Econometrics for Economics and Finance

Git and GitHub

Luc Clair
University of Winnipeg | ECON/GECON 3201

Preliminaries

- The necessary software to follow this lecture are
  - → **R**
  - → **RStudio**
  - → **Git**
  - → Created an account on **GitHub**
  - → **GitHub Desktop**

Introduction

Why is Version Control Necessary

- Version control software (VCS) is a system that helps people manage and track changes to their files over time (e.g., text documents and code files)
- It keeps track of every modification made to files, who made those changes, and when they were made
- Changes can be reverted, previous versions of files can be restored, and conflicts between different versions can be resolved

Git and Github

Git

Git
Version Control Software
Me telling my dog to stop begging for food

What is Git?

- Git is a popular VCS
- Allows multiple collaborators to work on the same project simultaneously, manage changes, and share work effectively
- Git’s greatest strength is its facilitation of collaboration
- It has been described as “track changes on steroids” or as “a marriage between Dropbox and Microsoft Word’s ‘Track Changes’ feature”

Git Repository

- Git tracks the changes of a set files in a Git enabled folder or **directory**
- The Git enabled directory is known as a **repository** (repo)
- One uses Git to flag any saved changes made to files in the repository

Git Repository (cont’d)

DIRECTORY
FOLDER
REPOSITORY

Changes and Checkpoints

- Git can then save a version (snapshot) of repository files at a given moment in time
- This is known as a **commit**
- Each commit has a message explaining what was changed, keeping track of the project’s progress

Branches

- Git allows you to create **branches**, which are akin to an alternate version of the repository to experiment with new ideas
- Once you're happy with the changes, you can **merge** these branches back into the main project
- A branch is a separate line of development
- The main branch is usually called main or master

Collaboration

- You can store your repository on a remote server (e.g., GitHub) and everyone can access it
- When someone makes changes, you can **pull** those changes into your local repository (folder on your computer)
- Similarly, when you make changes, you can push them to the remote server so your team can see and use them

Track and Undo Changes

- Git tracks every change you make
- If you realize you made a mistake or want to revisit an earlier version of your work, you can easily go back to previous commits
- This helps ensure you don’t lose valuable work and can always correct errors

Github

- GitHub is a web-based platform that helps researchers manage, share, and collaborate on code projects using Git
- It provides a cloud-based space where you can store your Git repositories and interact with other developers' projects
- GitHub has become one of the most popular platforms for open-source and private software development, but it's also used for data science, documentation, and more
- It was used to develop R Studio

Github

Key features

1. **Hosting Git repositories**: GitHub stores your Git repositories in the cloud, making it easy for others to collaborate on your project or for you to access it from different locations
2. **Collaboration tools**: GitHub allows multiple people to contribute to a project

Github

Key features (cont'd)

3. **Pull requests**: A way for contributors to suggest changes
    - The project maintainer can review the changes, discuss them, and decide whether to merge them into the project
4. **Issues**: A way to track bugs, enhancements, or any tasks related to the project
5. **Code Reviews**: Before accepting contributions, you can review and discuss the changes to ensure code quality

Github

Key features (cont'd)

6. **Branching and Merging**: Like Git, GitHub supports creating branches for different features or experiments
    - Team members can work on these branches and later merge them into the main codebase
7. **Version Control and History**: GitHub stores a complete history of your project, so you can see what has changed over time, who made thechanges, and when

Github

Key features (cont'd)

8. **GitHub Pages**: GitHub provides a free hosting service for static websites, allowing you to build and deploy websites directly from your GitHub repository
9. **Documentation and Wikis**: GitHub lets you include documentation for your projects, and it also supports wikis where you can add more detailed, structured information about your project

Link RStudio Project to GitHub

Link RStudio Project to GitHub

- There are a couple of ways to start our workflow
    1. Create a new Git repository on GitHub, then link it to an RStudio project
    2. Create a new RStudio project, then link it to GitHub
- Focus on number 1 for now

Link RStudio Project to GitHub (cont'd)

- Why do it this way?
  - → The first half of this workflow is applicable to all projects, not just those in RStudio

Create Git Repository on GitHub

1. Go to **github.com** and sign in

- On your home page, click the
  button

Create Git Repository on GitHub (cont’d)

2. Enter the name of the repository

Create Git Repository on GitHub (cont’d)

3. Choose whether you want the repository to be public or private

Create Git Repository on GitHub (cont'd)

4. Add a README file, which gives a description of your project

Create Git Repository on GitHub (cont’d)

5. Choose a license, then press **Create**

Create Git Repository on GitHub (cont’d)

- Your repository has been created
- Here are a few things to notice

Create Git Repository on GitHub (cont'd)

- Git defaults to the `main` branch

Create Git Repository on GitHub (cont'd)

