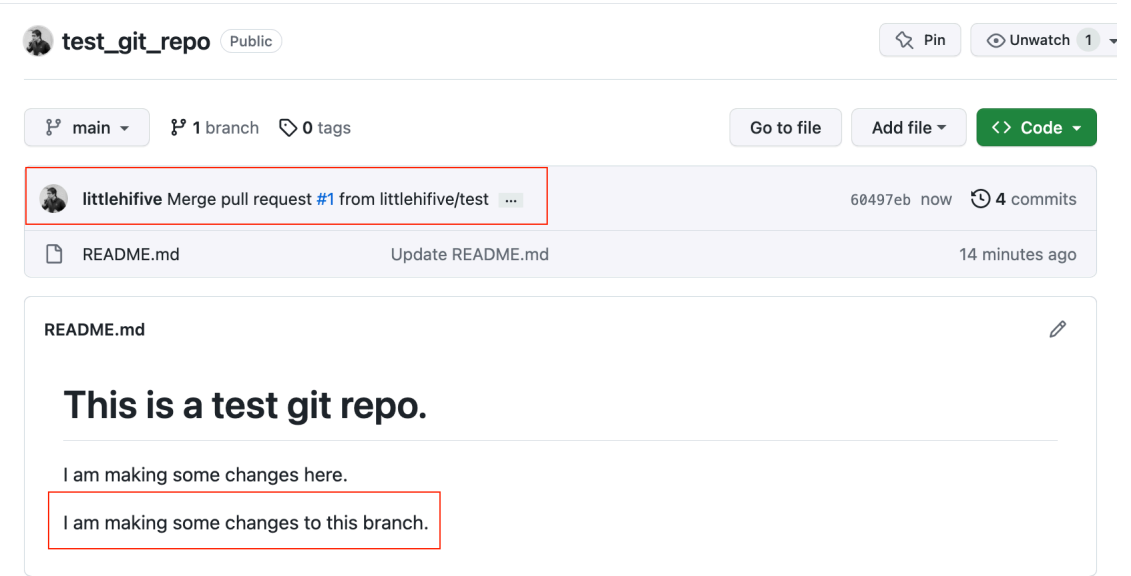
9. Pull request



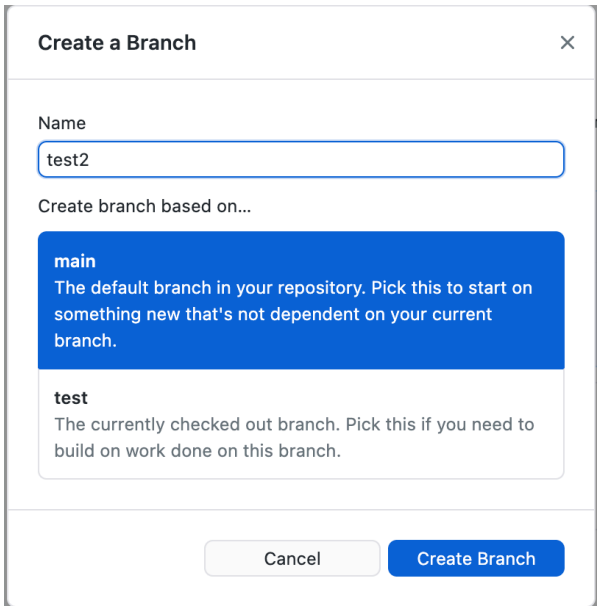
10. Merge pull request

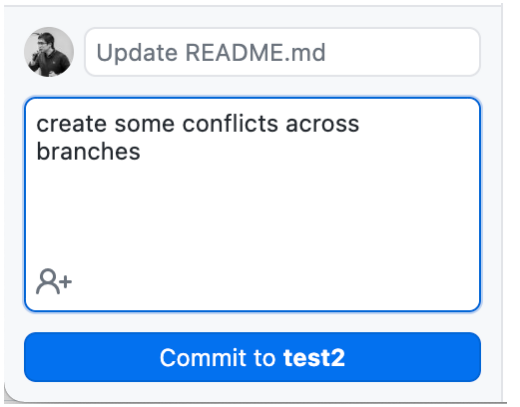


