

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

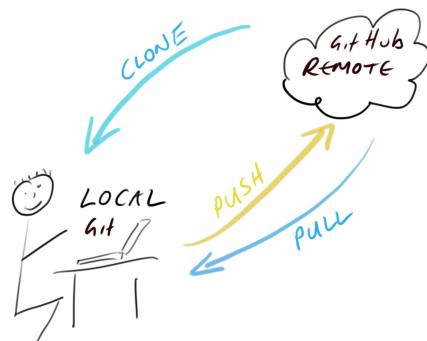
What is Git and GitHub?

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.



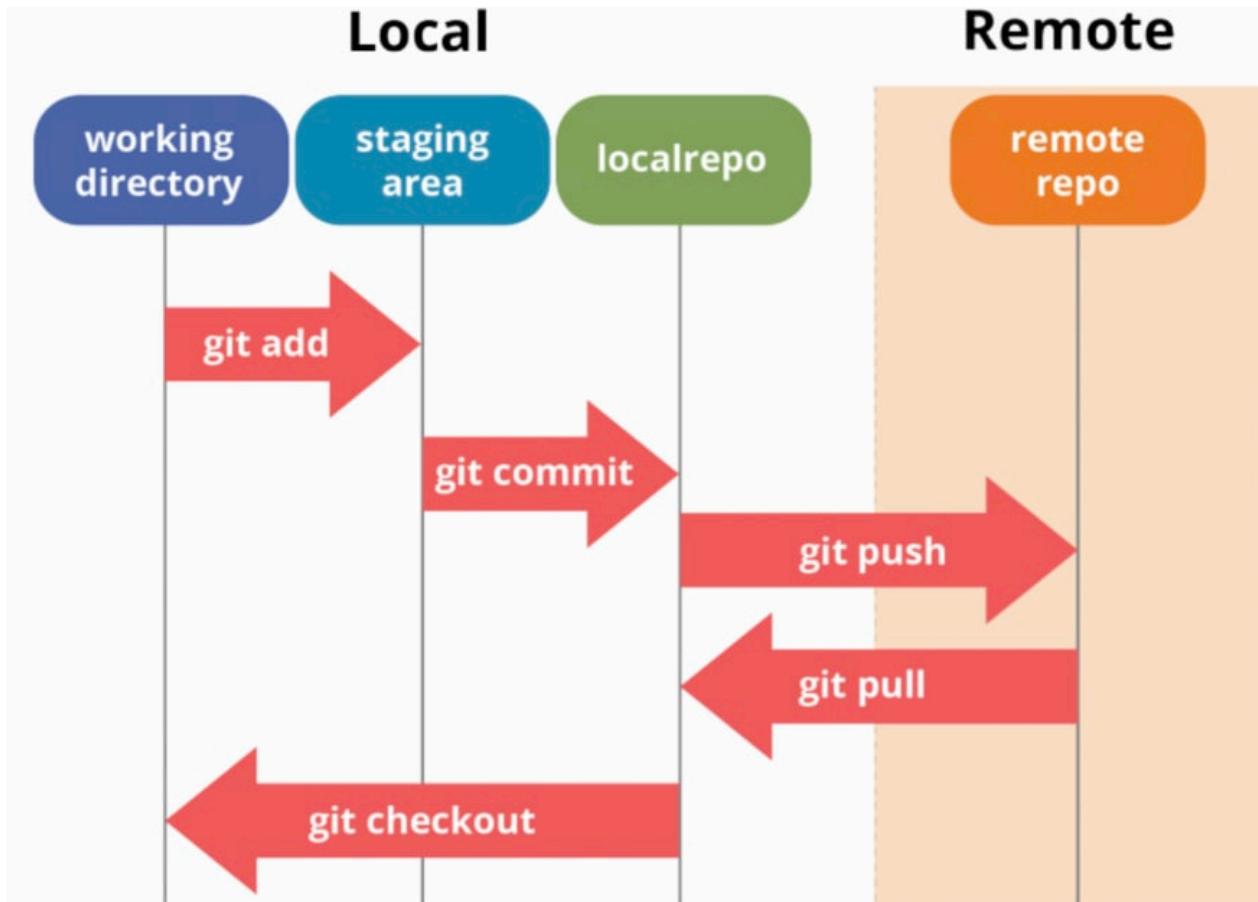
Basic Git Operations

- 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.

