



# Getting started with Git, GitHub & GitHub Desktop

Shirley Li, PhD  
Bioinformatician  
TTS Research Technology

Upcoming Workshop:

# *Best Practices for using **GPUs** in Jupyter for Data Science*

Wednesday, October 29, 2025

Register [HERE](#)

- Run **GPU-accelerated Jupyter notebooks** on the Tufts HPC cluster
- Fine-tune a **convolutional neural network (CNN)** as a hands-on example
- Learn best practices to **leverage Tufts GPU resources** efficiently



# Overview

- 1. Git & GitHub & Version Control Basics**
- 2. Navigating GitHub Desktop**
- 3. Resolving Merge Conflicts**
- 4. Best Practices for GitHub Usage**

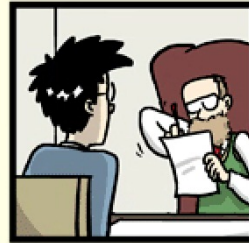
**Let's talk about version control**

**What's your biggest challenge keeping track of file versions?**

# "FINAL".doc



FINAL.doc!



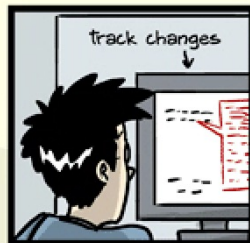
FINAL\_rev.2.doc



FINAL\_rev.6.COMMENTS.doc



FINAL\_rev.8.comments5.  
CORRECTIONS.doc



FINAL\_rev.18.comments7.  
corrections9.MORE.30.doc



FINAL\_rev.22.comments49.  
corrections.10.##\$%WHYDID  
ICOMETOGRADSCHOOL?????.doc

JORGE CHAM © 2012

# Version Control

- **Version control is a record of who make changes to what, and when they did it.**
- **We can always undo.**
- **Easier for collaboration without overwriting.**
- **A key skill in code & data management!**

# What is Git



- A version control system.
- Manage source code changes (changes to files)
- Two key features: Commit and Branches.
- **With Git, you can easily roll back to older code snapshots (commits) or develop new features without breaking production code.**

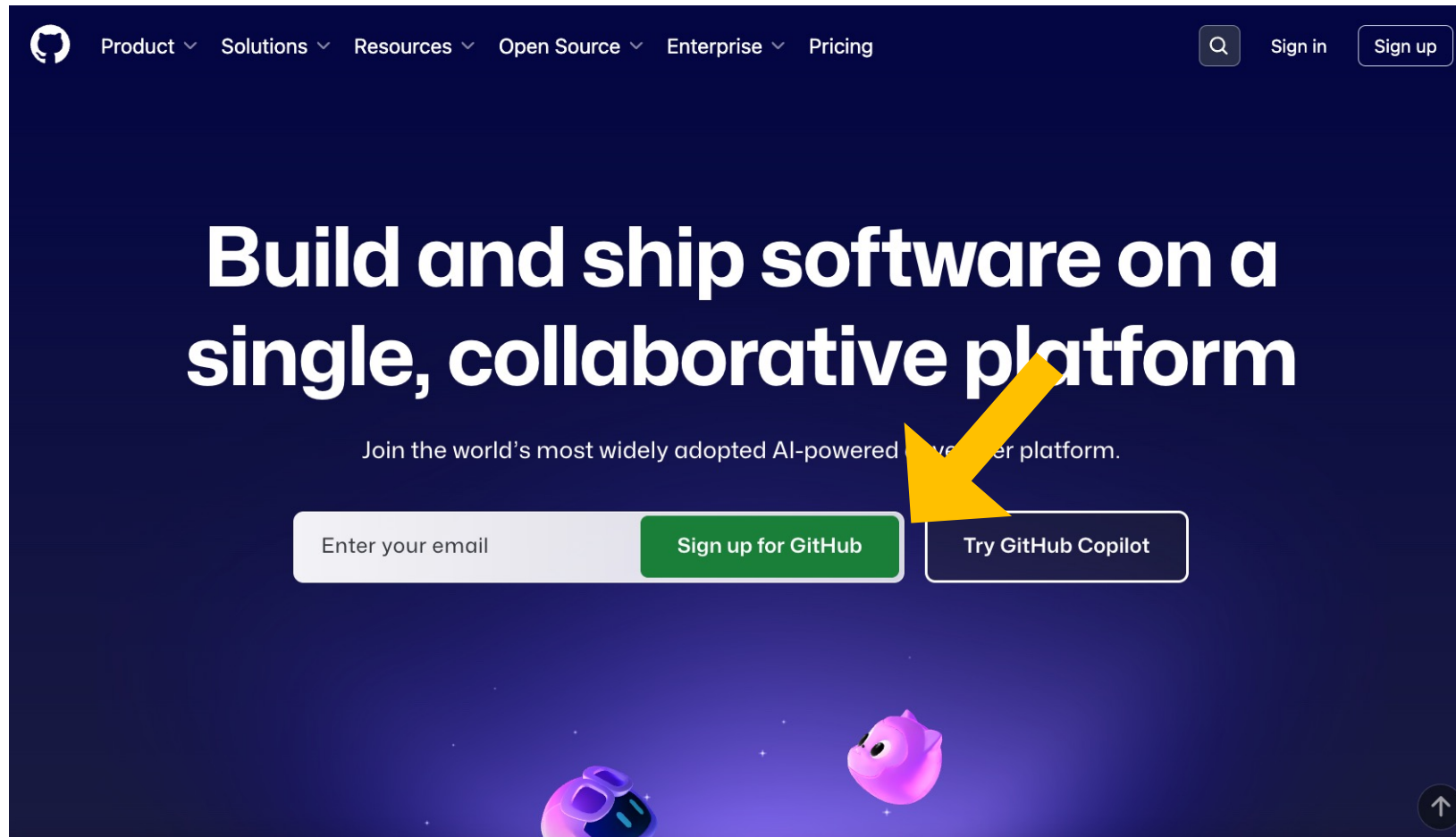
# What is GitHub



- A cloud Git repository & services provider.
- Code management & collaborative development.
- It can handle all the versioning and allows multiple people to collaborate on the same project.
- **Repositories support version control capabilities through Git.**
- **Graphical User Interface, beginner friendly**