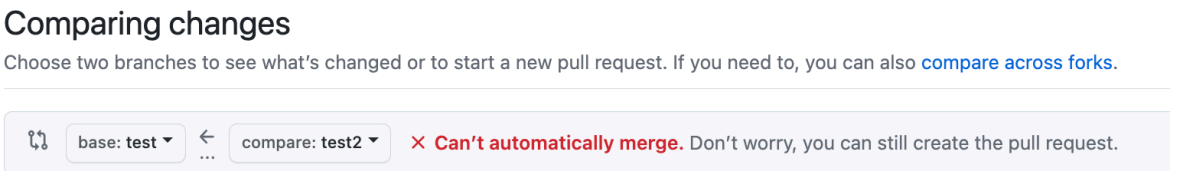
11. See changes on GitHub



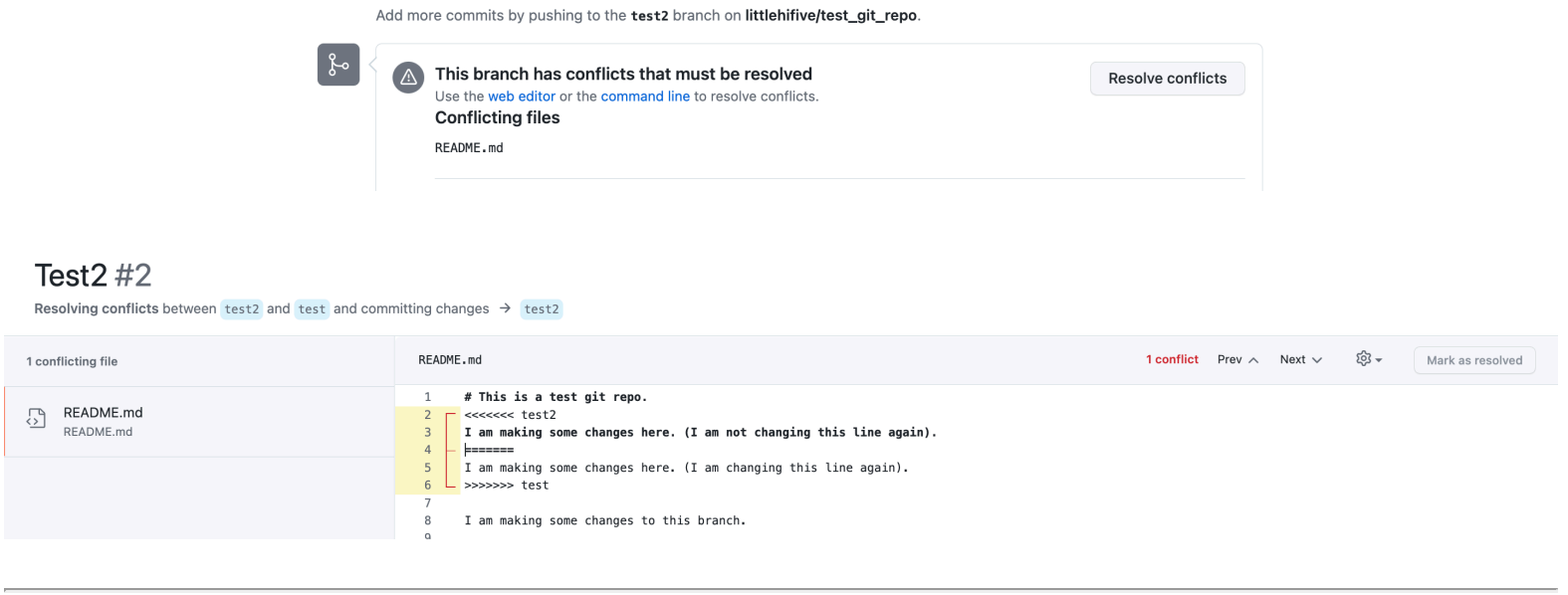
What if there are conflicts?