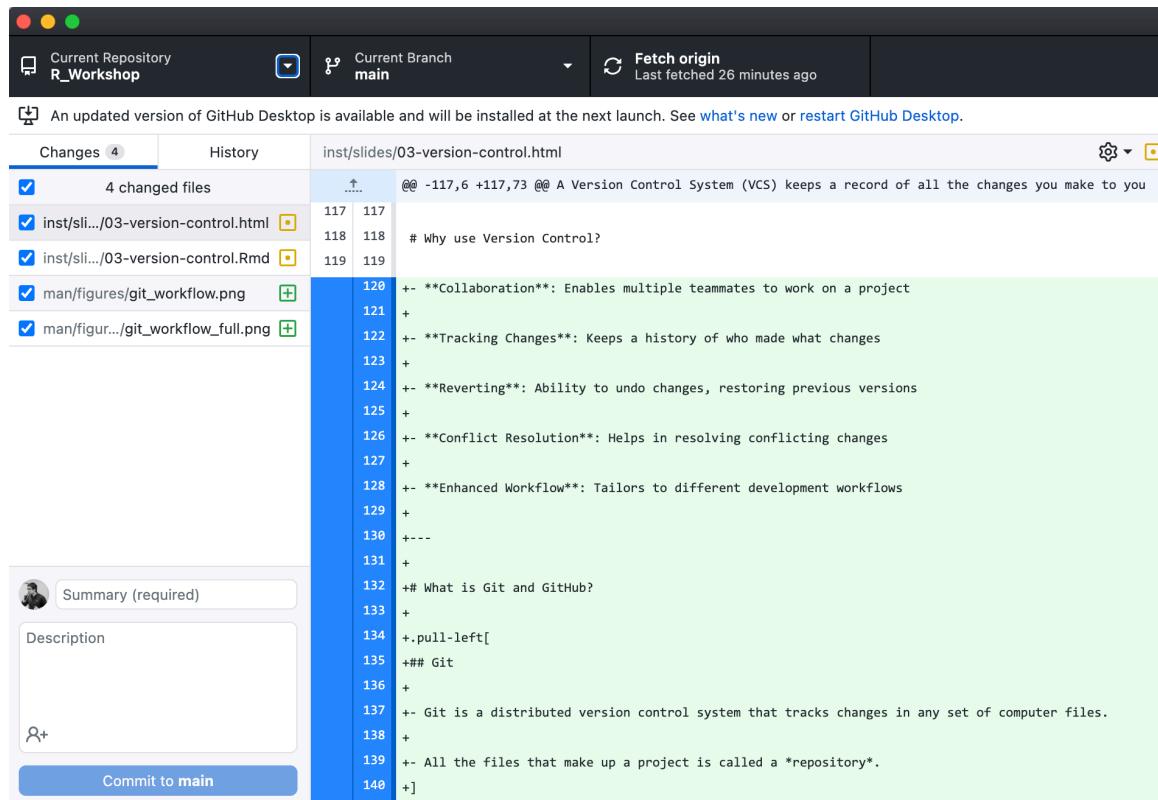
Git Workflow



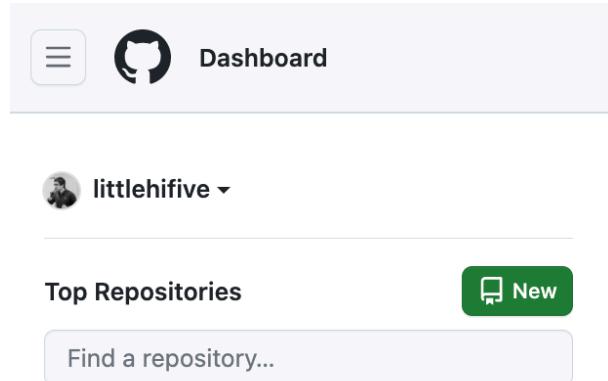
Source: <https://dev.to/mollynem/git-github--workflow-fundamentals-5496>

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



The screenshot shows the GitHub dashboard. At the top, there's a navigation bar with a menu icon, the GitHub logo, and the word "Dashboard". Below the navigation, the user profile "littlehifive" is displayed with a dropdown arrow. A section titled "Top Repositories" shows a green "New" button. Below this, there's a search bar with the placeholder "Find a 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.

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 (*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *



littlehifive

Repository name *



test_git_repo

test_git_repo is available.

Great repository names are short and memorable. Need inspiration? How about [fantastic-succotash](#) ?

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.

Initialize this repository with:

Add a README file

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

Add .gitignore

.gitignore template:None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

License:None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

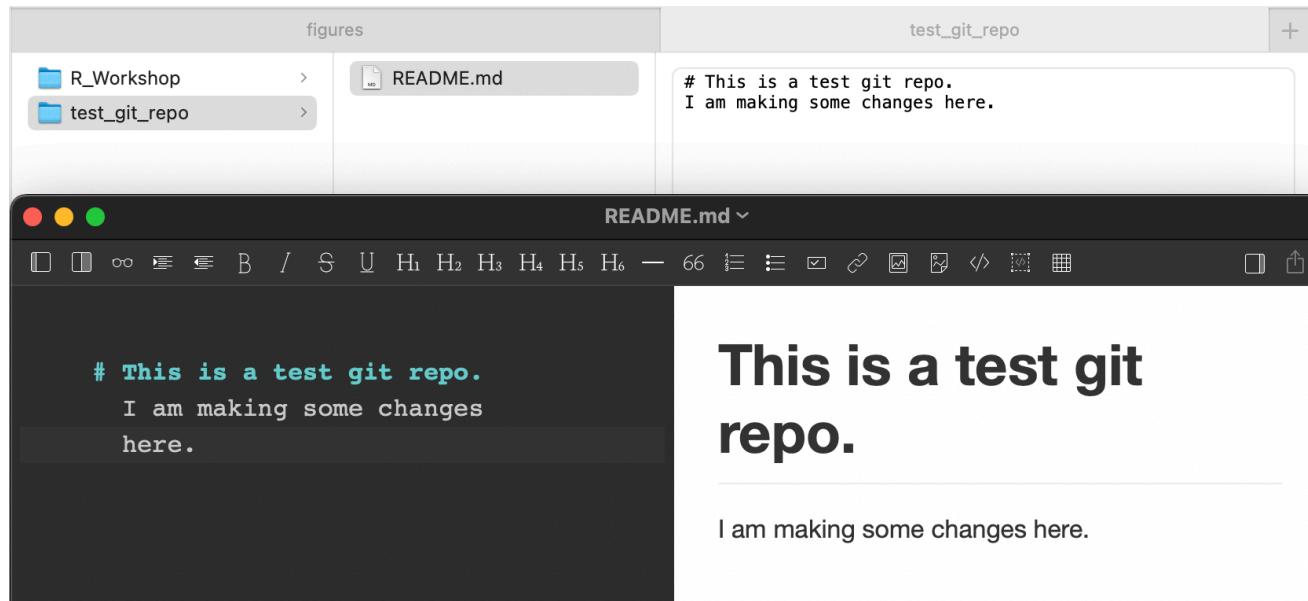
2. Clone the repository

A screenshot of a GitHub repository page for 'test_git_repo'. The page shows basic repository statistics: 1 branch, 0 tags, and a README.md file. A modal window titled 'Clone' is open, displaying cloning options via HTTPS, SSH, or GitHub CLI. The URL https://github.com/littlehifive/test_git_repo is shown. Below the URL are links for 'Open with GitHub Desktop' and 'Download ZIP'. To the right of the modal, there's an 'About' section with a note about no description, website, or topics provided, and sections for 'Readme', 'Activity', 'Stars', 'Watching', 'Forks', 'Releases', and 'Packages'.