# Sign up for GitHub account

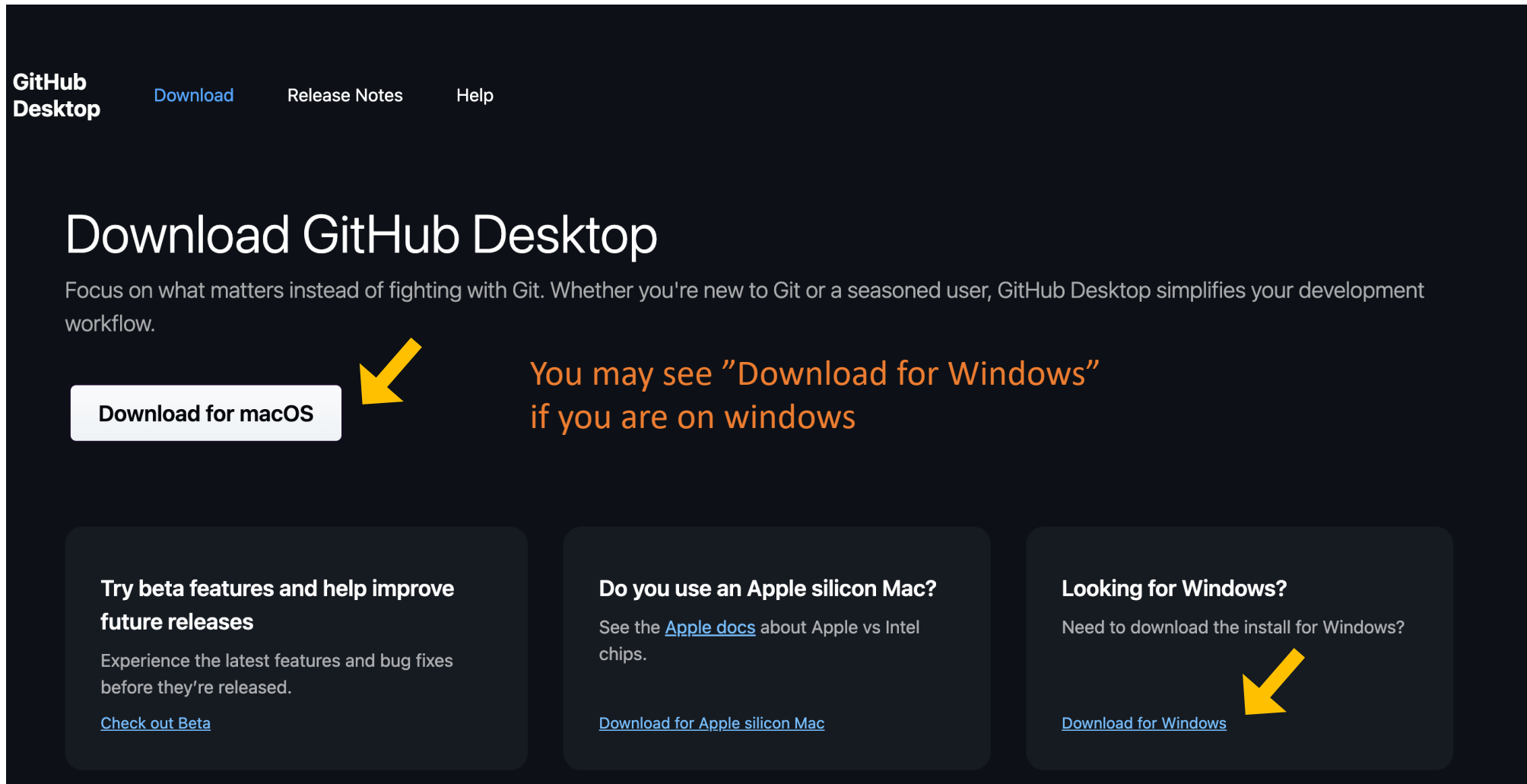
<https://github.com/>



# What is GitHub Desktop

- A user-friendly app for managing Git and GitHub projects.
- Simplifies version control with a visual interface.
- Supports cloning, committing, branching, and merging.
- Great for beginners and those who prefer a GUI over the command line.
- Works on macOS and Windows.
- Helps sync local changes with remote GitHub repositories.

<https://desktop.github.com/download/>



The screenshot shows the GitHub Desktop download page. At the top left is the 'GitHub Desktop' logo. To its right are links for 'Download', 'Release Notes', and 'Help'. The main heading is 'Download GitHub Desktop'. Below it is a paragraph: 'Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.' There are three primary download buttons: 'Download for macOS', 'Download for Windows', and 'Download for Apple silicon Mac'. A yellow arrow points to the 'Download for macOS' button. Another yellow arrow points to the 'Download for Windows' link, which is located in a box titled 'Looking for Windows?'. Below the main heading is a section with three cards. The first card is titled 'Try beta features and help improve future releases' and contains a link 'Check out Beta'. The second card is titled 'Do you use an Apple silicon Mac?' and contains a link 'Download for Apple silicon Mac'. The third card is titled 'Looking for Windows?' and contains a link 'Download for Windows'.

GitHub Desktop

[Download](#) [Release Notes](#) [Help](#)

# Download GitHub Desktop

Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

[Download for macOS](#)

You may see "Download for Windows" if you are on windows

[Download for Windows](#)

[Download for Apple silicon Mac](#)

[Check out Beta](#)

**Try beta features and help improve future releases**

Experience the latest features and bug fixes before they're released.

**Do you use an Apple silicon Mac?**

See the [Apple docs](#) about Apple vs Intel chips.

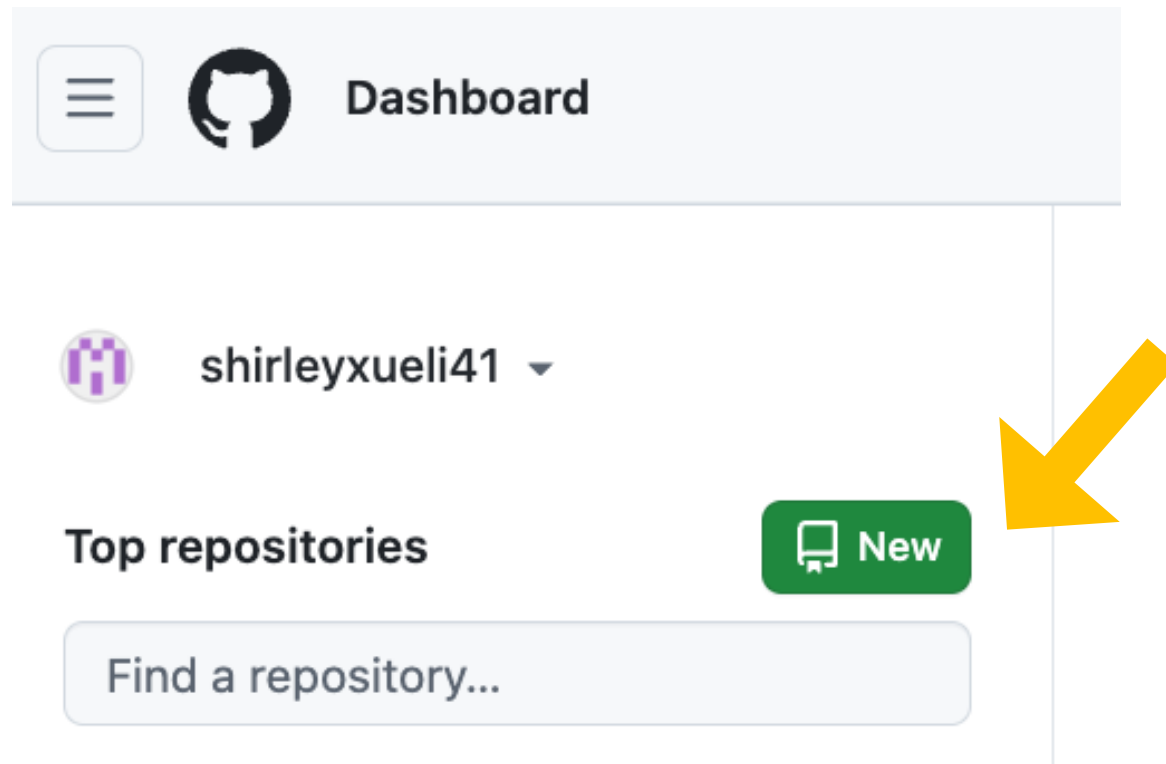
**Looking for Windows?**

Need to download the install for Windows?