- The commit message and commit ID

Create Git Repository on GitHub (cont’d)

- Two files were created: LICENSE and README.md
- Note the files created have the same Git message

Create Git Repository on GitHub (cont'd)

- LICENSE refers to the license chosen when creating the repository

Create Git Repository on GitHub (cont'd)

- README.md is the landing page for your repository
- It is where you describe your repository (more on this later)

Clone GitHub Repository in GitHub Desktop

- Open GitHub Desktop and log in (if you are currently logged out)

Clone GitHub Repository in GitHub Desktop (cont’d)

- On the left-hand-side, you will notice a list of your GitHub repositories
- Choose the repository you wish to make a copy to your local computer
- Then select “Clone **username/repo**”, where **username/repo** is your GitHub username and repo is the name of the repository
- In this example it is **clairl/demo**

Clone GitHub Repository in GitHub Desktop (cont'd)

- Choose the location for the clone of the GitHub repo on your local computer and click **Clone**

Clone GitHub Repository in GitHub Desktop (cont’d)

Link GitHub Repository to RStudio

- Open RStudio and create a new project

Link GitHub Repository to RStudio (cont'd)

- Choose **Existing Directory**

Link GitHub Repository to RStudio (cont’d)

- Choose the location of your GitHub repo on your local machine and click **Create Project**

Link GitHub Repository to RStudio (cont'd)

- We are now free to work on our repo files in RStudio

Link GitHub Repository to RStudio (cont’d)

- Note the two new files created: `.gitignore` and `demo.Rproj`

Link GitHub Repository to RStudio (cont'd)

- `.gitignore` files are special files where we include a list of files we want to exclude from our GitHub repository
  - → We will cover this later
- `demo.Rproj` is the R project file

Link GitHub Repository to RStudio (cont’d)

- Check Github Desktop

Link GitHub Repository to RStudio (cont'd)

- We see our two newly created files listed on the left-hand-side
- The green plus signs indicate that these files were added to the repo
- We can click on each file to observe the changes made
- New text is highlighted in green, old text is highlighted in red
- Note: LICENSE and README.md were created during the initial commit and we did not alter them

Link GitHub Repository to RStudio (cont'd)

- Add comment for the commit, e.g., "Linked RStudio project"

Push to Origin

- Origin refers to the origin repo, which is the repository hosted on GitHub.com
- Press **Push origin** to update files on GitHub.com

Push to Origin (cont'd)

- `.gitignore` and `demo.Rproj` are now on GitHub.com
- Note the different commit messages and commit ID

Make Local Changes

Local Changes

- **Local changes** refer to changes made to files saved on your computer
- When we make changes to project files, we want to update our GitHub repo to reflect these changes
- Return to RStudio and create a new R script and add the code `2+2`
- Save the new R script as `addition.R`
- Note that the new R file is saved in the project directory

Local Changes (cont’d)

Review files and Commit (cont’d)

- `addition.R` is a new file, which is why we see green plus sign

Review Files and Commit (cont’d)

- Add a message, e.g., “New R Script” and press **Commit to main**
- Then, push to origin

Review Files and Commit (cont'd)

Revert Commits

Revert Commits

- Say we've made changes to our project, but we come to regret them
- We want to revert to an older version
- E.g., delete `addition.R`

Revert Commits (cont’d)

Revert Commits (cont’d)

- The red minus sign indicates that the file has been deleted
- Write a comment (e.g., “Delete addition.R”), commit the change, and push to origin

Revert Commits (cont'd)

- `addition.R` has been removed from the origin repo

Revert Commits (cont’d)

- To revert the change, return to GitHub Desktop and click the **History** tab
- Right click the commit you want to revert and select **Revert changes in commit**

Revert Commits (cont’d)

- This reverses the previous commit and creates a new one reinstating the deleted file
- Push this new commit to origin

Revert Commits (cont'd)

- `addition.R` is back in the repo

Collaboration

Collaboration

- Most real-world projects involve teamwork
- Git and GitHub offer robust tools to manage shared work
- Reduce errors, supports transparency, and tracks progress
- There are a number of ways in which we can collaborate using Git
- Most widely recommended workflow is **branching** and **pull requests**

Collaborators

- We can invite collaborators by going to our repository on **GitHub.com** and selecting **Settings > Collaborators**
- Click **Add people**

Collaborators (cont'd)

- Search for team members and click **Add to repository**

Collaborators (cont'd)

- As a collaborator, you can accept an invitation to join a project

Collaborators (cont’d)

Collaborators (cont’d)

- To work on a project, follow the steps to clone a repository under **Git Repository to RStudio Project** above

Branches

What is a Branch?