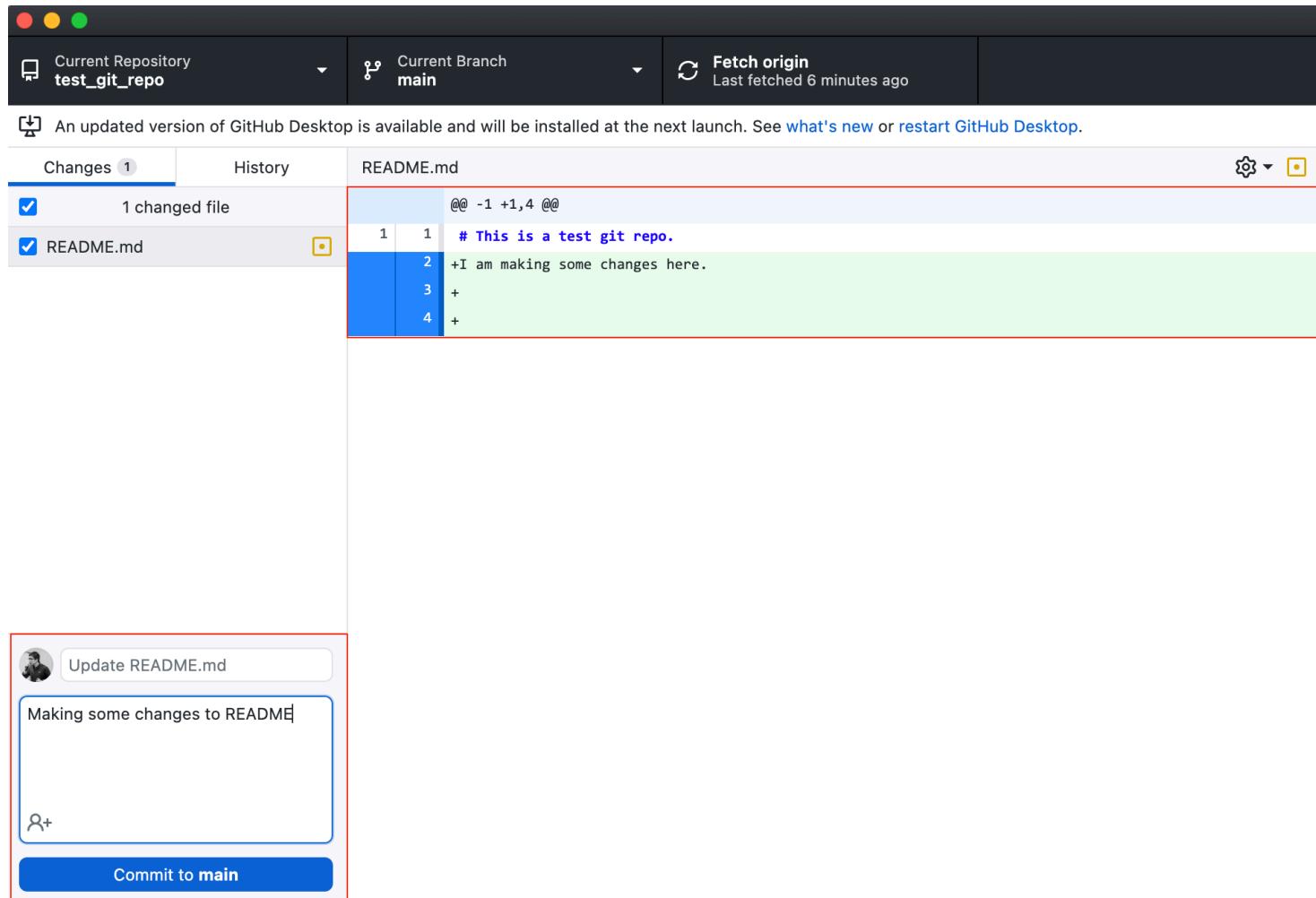
A screenshot of a 'Clone a Repository' dialog box. It has tabs for 'GitHub.com', 'GitHub Enterprise', and 'URL', with 'URL' selected. A text input field contains the repository URL https://github.com/littlehifive/test_git_repo. Below the URL input is a 'Local Path' field containing the path '/Users/michaelfive/Code/test_git_repo'. At the bottom of the dialog are 'Cancel' and 'Clone' buttons.