# Let's create a GitHub repo

# Create a GitHub Repo

In the browser, go to <https://github.com/>




# Enter the details

## Create a new repository

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

Required fields are marked with an asterisk (\*).

Owner \*

 shirleyxueli41 ▾

Repository name \*

githubtraining

✓ githubtraining is available.

Name of the repo, no space

Great repository names are short and memorable. Need inspiration? How about **cautious-fiesta** ?

Description (optional)

- Public: everyone can see
- Private: Only you and your collaborator can see



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.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Good practice: always create a README file and write the description about the repo


Create repository


 **githubtraining** Public

[Pin](#) [Unwatch](#) 1

[main](#) 1 Branch 0 Tags


Go to file t + [Code](#)

 **shirleyxueli41** Initial commit b1449f6 · now [1 Commit](#)

 README.md

Initial commit

now

[README](#) 

# githubtraining

# Getting started with GitHub Desktop App

<https://docs.github.com/en/desktop/overview/getting-started-with-github-desktop>

- Installing and authenticating
- Configuring and customizing GitHub Desktop
- Contributing to projects with GitHub Desktop

# **.git hidden folder**

**.git folder is created after you initiate a repository**

**.git contains all information required for version control.**

**Mac users, to view hidden folders and files:**

**Shift + Command + .**

# GitHub



a central hub for stored code, allowing team members to push and pull changes



# Git Command

## BASIC GIT COMMANDS EVERY DEV MUST KNOW



# remote



## git clone

# local



# remote



`.git`

# local



`.git`

**Make changes locally**

# remote



# local



**git commit**

# remote



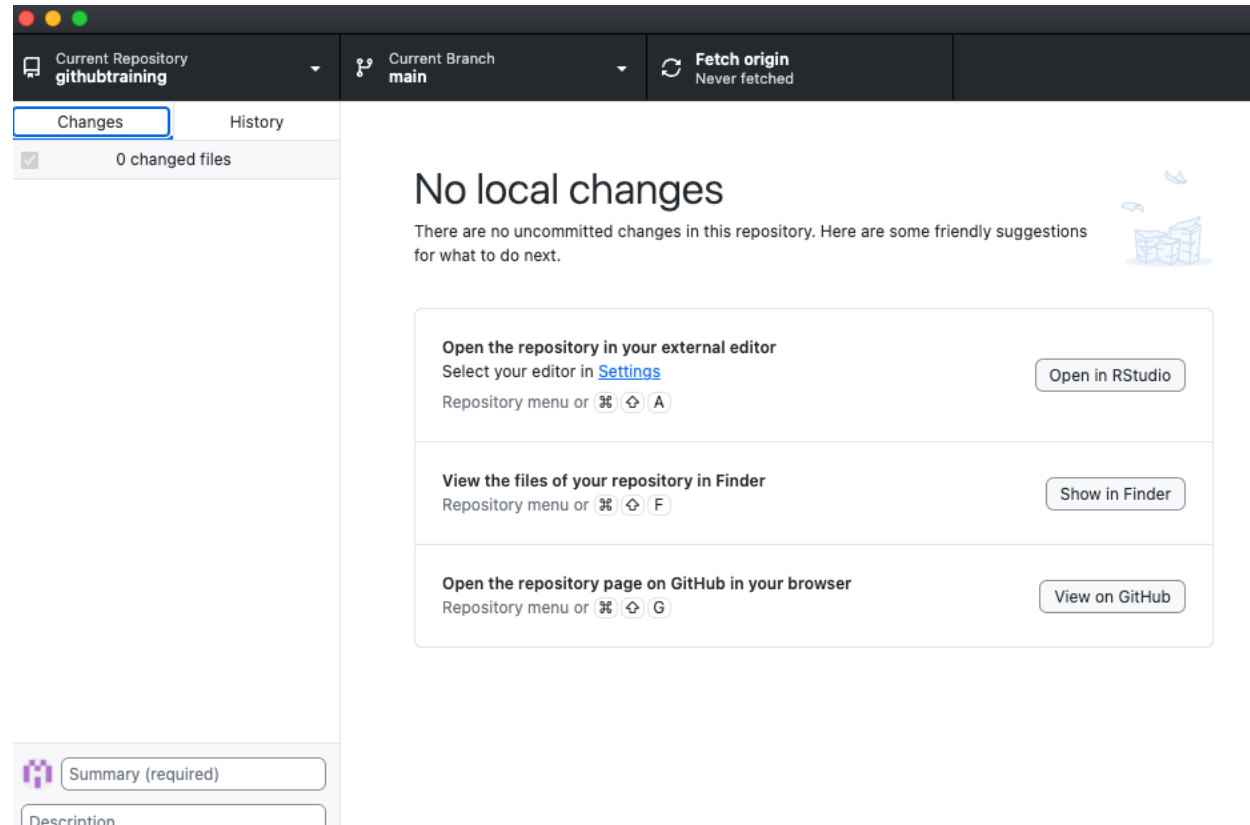
# local



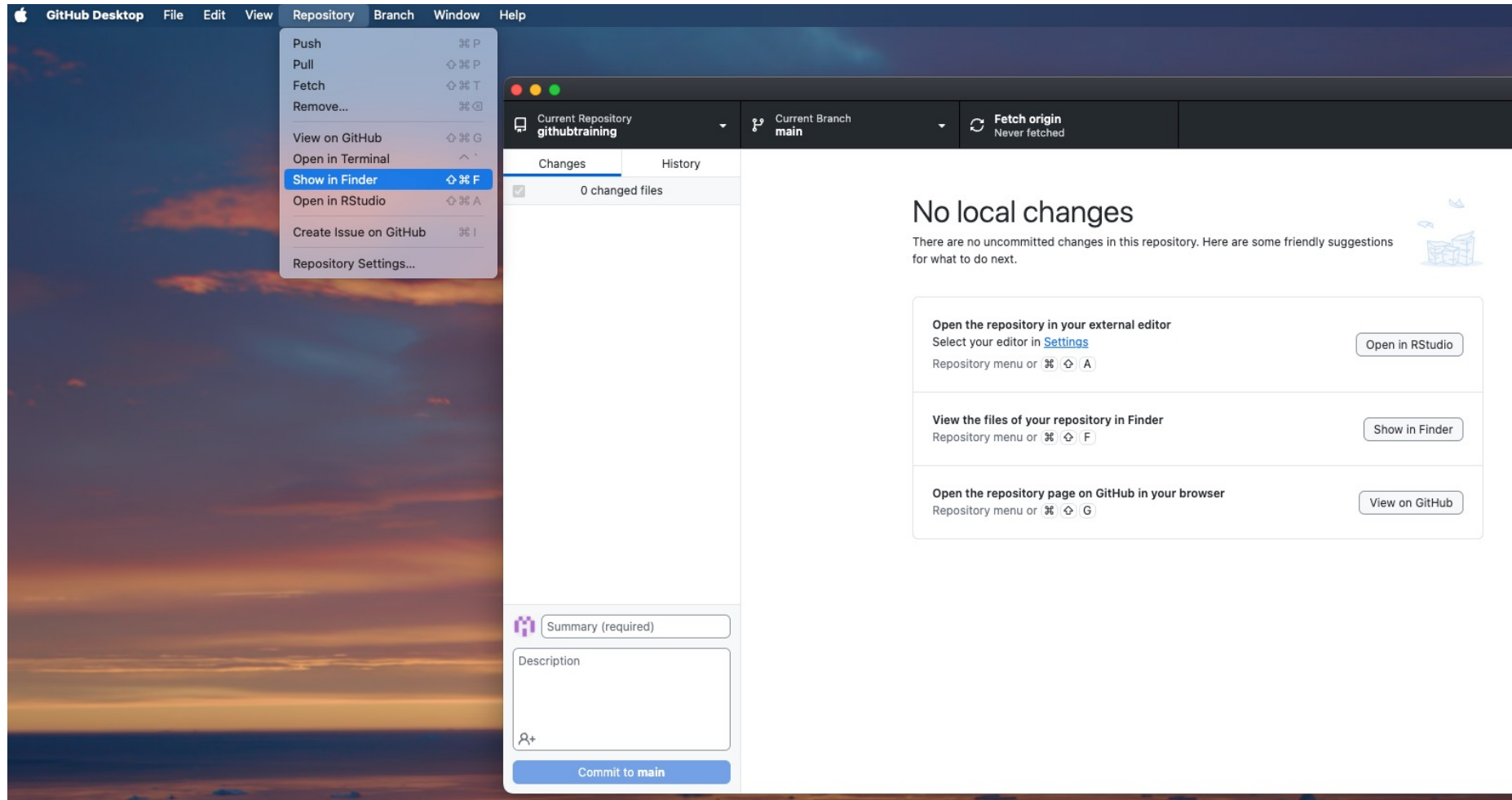
**git push**

# Making Local Changes and Pushing to Remote

Make sure you have cloned the remote repo to your laptop



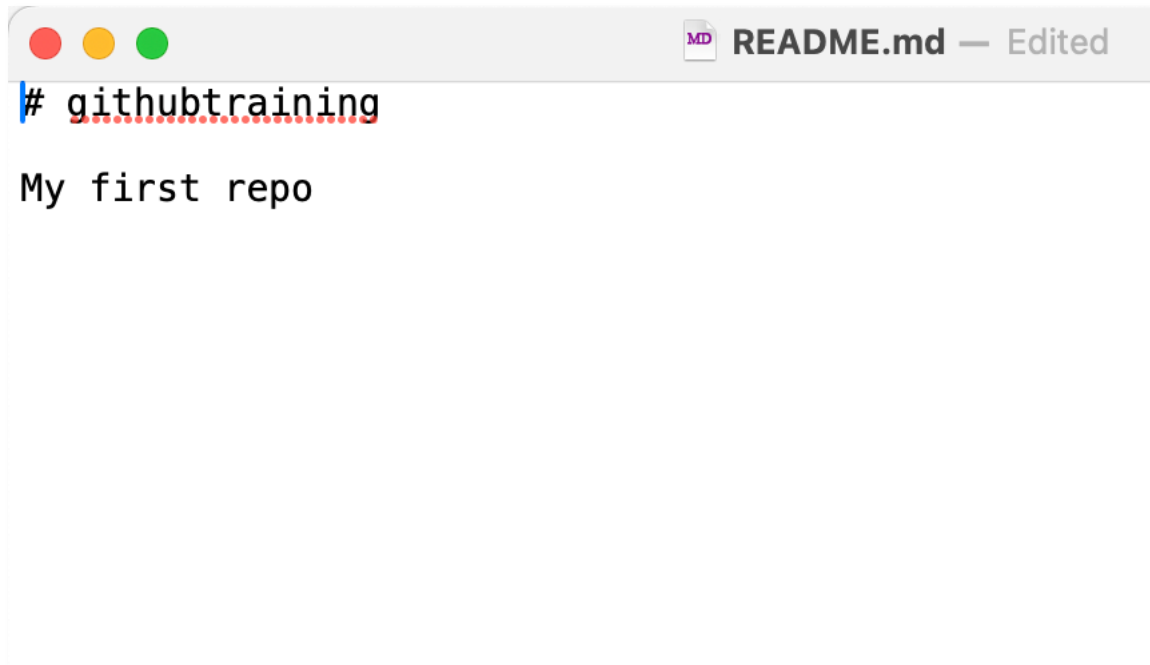
# Opening the Local Folder for the Repository



# Make changes to README.md file

Use any editor, write some text to this file.

Ex:

A screenshot of a text editor window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, and a document icon followed by the text 'README.md — Edited' on the right. The editor area contains two lines of text: the first line is '# githubtraining' with a blue cursor at the end of the line; the second line is 'My first repo'.

```
# githubtraining
My first repo
```

# Git commit

Current Repository: githubtraining | Current Branch: main | Fetch origin (Last fetched 1 minute ago)