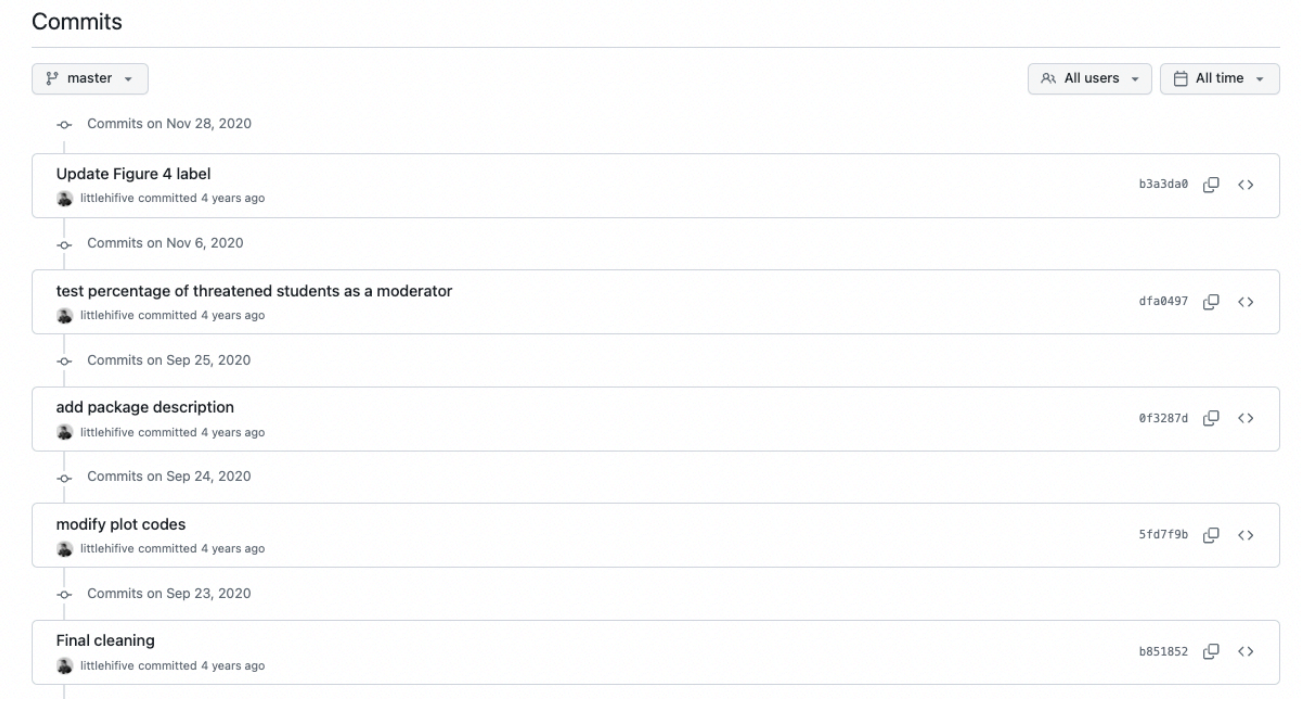
Resolving conflicts



- Depending on which programming language you are using, you can choose the code editor to resolve these conflicts on your own branch.
- Usually a more senior data manager checks code conflicts and make the executive decision to resolve conflicts and merge.