3. Make changes to a file

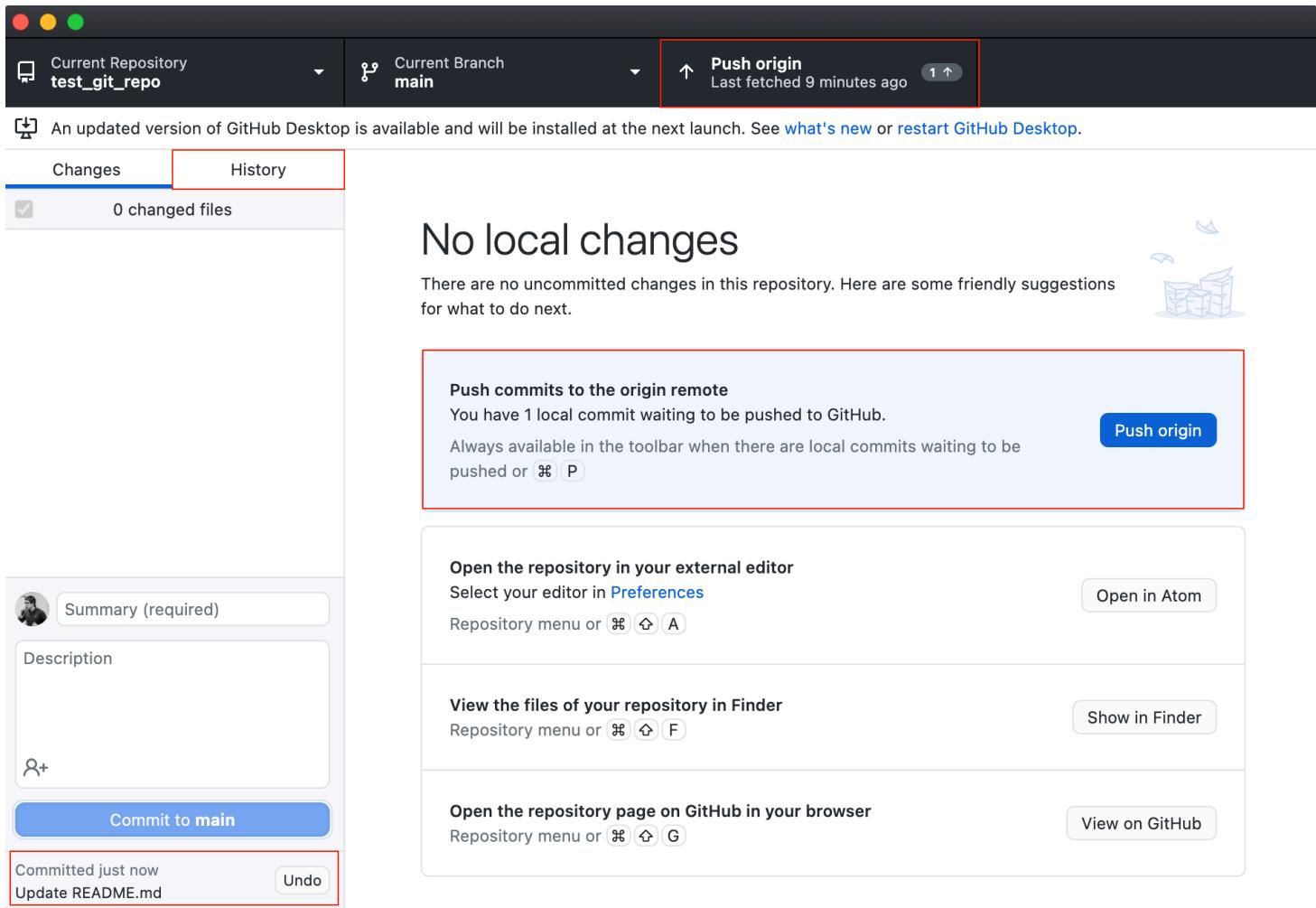
- Open README.md using any text editor on your computer
- Add a sentence and save the document



4. Commit changes to main



5. Push changes to origin



An updated version of GitHub Desktop is available and will be installed at the next launch. See [what's new](#) or [restart GitHub Desktop](#).

Changes History

Push origin
Last fetched 9 minutes ago 1 ↑

No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Push commits to the origin remote

You have 1 local commit waiting to be pushed to GitHub.

Always available in the toolbar when there are local commits waiting to be pushed or ⌘ P

Push origin

Open the repository in your external editor

Select your editor in [Preferences](#)

Repository menu or ⌘ ⌘ A

Open in Atom

View the files of your repository in Finder

Repository menu or ⌘ ⌘ F

Show in Finder

Open the repository page on GitHub in your browser

Repository menu or ⌘ ⌘ G

View on GitHub

Summary (required)

Description

Commit to main

Committed just now

Update README.md

Undo

6. See changes on GitHub

The screenshot shows a GitHub repository named "test_git_repo" which is public. The repository has one branch ("main") and no tags. A recent commit by user "littlehifive" titled "Update README.md" was made 3 minutes ago. The commit hash is ed696ec and it contains 2 commits. The README.md file shows the text "This is a test git repo." with a note below it stating "I am making some changes here." There is also an edit icon next to the file name.