Changes: 1 | History

1 changed file

README.md

@@ -1 +1,3 @@

- # githubtraining

+ # githubtraining

+ # githubtraining

+ My first repo

The changes we just made

add my first repo to readme.md

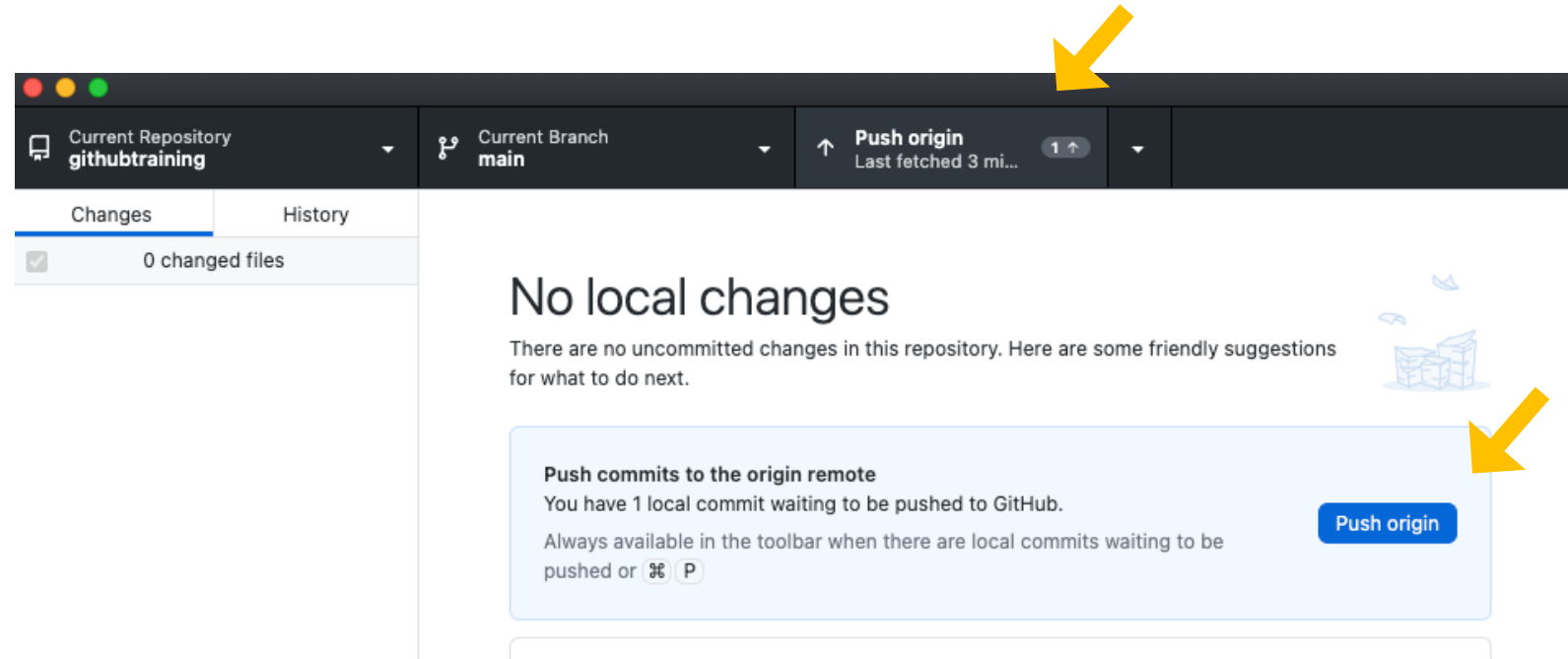
Description

Commit to main

Write a clear commit message

Click here to git commit

# Git push

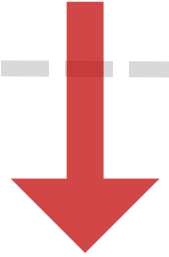


Check your remote repo after git push

# Git Commit

- A core function in the Git version control system that saves a snapshot of the project's staged changes, creating a "commit" object in the repository history.
- Each commit includes:
  - Snapshot of Changes
  - Unique Identifier
  - Author Information
  - Timestamp
  - Commit Message

# remote



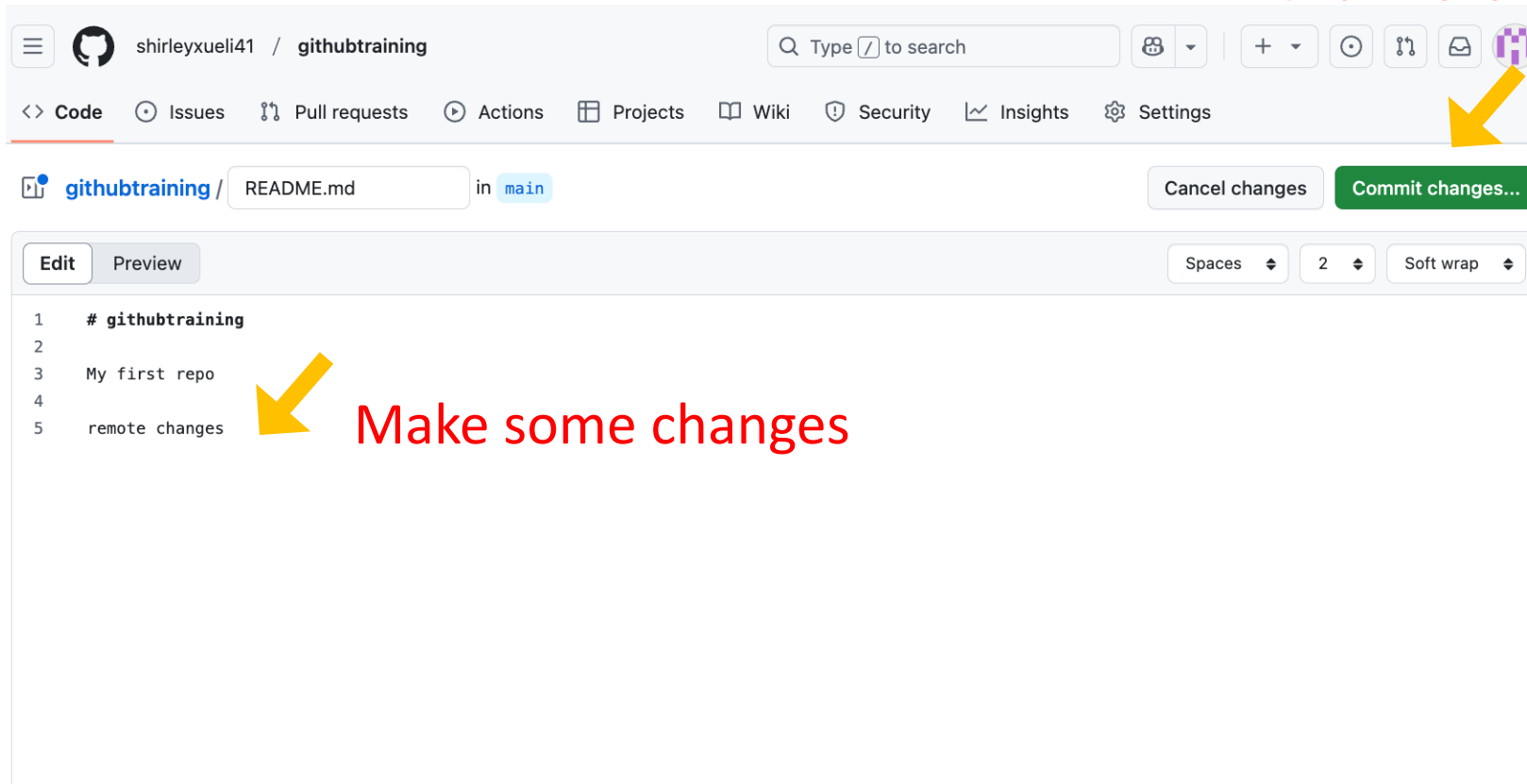
# Local (collaborator)



**git fetch & pull**

# Make changes to README.md file remotely

Click here to git commit



# Write clear commit message

Commit changes

×

Commit message

Write a clear commit message

add remote changes note

Extended description

Add an optional extended description..

