Introduction to Version Control with Git

Predoc Training 2024

Sagar Saxena

The Wharton School
University of Pennsylvania

August 6th, 2024

Version Control?

- A system that records changes to files over time so that you can recall specific versions later

Version Control?

- A system that records changes to files over time so that you can recall specific versions later
  - → e.g., `thesis_draft.doc`, `thesis_final_draft.doc`, `thesis_final_draft_really_final.doc`, etc.

Version Control?

- A system that records changes to files over time so that you can recall specific versions later
  - → e.g., `thesis_draft.doc`, `thesis_final_draft.doc`, `thesis_final_draft_really_final.doc`, etc.
  - → e.g., `main_regressions.do`, `main_regressions_v2.do`, `main_regressions_v2_fixed.do`, etc.

Why do we do this?

- Earlier versions had something you might want to refer back to?

Why do we do this?

- Earlier versions had something you might want to refer back to?
- Want to try out a new approach without losing the original?

Why do we do this?

- Earlier versions had something you might want to refer back to?
- Want to try out a new approach without losing the original?
- Your collaborators are making changes and you want to keep their work separate from yours?

Why do we do this?

- Earlier versions had something you might want to refer back to?
- Want to try out a new approach without losing the original?
- Your collaborators are making changes and you want to keep their work separate from yours?
- ...

What's wrong with the non-git approach?

- Easy to lose track of who made what changes.

What's wrong with the non-git approach?

- Easy to lose track of who made what changes.
- Easy to lose track of which version is the most recent (esp. for PI or new team members).

What's wrong with the non-git approach?

- Easy to lose track of who made what changes.
- Easy to lose track of which version is the most recent (esp. for PI or new team members).
- Hard to know exactly what changed between versions.

What's wrong with the non-git approach?

- Easy to lose track of who made what changes.
- Easy to lose track of which version is the most recent (esp. for PI or new team members).
- Hard to know exactly what changed between versions.
- Hard to revert the entire project to an earlier version.

What's wrong with the non-git approach?

- Easy to lose track of who made what changes.
- Easy to lose track of which version is the most recent (esp. for PI or new team members).
- Hard to know exactly what changed between versions.
- Hard to revert the entire project to an earlier version.
- Sync conflicts?

Outline

~~1. Version Control: What and Why?~~

2. Git Basics

3. Workflow & Best Practices

4. Demo

What is Git?

Solution to all the problems we just discussed!

- Open-source version control system, developed in 2005 by creator of Linux
- Very popular: Github alone has over 420 million repositories and 100 million developers who use it
- Very powerful and flexible (we'll only scratch the surface today)

Key Components

1. Repository (“repo”)
   - → Container for your project, including all files and version history
   - → Local repository (on your computer / your collaborators’ computers)
   - → Remote repository (on a server like GitHub, GitLab, Bitbucket, etc.)
2. Commit
   - → Snapshot of your project at a specific point in time
   - → Includes a message describing the changes relative to the previous commit
3. Branch
   - → Independent line of development (e.g., `main`, `new-data-filters`, `new-IV`, etc.)
   - → Default branch is usually `main` (in older projects, it might be `master`)

Outline

~~1. Version Control: What and Why?~~

~~2. Git Basics~~

3. Workflow & Best Practices

4. Demo

Basic Git Workflow

1. Create or clone a repository
2. Make changes to your files
3. Stage changes: `git add`
4. Commit changes: `git commit`
5. Pull changes from remote repository: `git pull` (if applicable)
6. Push changes to remote repository: `git push` (if applicable)

Branching

- By default, changes (or commits) are made to the `main` branch.
- But can create additional branches to experiment with new ideas, estimation strategies, data filters, etc.
- Key commands:
  - → Create a new branch: `git branch new-branch-name`
  - → Switch to the new branch: `git checkout new-branch-name`
    - Alternatively, can use `git switch new-branch-name`
  - → Shortcut (create and switch): `git checkout -b new-branch-name`
  - → Merge changes from a branch back into `main`
    - Switch to `main` branch: `git checkout main`
    - Make sure main is up-to-date: `git pull`
    - Merge changes from the other branch: `git merge new-branch-name`

Collaboration

- Add collaborators to the GitHub repository
- Each collaborator should have a local copy of the repository (cloned repo) on their computer
- If pushing conflicting changes to the remote repo, Git will throw an error
  - → Need to pull changes from the remote repo first, resolve conflicts, and then push changes
  - → Communicate with collaborators to avoid conflicts
- Github doesn't allow large files (100 MB+) to be pushed to the remote repository so most likely you will share data over dropbox or other cloud storage; careful about modifying "clean" / "ready-for-analysis" files when experimenting on new branches

Best Practices

- Commit early, commit often, and write meaningful commit messages
- Communicate with collaborators and try and work on different files or different sections of the same file at the same time
- Commit your code before pulling changes from the remote repository
- Pull changes from the remote repository before pushing your changes
  → Esp. when working on a new branch, make sure to stay up-to-date with the main branch
- As a rule of thumb, do not put large files (esp. data files) on GitHub
  → Use `.gitignore` file to exclude certain files or folders from being tracked by Git

Outline

~~1. Version Control: What and Why?~~

~~2. Git Basics~~

~~3. Workflow & Best Practices~~

4. Demo

Live Demo

- Initialize a local and remote repo, and link them
  - → learn about .git folder, and .gitignore file
- Push and pull changes
- Revert to an earlier commit
- Create a new branch, make changes, and merge them back into `main`
- Learn about merge conflicts and how to resolve them