test_git_repo Public

Pin Unwatch 1

main 1 branch 0 tags Go to file Add file Code

littlehifive Update README.md ... ed696ec 3 minutes ago 2 commits

README.md Update README.md 3 minutes ago

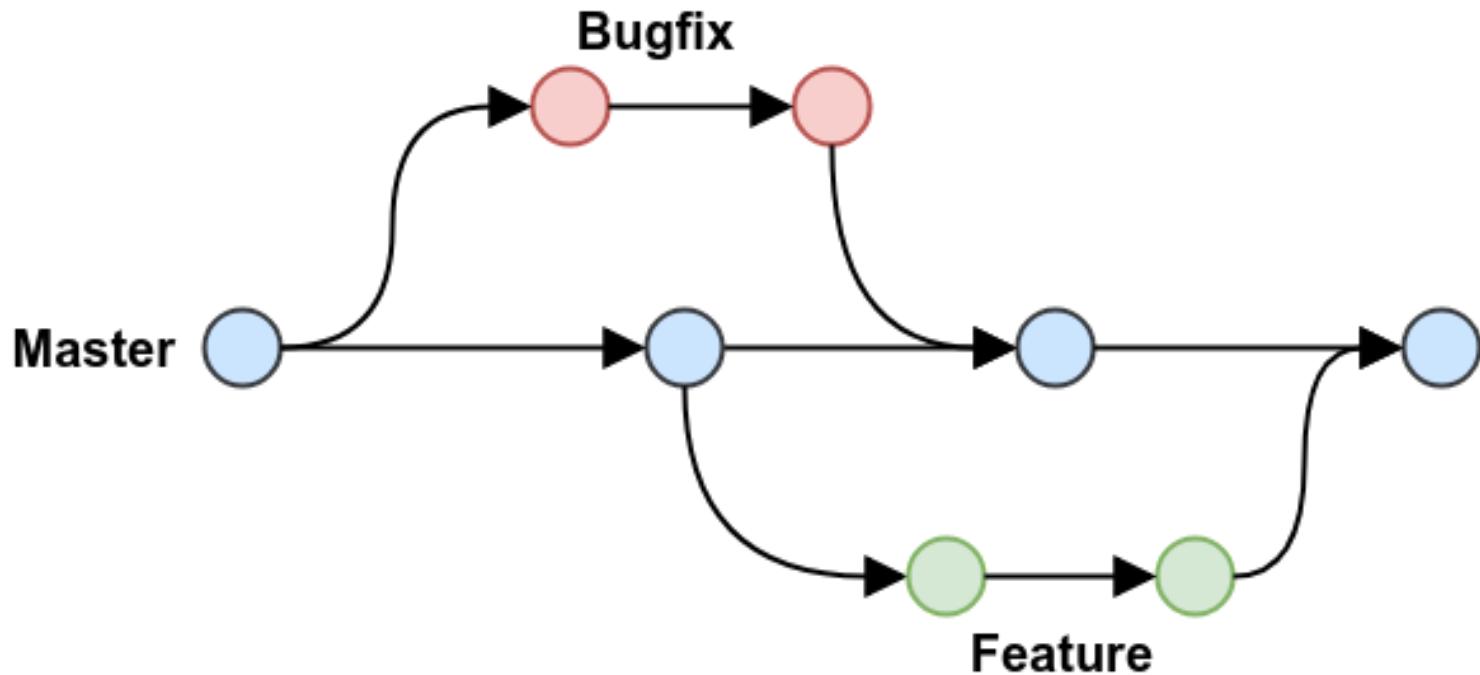
README.md

This is a test git repo.

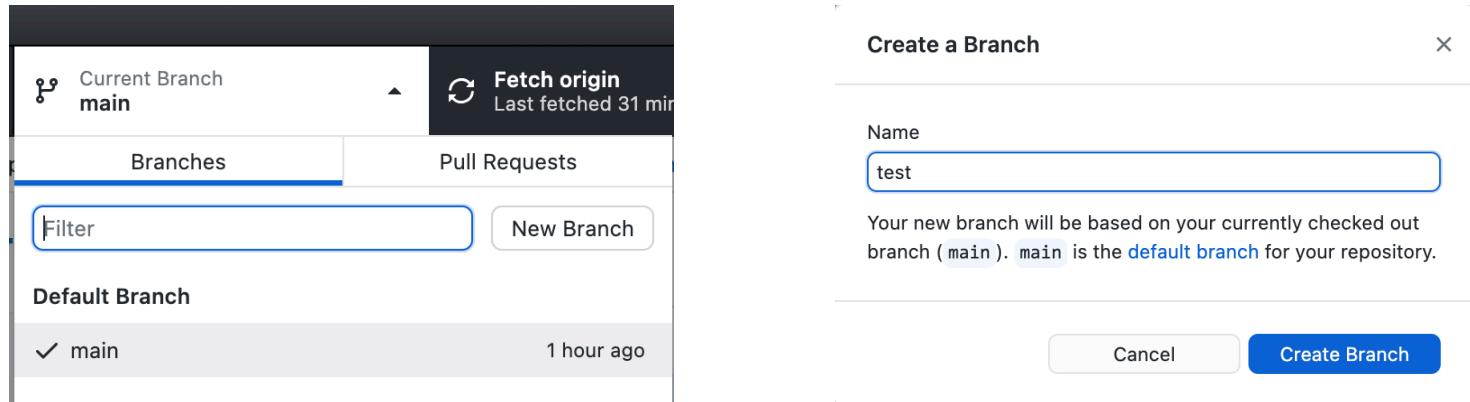
I am making some changes here.

Branching

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



7. Create a branch



The screenshot shows a file browser with a sidebar containing 'R_Workshop' and 'test_git_repo'. The 'test_git_repo' folder is selected, revealing a 'README.md' file. The file content is:

```
# This is a test git repo.  
I am making some changes here.  
  
I am making some changes to this branch.
```

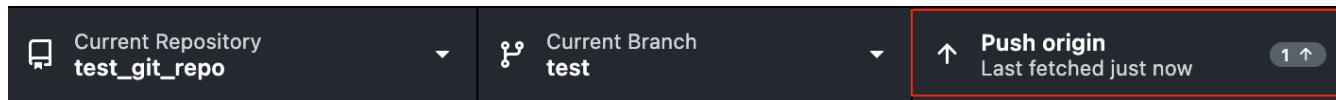
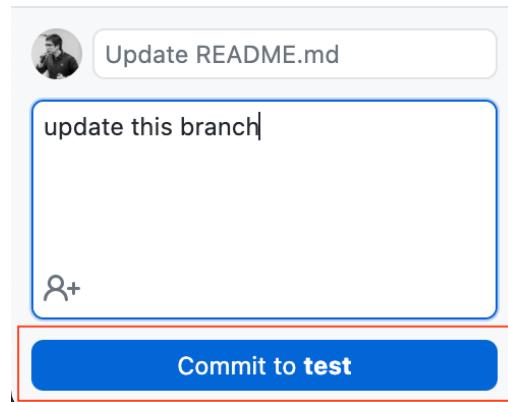
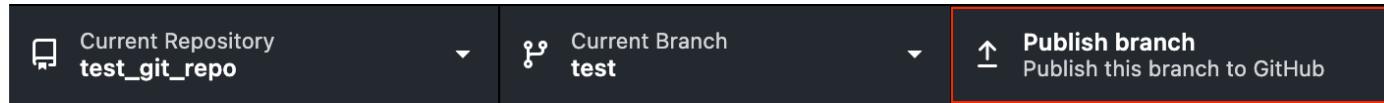
Below the file browser is a code editor window titled 'README.md' with the same content displayed.