- ☒ Commit directly to the main branch
- ☐ Create a **new branch** for this commit and start a pull request  
[Learn more about pull requests](#)

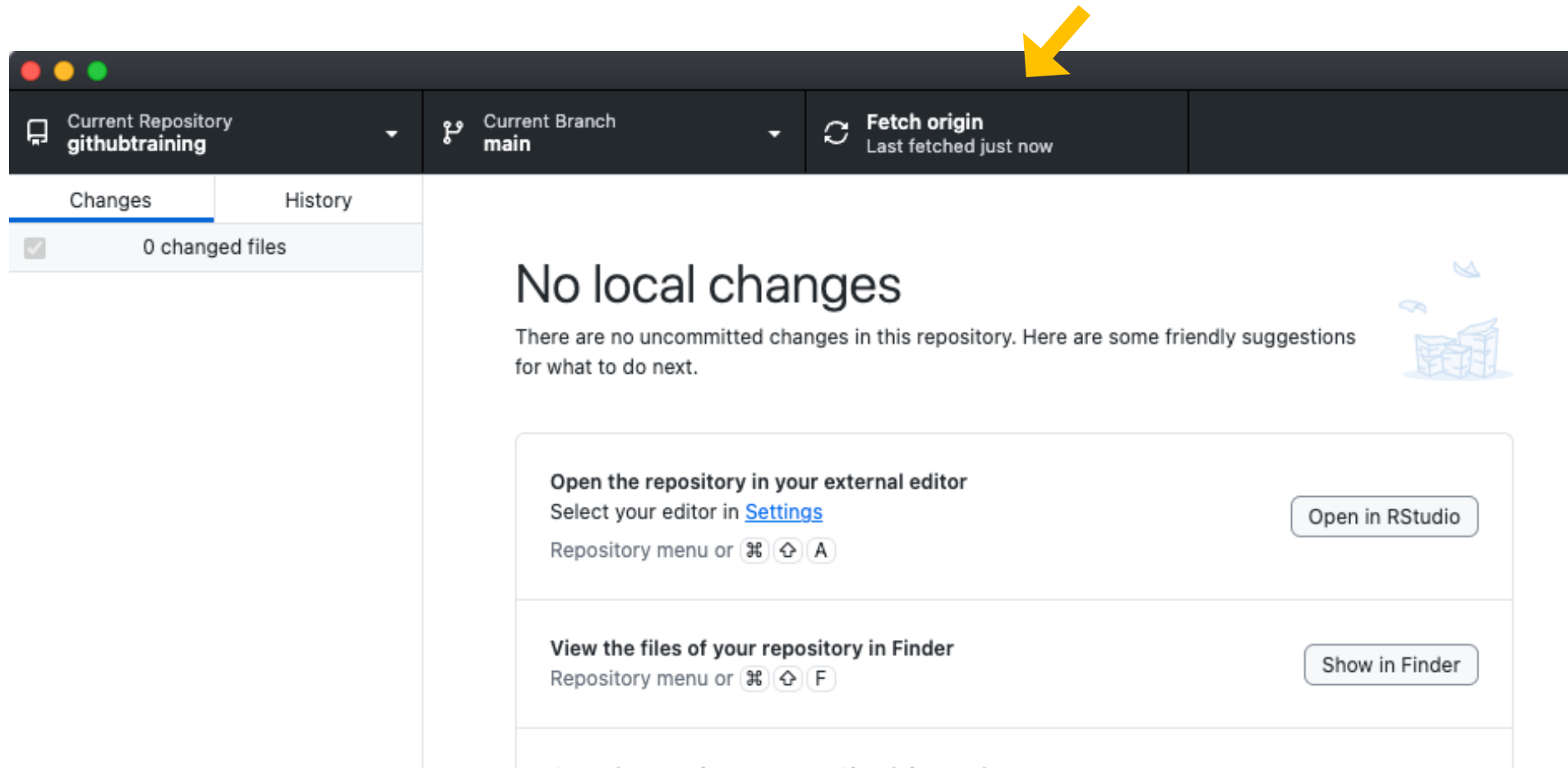
Click here to git commit



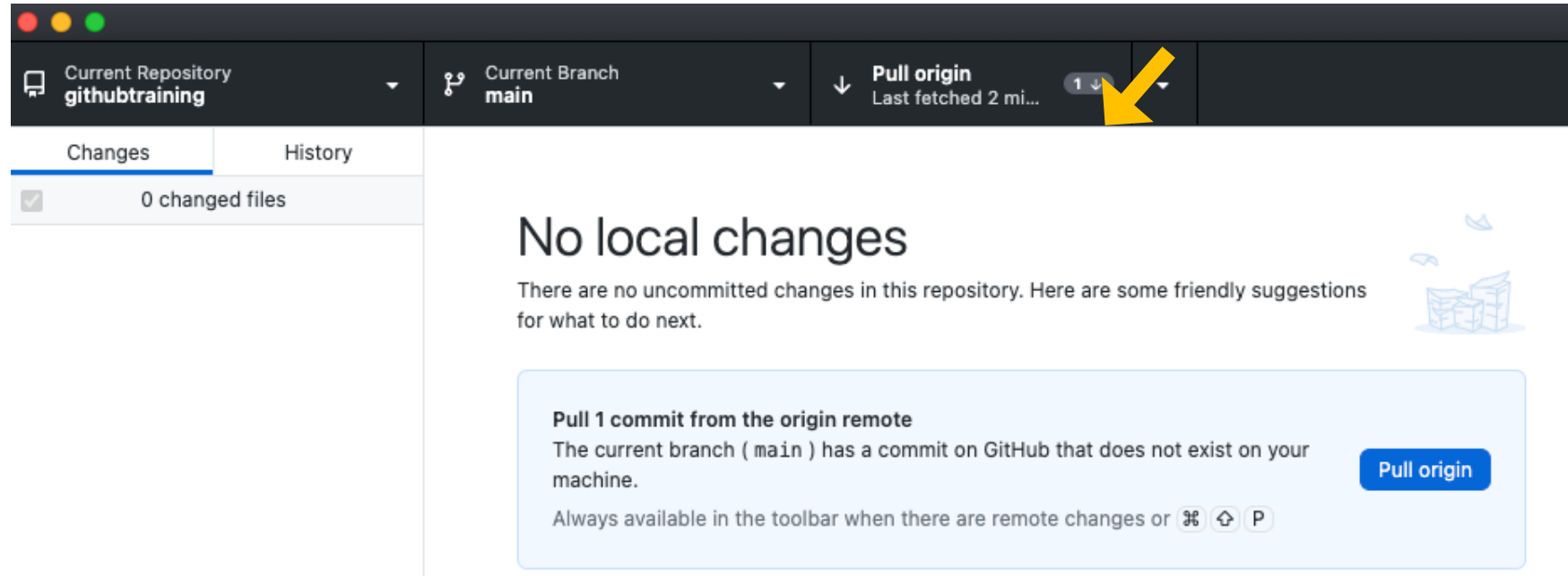
Cancel

Commit changes

# Git Fetch



# Git Pull



After fetch and pull, check README.md file locally to view the changes

# Revert to old version

**Commit 1:**  
Create file.txt

**Commit 2:**  
Delete file.txt

- How to recover file.txt?