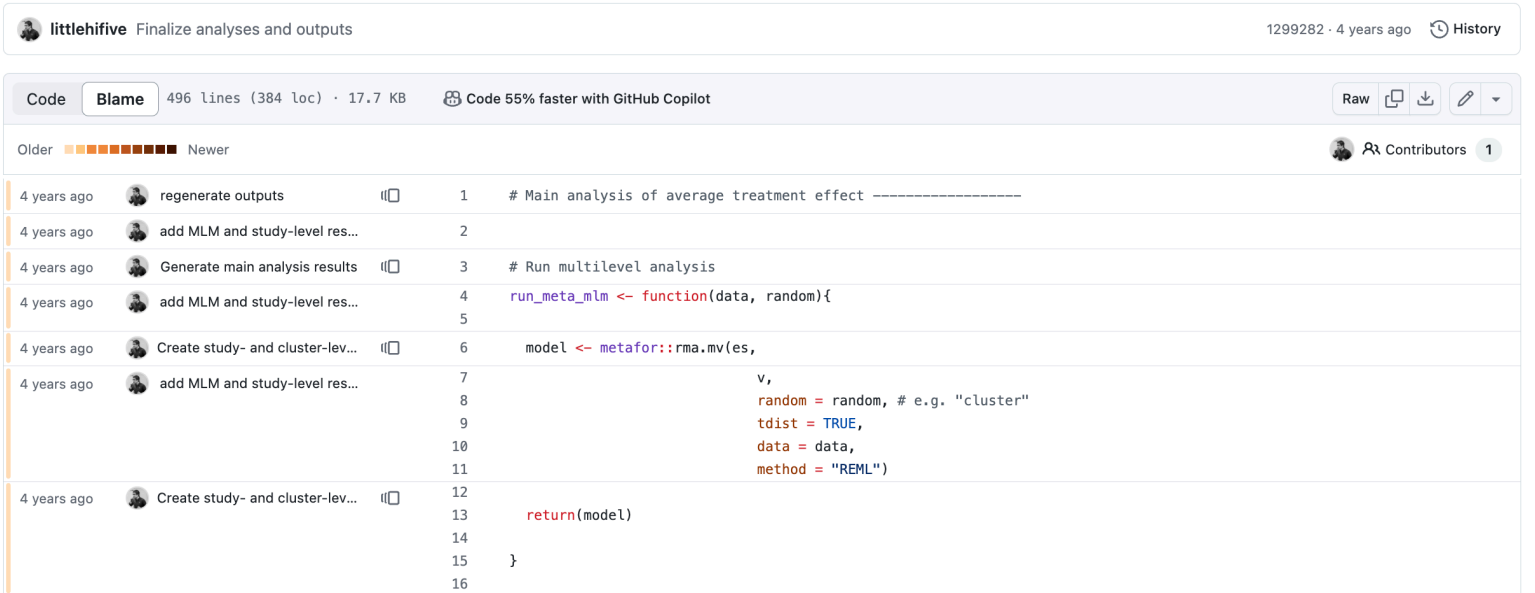
Why do this (even for one person or a small team)?

- You have a clean history of why certain changes are made to certain .do files or R script (i.e., commit history).



Why do this (even for one person or a small team)?

- You know who is responsible for what changes made to each line of code (i.e., `git blame`).



Why do this (even for one person or a small team)?