This is a test git repo.

I am making some changes here.

I am making some changes to this branch.

8. Publish the branch



9. Pull request

The screenshot shows the GitHub interface for creating a pull request. At the top, there's a header with the repository name 'test_git_repo' (Public), a 'Pin' button, and an 'Unwatch' button with a count of 1. Below the header, a yellow banner displays a message: 'test had recent pushes less than a minute ago' and a green 'Compare & pull request' button.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main ▾ compare: test ▾ ✓ Able to merge. These branches can be automatically merged.

Update README.md

Write Preview

update this branch

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Development
Use [Closing keywords](#) in the description to automatically close issues

Helpful resources
[GitHub Community Guidelines](#)

-o 1 commit 1 file changed 1 contributor

10. Merge pull request

Update README.md #1

[Open](#) littlehifive wants to merge 1 commit into `main` from `test`

Conversation 0 Commits 1 Checks 0 Files changed 1

littlehifive commented 2 minutes ago

update this branch

-o Update README.md ... 6e13082

Add more commits by pushing to the `test` branch on [littlehifive/test_git_repo](#).

Require approval from specific reviewers before merging
[Branch protection rules](#) ensure specific people approve pull requests before they're merged.

Continuous integration has not been set up
[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

[Merge pull request](#) You can also open this in [GitHub Desktop](#) or view [command line instructions](#).

11. See changes on GitHub

The screenshot shows a GitHub repository page for a public repository named "test_git_repo". The main navigation bar includes "Pin" and "Unwatch 1". Below the repository name, it shows "main", "1 branch", "0 tags", "Go to file", "Add file", and "Code". A red box highlights a merge pull request from "littlehifive/test" to "main". The commit details show "60497eb now" and "4 commits". A file change log shows "README.md" updated 14 minutes ago. The README content includes the text "This is a test git repo." and "I am making some changes here.", with the latter in a red box. A red box also highlights the commit message "I am making some changes to this branch.".

What if there are conflicts?

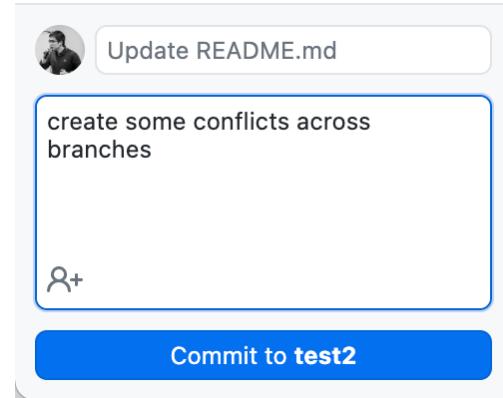
Create a Branch

Name
test2

Create branch based on...

main
The default branch in your repository. Pick this to start on something new that's not dependent on your current branch.

test
The currently checked out branch. Pick this if you need to build on work done on this branch.



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).



base: test



compare: test2

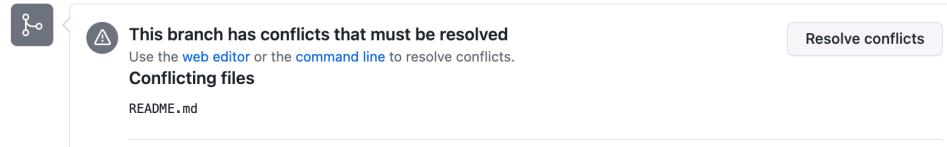
...

X Can't automatically merge.

Don't worry, you can still create the pull request.

Resolving conflicts

Add more commits by pushing to the `test2` branch on [littlehifive/test_git_repo](#).



Test2 #2

Resolving conflicts between `test2` and `test` and committing changes → `test2`

The screenshot shows a GitHub code editor interface. On the left, there's a sidebar with "1 conflicting file" and a list containing "README.md". The main area shows the content of "README.md". Lines 2 through 6 are highlighted in yellow, indicating a conflict. The code is as follows:

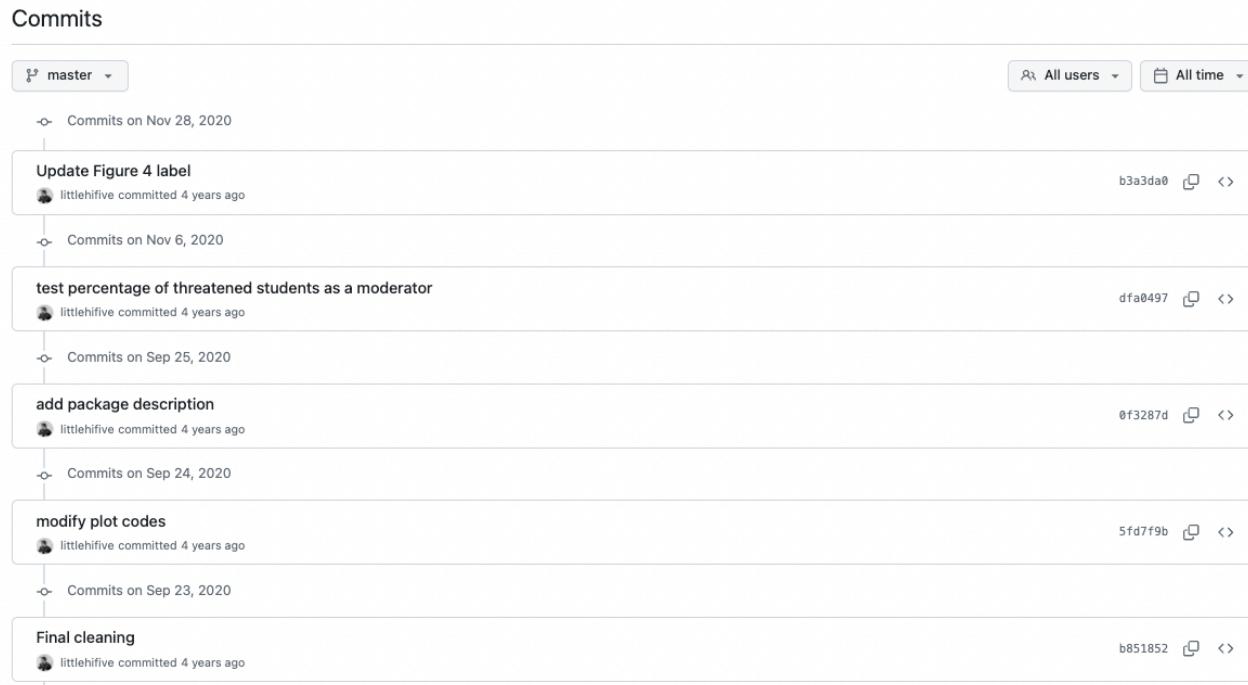
```
1 # This is a test git repo.
2 <<<<< test2
3 I am making some changes here. (I am not changing this line again).
4 =====
5 I am making some changes here. (I am changing this line again).
6 >>>>> test
7
8 I am making some changes to this branch.
```

At the top right, there are buttons for "1 conflict", "Prev ▲", "Next ▼", "Mark as resolved", and a gear icon.

- 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`).

A screenshot of a GitHub commit page for a file named "Finalize analyses and outputs". The commit was made by user "littlehifive" 4 years ago. The commit message is "Finalize analyses and outputs". The blame tab is selected, showing the history of changes across 16 lines of R code. The code implements a multilevel analysis function, starting with "# Main analysis of average treatment effect -----". The blame history shows contributions from multiple users over four years, with color-coded blame blocks indicating the author of each line.

```
# Main analysis of average treatment effect -----
# Run multilevel analysis
run_meta_mlm <- function(data, random){
  model <- metafor::rma.mv(es,
                           v,
                           random = random, # e.g. "cluster"
                           tdist = TRUE,
                           data = data,
                           method = "REML")
  return(model)
}
```

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

The screenshot shows the GitHub Desktop application interface. At the top, there are three dropdown menus: 'Current Repository' set to 'test_git_repo', 'Current Branch' set to 'test2', and 'Fetch origin' with a note 'Last fetched just now'. Below these, a message indicates an updated version of GitHub Desktop is available.

The main area displays a list of commits on the left and a detailed commit view on the right. The commit list includes:

- Merge branch 'test2' of https://github.com/test-git-repo into test2 by Michael Wu · just now
- Revert "Update README.md" by Michael Wu · just now
- Merge branch 'test' into test2 by Zezhen Wu · 8 minutes ago
- Update README.md by Michael Wu · 18 minutes ago
- Update README.md** by Michael Wu · just now (highlighted)
- Merge pull request by Zezhen Wu · 2 hours ago
- Update README.md by Michael Wu · 2 hours ago
- Create README.md by Zezhen Wu · 2 hours ago

The detailed view for the highlighted commit 'Update README.md' shows the file 'README.md' with the following diff:

```
@@ -1,5 +1,5 @@
 1 1 # This is a test git repo.
 2 -I am making some changes here.
 2 +I am making some changes here. (I am changing
 3   this line again).
 3 3
 4 4 I am making some changes to this branch.
 5 5
```

A context menu is open over the commit in the list, showing options: 'Revert Changes in Commit', 'Create Branch from Commit', 'Create Tag...', 'Cherry-pick Commit...', 'Copy SHA', and 'View on GitHub'.