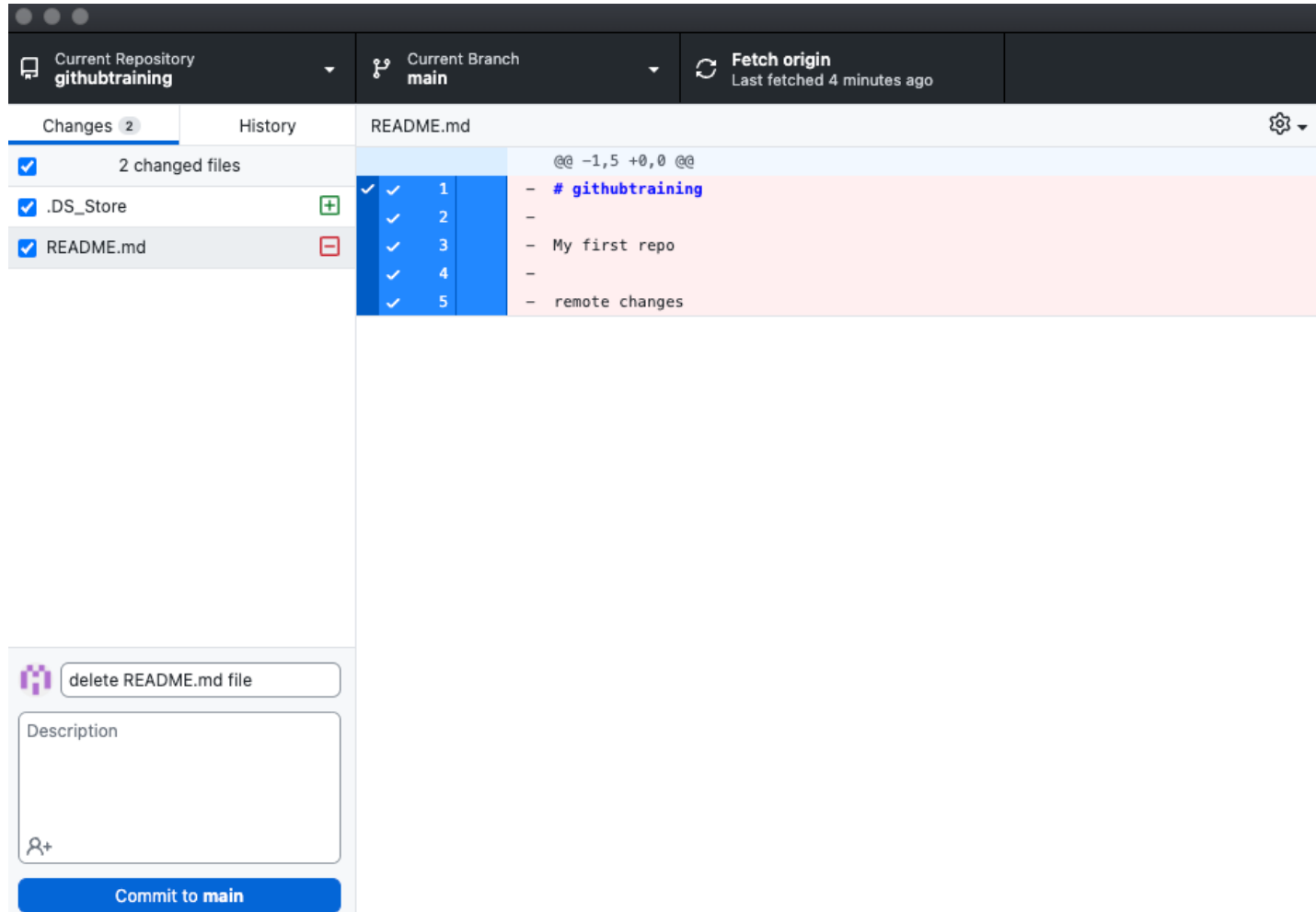
**Commit 1:**  
Create file.txt

**Commit 2:**  
Delete file.txt

**Commit 3:**  
Revert "Delete file.txt"

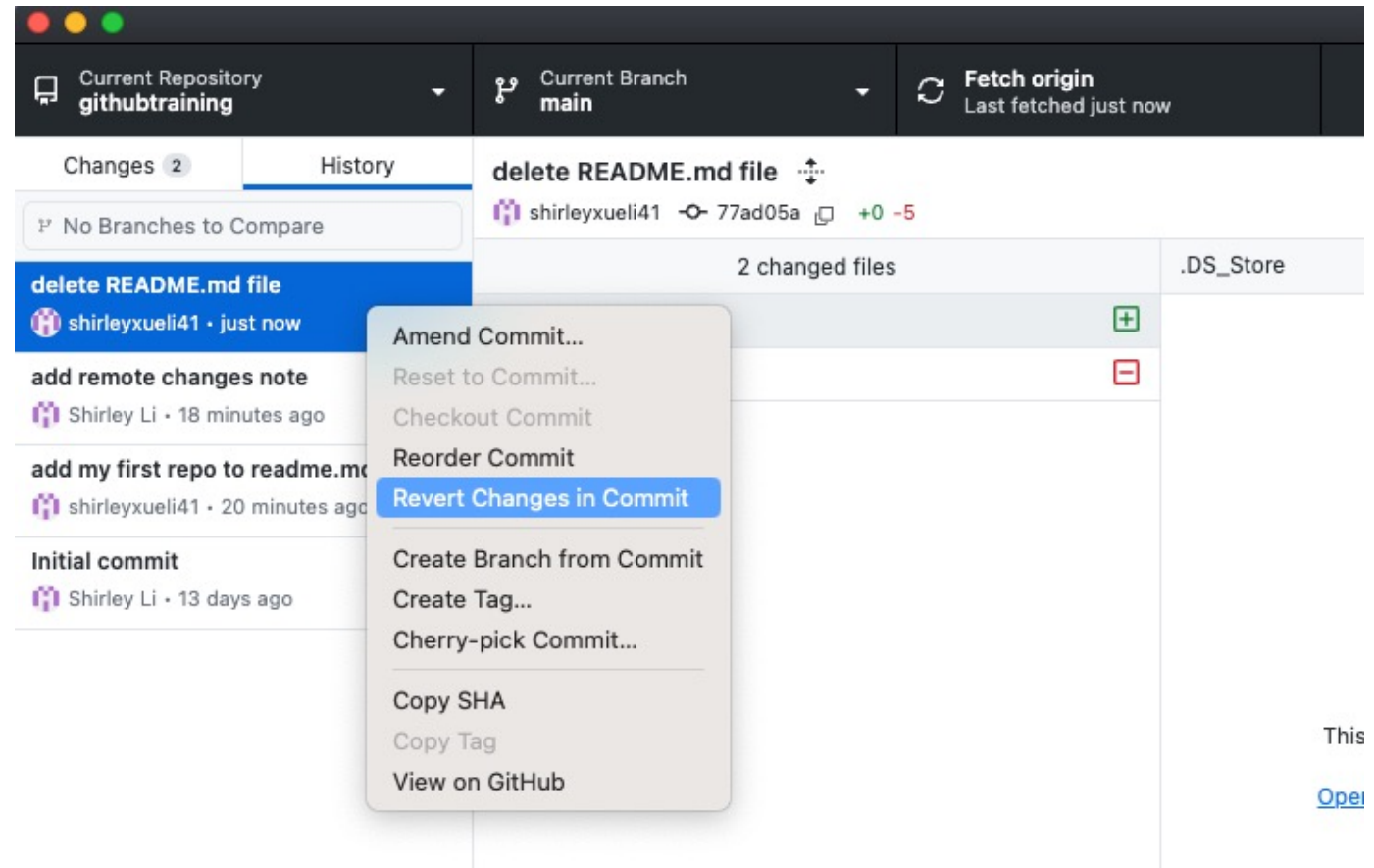
**A new commit**

# Delete README.md file locally and then revert



# Revert changes in commit

This will create a new commit that reverses your changes while preserving the history