- You can go back in time and restore certain commits if recent commits don't work properly.
- Codes are stored separately from data files, so that codes are open-sourced and reproducible, while data files (on Dropbox) can be private unless made public.

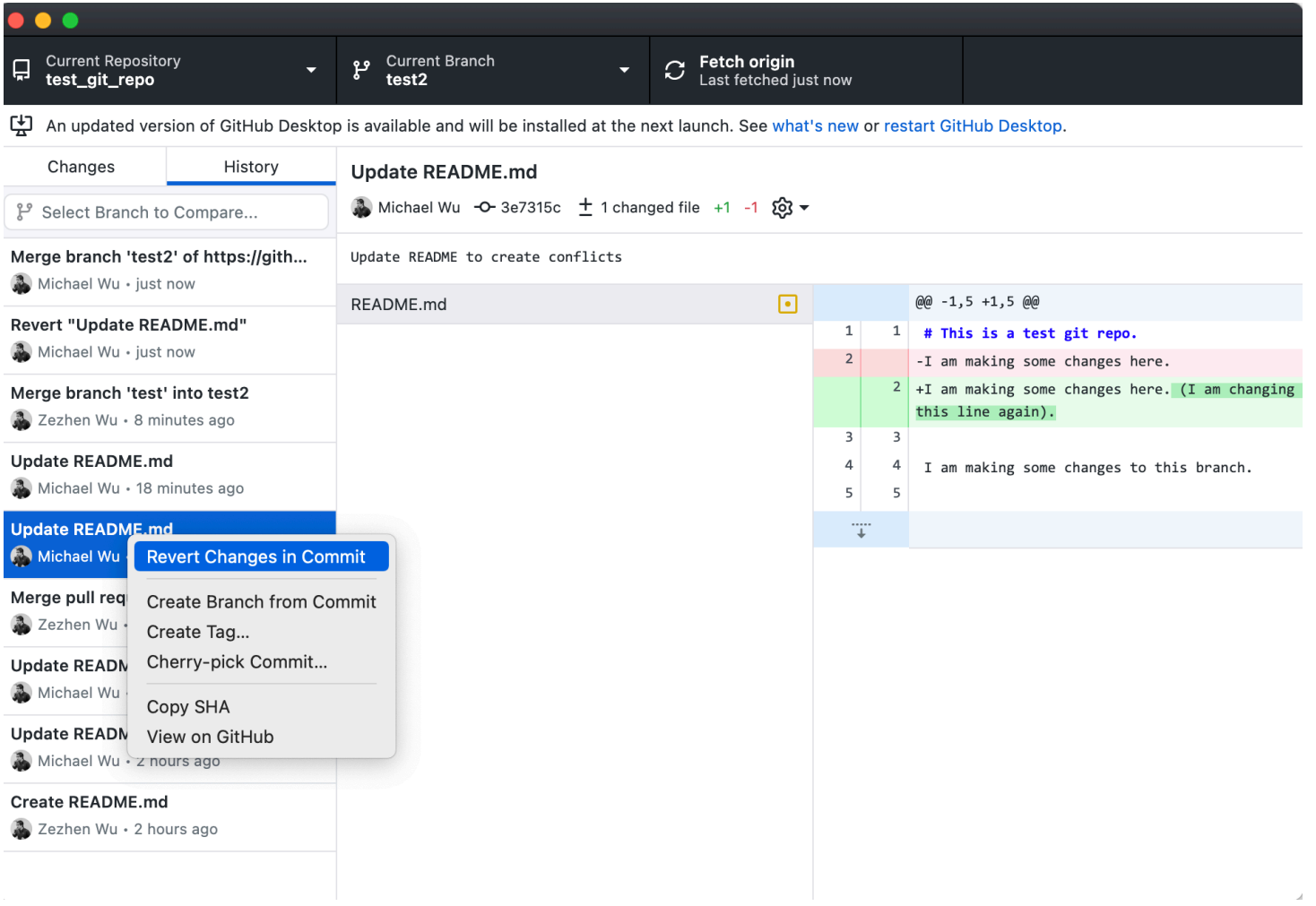
Do not use git together with cloud storage for the same project files!

