**Git and GitHub Tutorial**

R Workshop, Emory University

Spring 2025

Adee Weller and Eddy Yeung

What is Git?

* Git is a free, open-source version control system that helps developers track changes to their code. It's the most widely used version control system in the world. Every coder has their own codebase and can work on them at any time. You create branches, merge them, and delete (and undelete!) them with ease. This makes it very easy to track changes and revert to older versions of the code.
* It is particularly popular with software developers and collaborative teams. It is not yet widely popular with political scientists (who are scared of coding) but it **Should Be** (and is becoming more popular)!
* It is open-source and free.
* Git comes with a terminal called GitBash.

What is GitHub?

* GitHub is a cloud-based hosting service and website that helps software developers store, track, and collaborate on projects. It's a social network for programmers that encourages collaboration and sharing.
* It used Git to move files and track changes while multiple people work on the same project simultaneously.
* Software developers use it to show off their code and work ethic through their profiles.
* All files are visible by all, increasing transparency and honesty.
* You can also make free websites through it, and there are many tutorials online to help with its use.
* Also has GitHub CoPilot, and AI/LLM for coding questions.
* You need git to operate GitHub.

Go to GitHub, and show the page, click on a profile, play around on the site and get familiar with it.

Then go to [Adee’s R workshop page](https://github.com/adeeweller/R_Workshop) – that’s where we’ll be focusing.

**Opening Git**

Think of a repository or ‘repo’ as a folder where all your code and data are kept. You can share this folder and edit it as you wish. You can either take local files and make them into a repo, or you can clone one from an existing Git Repository. There will be two repos – one in the cloud and one on your device.

1. Open up the Bash on your device. For me, I type ‘bash’ into the search bar and hit enter.
   1. For a Mac, this is ‘terminal’ where GitBash is accessed
   2. (You could also use any other terminal of your choice, but I will focus on GitBash for PC and Terminal for Mac for this tutorial.)

**Configure Git**

1. Add your name and email in the global settings:
   1. git config --global user.name “Adee Weller”
   2. git config --global user.email [adee.weller@emory.edu](mailto:adee.weller@emory.edu)
   3. git config --global init.default branch main
      1. this sets up the default branch that Git will use
   4. If you need help for any command, try <command> -h or help <command>
   5. To clear the terminal, just type ‘clear’

**Creating a Local Git Repository**

1. Move to the working directory where you want your project to be stored:
   1. cd pathfile
      1. if the folder doesn’t exist yet, make one:
         1. mkdir foldername
   2. for PCs, folders end in ‘/’
   3. for Macs, folders are just words
2. Initialize Git repo:
   1. git init
      1. this will initialize an empty Git Repo ‘/.git/’
3. Check if that worked:
   1. git status

**Make a new file (working files**)

1. Create a new blank file to practice moving code.
   1. touch newfile\_AW.txt – creates a blank file named newfile.txt in your wd.
      1. Instead of AW, put your own initials
2. Check if this worked:
   1. Ls
3. To rename if needed:
   1. git mv “New\_name” “Old\_name”

**Add that file to the staging environment**

Think of the staging environment like you’re putting the code in the wings of a stage, ready to be ‘pushed’ in front of an audience, but still behind the curtains. That ‘push’ or commit is a record of what changes you are making when you push code from the staging environment onto the cloud. In the past, this was sometimes called the ‘index’.

1. Add the file you made to the staging environment:
   1. git add <filename>
      1. git add newfile\_AW.txt
2. You can check if this worked:
   1. git status
      1. This should show that one change is ready to be committed
      2. Any other changes there will be tracked
3. To unstage a file:
   1. git rm --cached <filename>
4. To track all the files in the working directory:
   1. git add .
5. To organize files that Git ignores altogether (such as sensitive data or identifying information), create a “.gitignore.txt” file with the names or types of the files to be ignored. There is a lot of information online for suppressing different file typesaccording to your own needs.

**Commit Changes (log of changes)**

1. Commit the changes with a comment to tell collaborators (and yourself) what changes are being made:
   1. git commit -m “blank file”
2. Check with “git status”
3. If there are differences, check what the old and new changes are using:
   1. git diff
4. To get an earlier version, type:
   1. git restore --staged <filename>
   2. if you accidently deleted a file, this can restore it!
5. You can also skip staging to commit directly:
   1. git commit -a -m “message”
6. To see all changes, use “git log” or “git log --oneline”
7. To go back to an earlier version,
   1. git reset ###### , using the numerical ID for the version you want from gitlog

**Branches**

A branch is a copy of a repo, but which can be edited without affecting the main branch. Once the bug is fixed or other issues addressed, a branch can be merged with the main branch without damaging the core code.

1. To create a new branch, use ‘git branch NameBranch’
2. To check, use ‘git branch’
   1. The branch you are currently in is marked with an \*
   2. To move branches, type ‘git switch NameBranch’
3. Manually edit “newfile\_AW.txt” and close.
4. Commit the change to the branch.
   1. git add .
   2. git commit -m “newfile change”
5. Let’s merge the branches!
   1. git merge -m "merged sandbox back to main" NameBranch
6. Delete branch once done:
   1. First move to a different branch using ‘git switch’
   2. git branch -d NameBranch

**Making your Own Repo in GitHub**

You can make a new cloud repo from GitHub’s website.

1. Create a repo using “New” on the Repositories page.

Let’s push the repo we have been working on to the new cloud repo.

1. Let’s establish a remote link to an existing one using the command line.
   1. git remote add origin LINKHERE
2. Pull data
3. Set the target branch
   1. git branch -M main
4. Push the code we’ve been working on:
   1. git push -u origin main
   2. Or for all of the branches, ‘git push --all’
5. To **pull** edits main in GitHub or by other users:
   1. git pull
      1. these changes will now be stored on your local repo. I recommended beginning all sessions with this, and pushing them throughout but certainly at the end.

**Working with an Existing Repo**

You can also link up with (‘clone’) an existing repo in GitHub. Here, we’re going to pull from an existing one: <https://github.com/adeeweller/R_Workshop/tree/main>. Cloning pulls a full copy of a repo from GitHub to your local repo, as well as all versions and commit history.

1. Go to the main page of the repository that you want to code, and click the green ‘Code’ button.
2. Save the HTTPS link.
3. Open GitBash (PC) or Terminal (Mac) and navigate to your working directory.
4. Clone the Repo:
   1. git clone LINK
   2. git clone <https://github.com/adeeweller/R_Workshop.git>
5. Now all the files should be on your local repository. Pull for updates, commit changes as normal.
6. If you are having problems, check which remote you are using:
   1. git remote -v
   2. Or maybe you need to update the url:
      1. git remote set-url origin LINK
      2. git remote set-url origin <https://github.com/adeeweller/R_Workshop.git>
   3. Pull with rebase to apply local changes:
      1. git pull --rebase origin main