- A **branch** is an independent line of development in your project
- By default, every Git repository starts with a branch named `main` (older repos may use `master`)
- You can create a *new branch* to work on a feature, fix, or experiment without touching the main codebase or files.

What is a Branch? (cont'd)

- Creating a new branch is akin to creating a parallel universe for your project
    - → It creates a snapshot of the project from the last commit
    - → You can make changes, commit them, and those changes will only exist on the new branch
- When finished, you can merge the branch back into the `main` branch

Branching and Pull Requests

- Workflow:
    1. Each contributor creates their own branch
    2. Make changes and commit regularly
    3. Push branch to GitHub
    4. Open a pull request (PR)
    5. Review, discuss, and merge into `main`

How to Create a Branch

- In GitHub Desktop, go to **Branch>New Branch**

- Note that the new branch will be based on the current branch, which is the `main` branch

How to Create a Branch (cont’d)

- Name your branch, e.g., `new_branch` and click **Create Branch**

How to Create a Branch (cont’d)

- We are now working on a new branch
- To publish your branch on GitHub, press **Publish branch**

How to Create a Branch (cont’d)

Make Changes on Branch

- We can now edit files without altering the main branch
- E.g., add a comment to your R script

Make Changes on Branch (cont'd)

- Open GitHub Desktop to review saved changes, commit, and push to origin

Make Changes on Branch (cont'd)

- On GitHub we can compare the .R file on the two branches

.

Main Branch

.

New Branch

Make a Pull Request

- Now that we’ve made a change, we want to incorporate into the main branch
- To do this we will make a **pull request**, a request to have changes reviewed, discussed, and accepted into the main branch
- There are multiple ways to do a pull request, however, we will focus on using GitHub Desktop

Make a Pull Request (cont'd)

- In GitHub Desktop, click **Branch > Create Pull Request**

Make a Pull Request (cont’d)

- This will open a GitHub.com window in your browser
- You can add a comment (if you want) the click **Create pull request**

Make a Pull Request (cont'd)

- Collaborators can then review changes made by looking at **Files changed**

Make a Pull Request (cont’d)

- If you want, you can comment on specific lines within the document

Make a Pull Request (cont’d)

- Once reviewed, you can leave a comment, approve the changes, or request changes
- Click **Submit review**

Make a Pull Request (cont’d)

- Other collaborators can review the pull request and comments made by other team members

Make a Pull Request (cont’d)

- Finally, they can accept the changes once everyone is happy

Make a Pull Request (cont’d)

- Once the changes are accepted, we are given the option to delete the branch

Forks

Forks

- A **fork** is an independent copy of someone else's Git repository
- It allows people unaffiliated with the project to make edits to project files
- To perform a fork press the
  button on the repository page
- Edit a forked repository by making a clone as before

Pull Requests from Forks

- A PR from a fork is a way of saying: “Here are my suggested changes. Would you like to include them in your project?”
- To perform a pull request from a forked repository
    1. Create a branch and make changes
    2. Push to your fork
    3. Open a pull request to the original repository

README Files

README Files

- A README.md is a markdown-formatted file that serves as the main landing page for your GitHub repository
  - → It's the first thing people see
  - → It introduces your repository to anyone who visits and explains what it is, how to use it, and why it matters

README Files (cont’d)

- A good README typically answers the following:
  1. A short description or summary
  2. Installation instructions, usage examples, file structure
  3. Guidelines for collaborators, branching strategy, etc.
  4. Author info, credits, license, links to paper or website

README Files (cont’d)

- GitHub automatically renders README.md in the repo’s homepage
- You can use Markdown for formatting: headers, links, lists, images

README Files (cont’d)

- Many authors use GitHub repos to share code, data, and replication materials for their research papers
- README files provide an overview of the project and how to use each file

README Example

- Can edit README files in RStudio

README Example

- Save, commit, and push to orign

Excluding Files

Excluding Files

- There may be certain files we wish to exclude from our GitHub repo
  - → Unnecessary, sensitive, or large files
- These files are necessary for our project, but we don’t want to make them public
- In GitHub Desktop, we can exclude files from our commit by unchecking the box for that file

Excluding Files (cont'd)

- E.g., temp.txt

Excluding Files (cont’d)

.gitignore

- Unchecking a box for every commit can get tedious
- Instead, we can add the file to our `.gitignore` file
- A `.gitignore` file tells Git what files to leave out of a commit

.gitignore (cont'd)

- In GitHub Desktop, right click the file name listed under "changes" and select ignore file (add to .gitignore)

.gitignore (cont'd)

- We can now see that `temp.txt` has been added to the `.gitignore` file

.

.gitignore (cont'd)

- If we want to ignore all files of a certain type, we can use the wildcard symbol `*`
- If we want to ignore all text files, we can write `*.txt` in the `.gitignore` file