Cloud storage services automatically sync files, which can lead to conflicts with Git's own version control mechanisms.

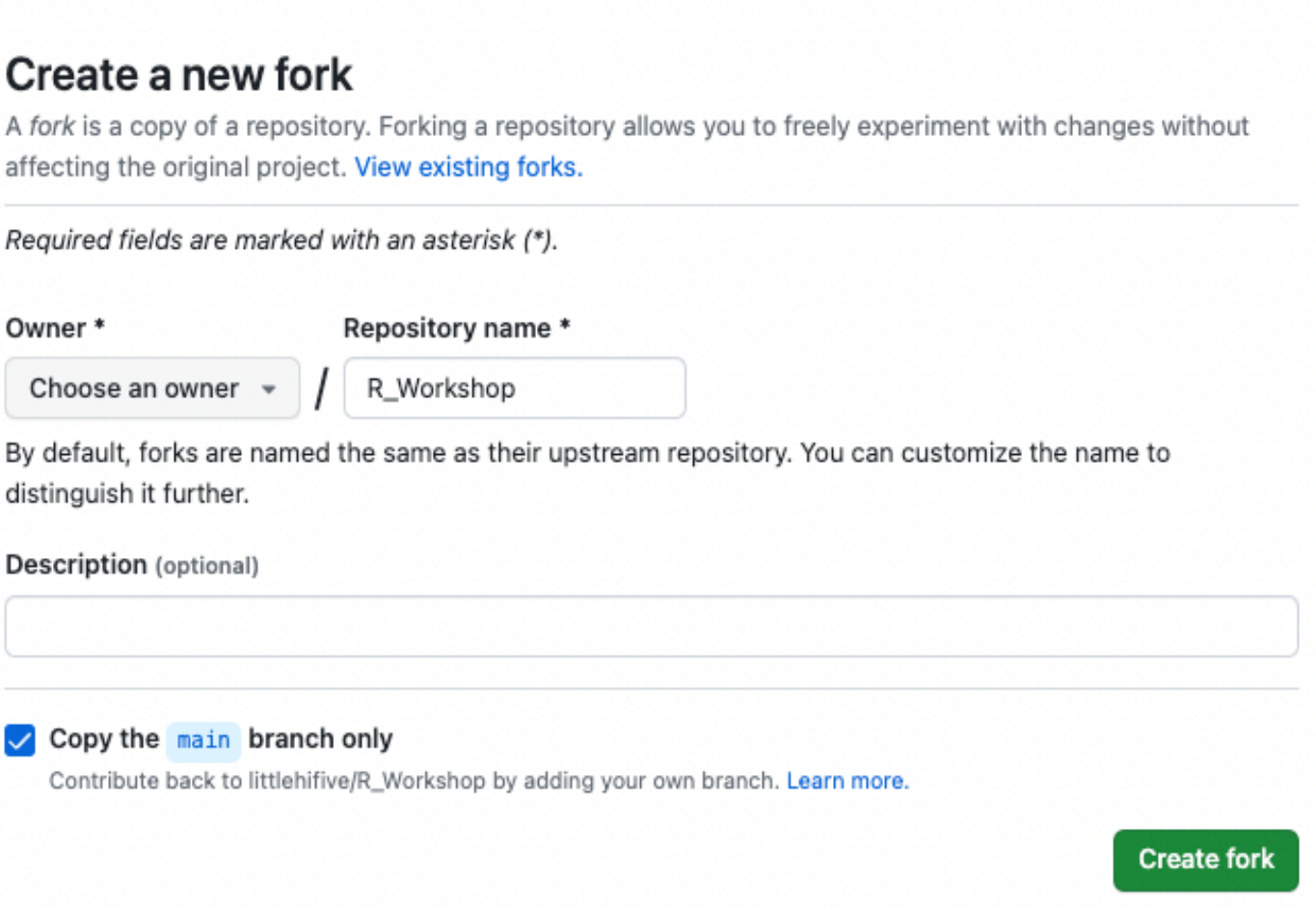
Recommended steps

- Use git and GitHub for version control with any code (including Stata .do files, R scripts, markdown files, R markdown files, etc.)
    - For instance, see the dashboard repository that Jake made for Youth Impact
    - Avoid putting any codes on cloud drives, especially when collaborating with others on coding
- DO NOT push data files to GitHub, store them on Dropbox/Google Drive instead
- Create centralized data-only folders on Dropbox, in lieu of project folders where data folders are sub-folders
- Create unified file and data variable naming rules across projects
- Include a README for each project and data folder
- Automate data cleaning as much as possible (check out the `targets` package)