Current Repository **githubtraining**

Current Branch **main**

Push origin  
Last fetched just ... 1 ↑

Changes | **History**

No Branches to Compare

**Revert "delete README.md file"**

This reverts commit 77ad05ae33134b8efae8f31706b3032766347c5d.

shirleyxueli41 · bb03471 · +5 -0

2 changed files

.DS_Store	[-]
README.md	[+]

.DS\_Store

This binary file has changed.

[Open file in external program.](#)

# Git Fetch vs. Pull

Command	What It Does	Use Case
Git fetch	Downloads changes but doesn't merge.	Check remote updates before merging.
Git pull	Fetches and merges changes automatically.	Quickly update your branch.

# Git vs. GitHub Desktop

- **Git: a command line tool**
  - Git provides more detailed control over all aspects of version control, suitable for complex development workflows.
- **GitHub Desktop: a graphical user interface (GUI)**
  - GitHub Desktop focuses on simplifying common Git operations, which may limit some advanced functions.

# .gitignore

- A file that tells Git which files or directories to ignore.
- Helps keep unnecessary or sensitive files out of version control.
- **Avoid committing large files**
- **Protect sensitive information**

# Common .gitignore examples

```
# Ignore all log files
*.log

# Ignore environment files
.env

# Ignore data files
data/
```

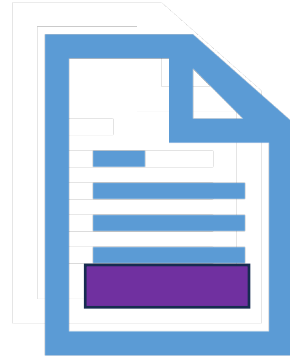
Place .gitignore in your repository's root directory

# How conflicts are generated and how should we resolve them?

# Conflicts

- In services like Box or Dropbox, conflicts arise when two team members simultaneously make changes to the same file.
- Similarly, in a GitHub repository, conflicts occur when two team members modify **the same part of a file concurrently**.

# remote



Your collaborator make changes remotely  
or they make changes on their local copy  
and commit and push back to GitHub.

# local

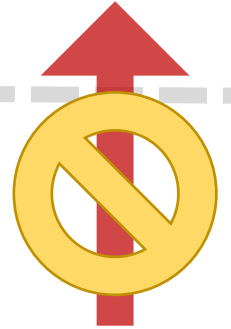


You make changes locally

# remote



# local



**Commit and push will  
cause conflicts**

# remote



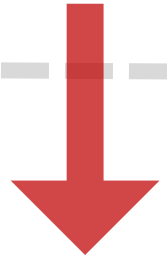
---

When two team members modify  
**different part of a file concurrently**, no  
conflicts will occur.

# local



# remote



# local



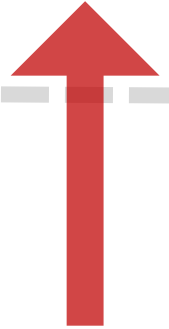
You will first pull to update your local copy with the latest changes

# remote



then push to apply your changes to the remote copy.

# local



# Best practices for GitHub Usage

# Best practices

- Sync frequently to make your changes available to others. Frequently: Fetch& pull before making any local changes.
- Small, Frequent Commits.
- Push Regularly: Push your commits
- Branch Strategically: Use branches to manage features, bug fixes, and experiments separately from the main codebase.
- Communicate Regularly.
- Document Changes: Update README.
- Do not store large files in GitHub. It has limited storage.

# Good commit

- Single Focus: Each commit should represent a single logical change.
- Small Size: Smaller commits are easier to understand and less likely to introduce complex merge conflicts.
- Always write good commit messages.

# Good commit messages

- **Concise, specific.**
  - Good: “Added comments explaining job script parameters”
  - Bad: “Update of file.txt”, “Fixed it”
- **Detailed explanation including what, why, and how.**
- **Not too long, not too short. ~50 characters.**
- **References to related issue or pull requests: “See also #46”**

# Example repositories

<https://github.com/orgs/fanzhangelab/repositories>

<https://github.com/tuftsdatalab/tuftsWorkshops/tree/main>

# A summary of what we learned

- Learned the basics of Git and how it works
- Created a GitHub repository
- Used GitHub Desktop to manage repositories
- Explored common scenarios like ignoring files and resolving conflicts
- Discussed best practices for commits and repository organization

# More about best practices

- <https://docs.github.ncsu.edu/github-best-practices/>
- <https://docs.github.com/en/repositories/creating-and-managing-repositories/best-practices-for-repositories>
- <https://github.com/orgs/community/discussions/39082>
- <https://dangitgit.com/>

# Tufts University GitHub Enterprise

## Access:

Visit [access.tufts.edu/github](https://access.tufts.edu/github) for setup instructions and account access.

## About the Enterprise License:

Tufts University maintains a **GitHub Enterprise** license that provides:

- Centralized administration and policy management
- Advanced security, compliance, and audit features
- Single sign-on (SSO), internal repositories, and enhanced support

## Account Management:

Your account is created under the **tufts.edu** domain.

If you leave Tufts, you will need to transition to a **personal GitHub account**.

## Contact:

For questions or assistance, email [it@tufts.edu](mailto:it@tufts.edu)

# Recommended tools



<https://code.visualstudio.com/>

# VSCode server on Tufts OOD

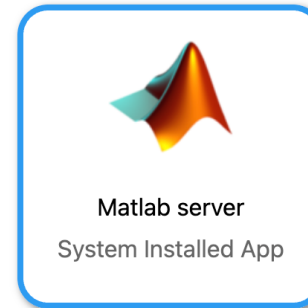
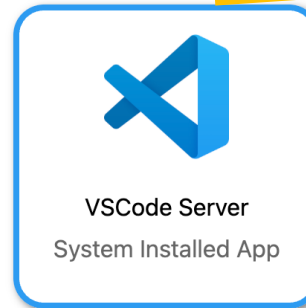


## High Performance Computing

Research Technology, TTS

Welcome to the Open OnDemand web portal to Tufts HPC cluster. It provides convenient access to many HPC applications.

**Pinned Apps** A featured subset of all available apps



# Useful Materials

- [Git and GitHub for Beginners – Crash Course from freeCodeCamp.org](#)
- [Git cheatsheet](#)