Bonus: Fork a project

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk ().*

Owner *

Choose an owner ▾

Repository name *

R_Workshop

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

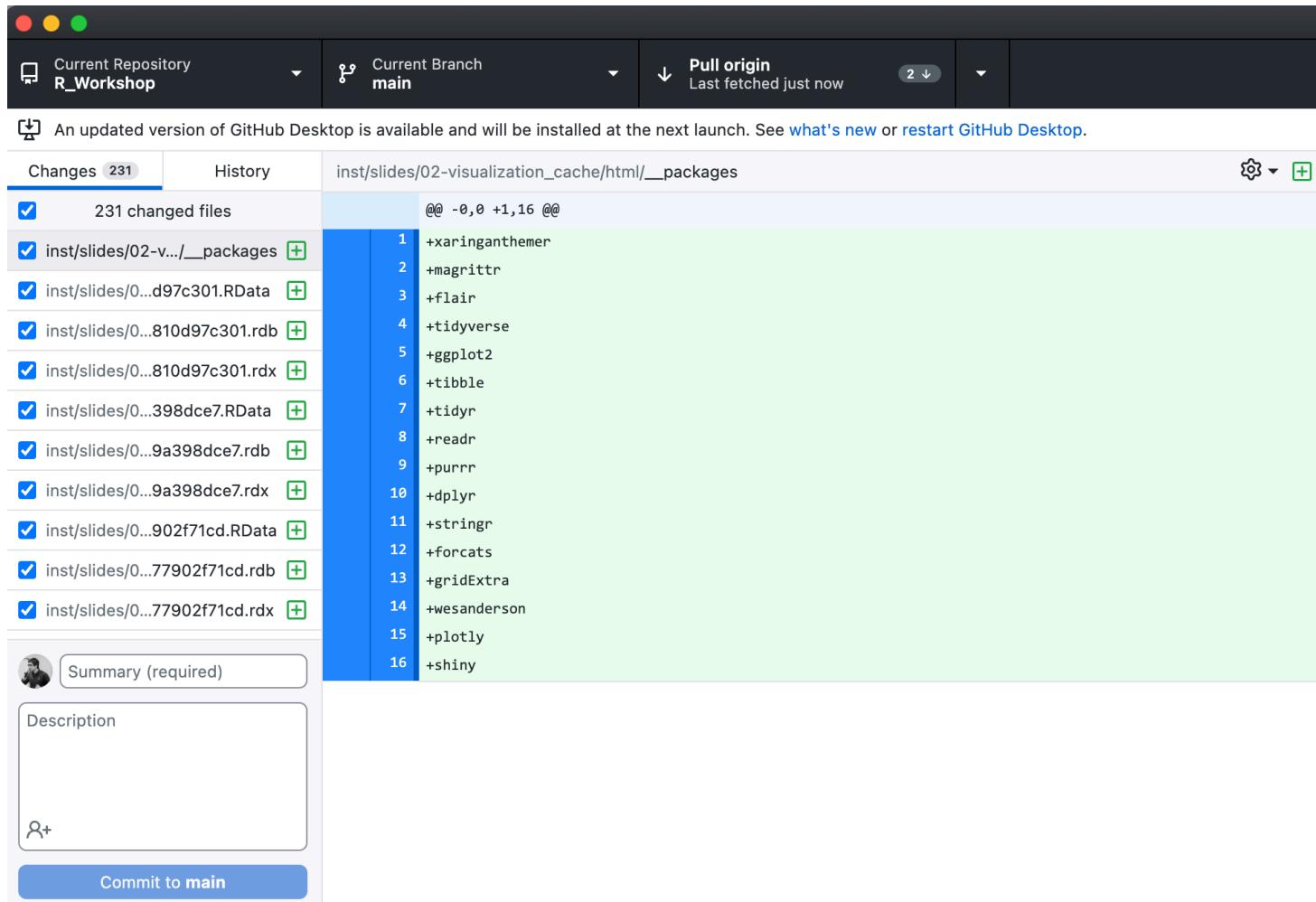
Description (optional)

Copy the `main` branch only

Contribute back to `littlehifive/R_Workshop` by adding your own branch. [Learn more.](#)

Create fork

Bonus: Pull from GitHub



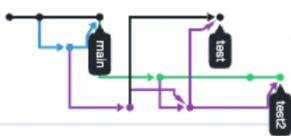
Bonus: Network graph

Projects Wiki Security Insights **Insights** Settings

Pulse
Contributors
Community
Community Standards
Traffic
Commits
Code frequency
Dependency graph
Network
Forks

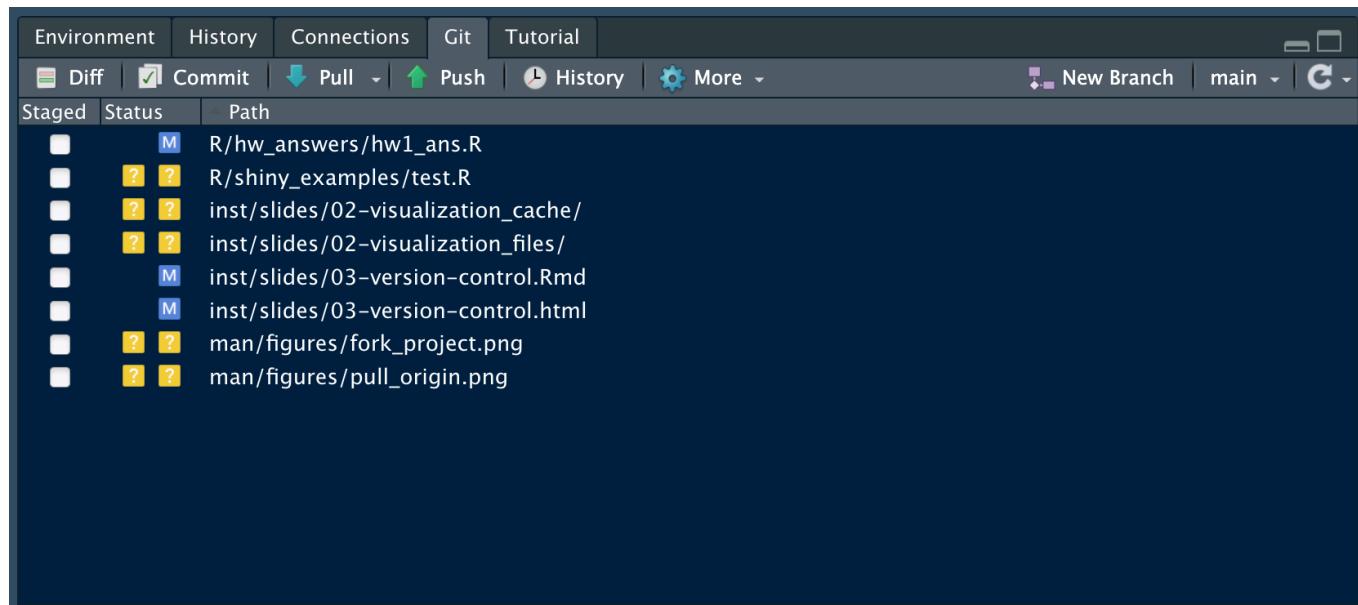
Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

Owners	Aug
littlehifive	 <p>The graph illustrates the timeline of recent commits across multiple repositories. The x-axis represents time in August, with specific dates like Aug 7, 1889, and test12 marked. The y-axis lists the repository owners: littlehifive, unim, 1889, and test12. Commit arrows connect nodes between these owners, showing the flow of code changes. For example, there are purple arrows from littlehifive to unim and 1889, and a green arrow from 1889 to test12.</p>

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.