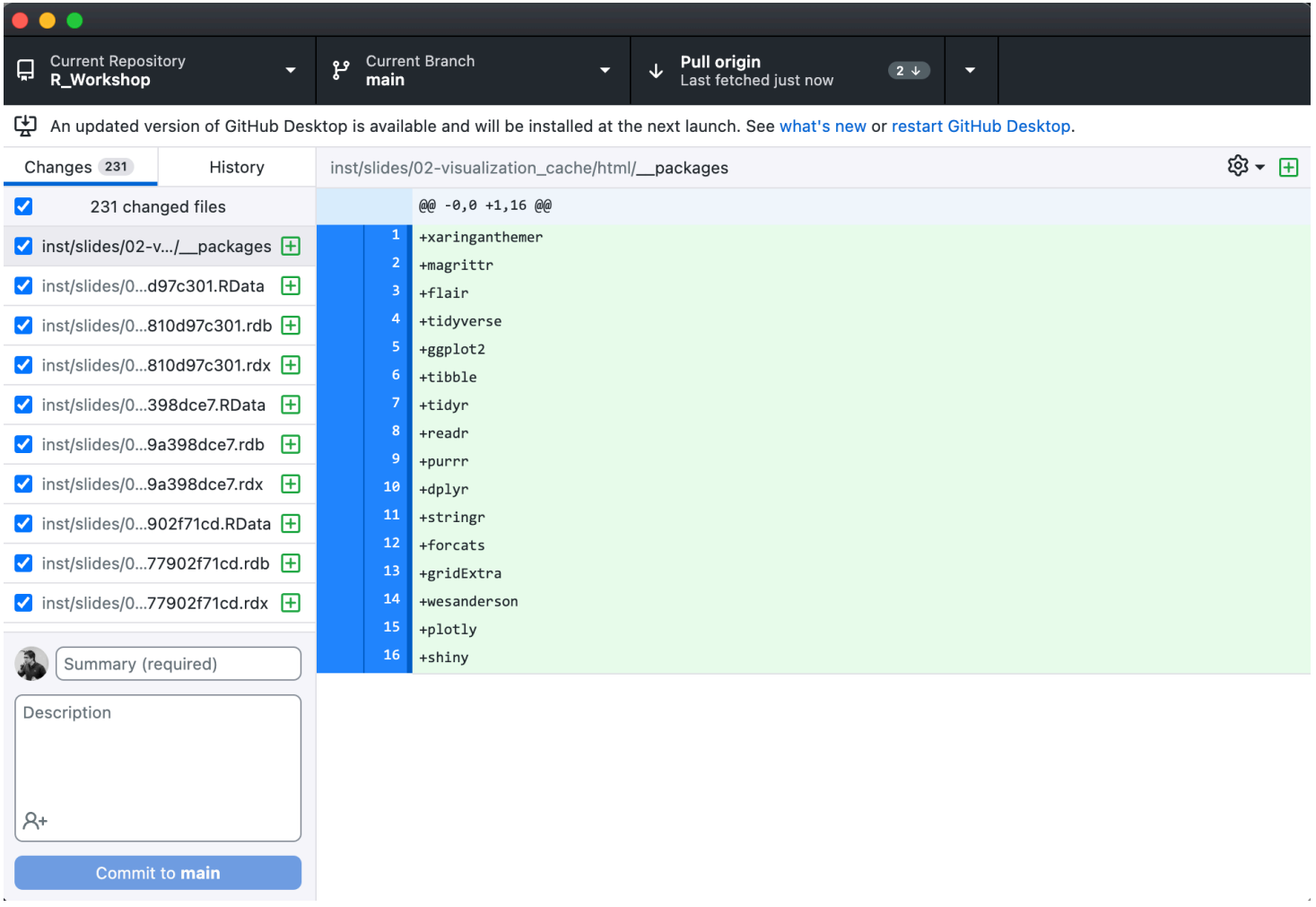
Bonus: Reverting a commit



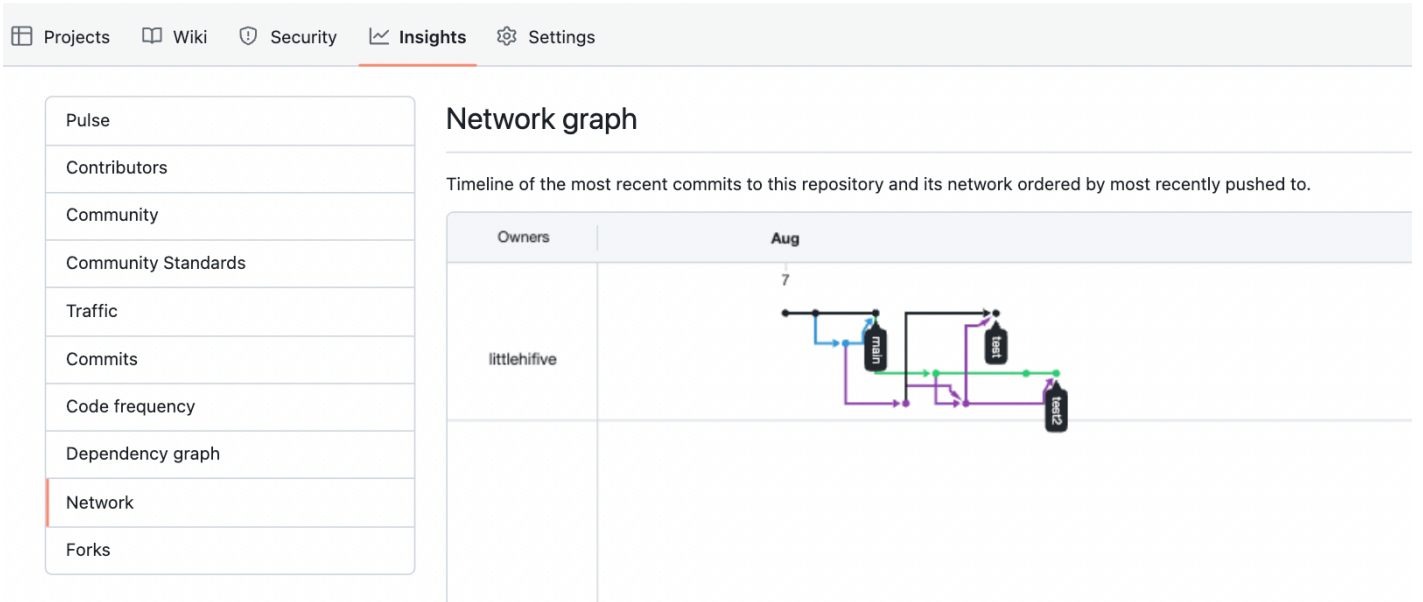
Bonus: Fork a project



Bonus: Pull from GitHub

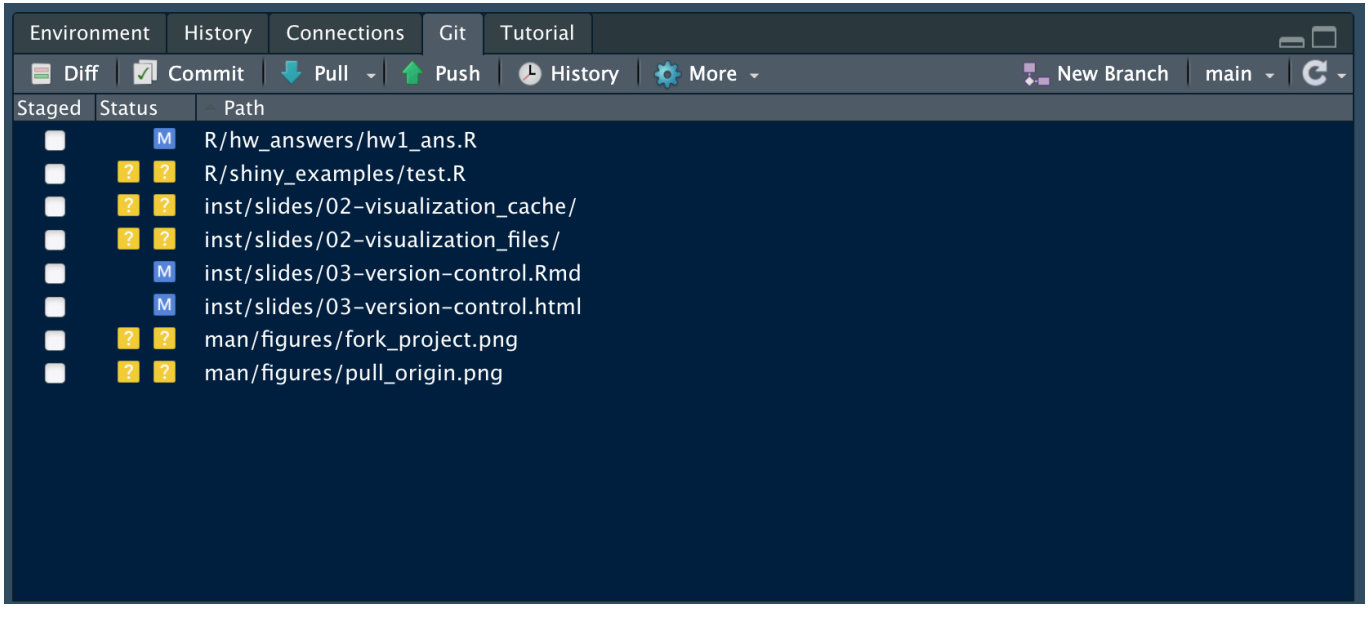


Bonus: Network graph



Bonus: Using git in R Studio

- You may need to configure R Studio to get the Git tab in an R project.
- See tutorials here and here.



Bonus: Using git for Stata do files

- Check out this full tutorial here.
- .do file is just another text file, so git is able to track changes in do files line by line.
- All the steps that we learned today (commit, push, pull, etc.) are the same for text files including Stata do files.

Bonus: An example data cleaning project

- See this github repository for how I structured a data cleaning project using `tidyverse` and `targets` and version controlled by git.
- A streamlined data cleaning project involves functional programming, which requires putting every little steps into little functions, little functions into bigger functions, and bigger functions into building blocks of a data pipeline.