Intro to Git and GitHub

PSU Center for Infectious Disease Dynamics Workshop

Callum Arnold

2/12/23

Table of contents

Welcome, and what this book is about

This book accompanies the Center for Infectious Disease Dynamics short workshop on using Git and GitHub as researchers. The content will mirror much of the workshop's syllabus, and act as a reference for attendees (and others), although I would highly recommend reading through the excellent book https://happygitwithr.com by Jenny Bryan and co.

Pre-requisites

There are some pre-requisite tasks to get set up ahead of the workshop.

1. What are git and GitHub, and why do I care?
2. How to install Git
3. Setting up a GitHub account
4. Connecting GitHub to my machine

TOC

After getting setup, the workshop will cover the following sections:

1. An overview of how git works
2. Making your first git repository
3. Git mechanics - how to actually use git
4. Git branching
5. Recommended practices

1 Pre-Requisites

1.1 What is Git?

If you're in this workshop, or have stumbled across this book, there's a good chance you already know what git is, or have at least heard of it. However, if you don't and you've been told by someone you should start using git, but have no idea what that even means, then hopefully this subsection will help.

Git is a version-control system (VSC). Think of it like a better version of Microsoft Word tracked changes and Google version history that can track everything from code, to text, to pdf images. Much like how tracked changes is useful for both single and multi-user documents, git can help us remember what we've done, when, and what version of the document(s) previously existed, as well as denoting which user made the changes. Where tracked changes can get unwieldy after multiple iterations, git makes it easy to understand the whole file history without needing to use `document_v3` filenames - just keep changing the same original file for as long as you want! In addition to just being able to understand a file history, when working on a project, even if you're the only one coding, it's important to be able to go back to previous versions if you make a mistake. This is possible with git! Git isn't the only VCS available, but it's the most prevalent, and has a good support community, so is what will be the focus in this book.

1.2 What is GitHub?

Hopefully I've convinced you that git is a useful addition to your research, so now let's turn our attention to GitHub. GitHub is a website and server system makes it easy to collaborate and share your code with the scientific community. The key feature of git is that it's a distributed VCS. What this means is that users can make changes to a file on their own computer and then `push` the updated version to GitHub so that all collaborators can then use this version of the file. Later on we'll go through the mechanics of this, including what happens when two users make changes to the same lines of code and try to `push` to GitHub, but for the moment we can just appreciate that GitHub allows us to both work offline and collaborate. There are many different remote services that can be used to host our remote code, such as Bitbucket or GitLab, but I'd strongly recommend you use GitHub over the alternatives for a number of reasons. Principally, GitHub has the largest user base, so more people will likely see your work.

With GitHub, if you ever want to make your code open-source, you immediately have access to the largest community of programmers who can help you improve your code, as well as putting it to good use. And isn't that why we do research? GitHub is also owned and backed by Microsoft. While this is a negative for some, it does result in tighter integration with Azure cloud computing, very active development of the platform with frequent improvements to the user experience (e.g. GitHub Actions that allow for easy continuous integration), and fewer data storage concerns with regards to university policies. If you really want to avoid all Microsoft based products, I'd recommend you look at GitLab.

1.3 Installing git

To get started, you first need to install git. There are many ways to get git running on your computer, but the recommended steps depend on the operating system you have.

1.3.1 Mac OS

If you're on Mac OS, you likely already have git pre-installed. However, you are unlikely to have the most up-to-date version and I'd recommend you install it manually. If you do not already use a package manager, I would suggest you download homebrew as it is the most widely used and therefore can download the most applications.

1. Open the terminal and enter `/usr/bin/ruby -e "$(curl -fsSL https:/raw.githubusercontent.com/`
2. Enter `brew install git` into the terminal
3. Enter `which git` into the terminal
    a. You should see `/opt/homebrew/bin/git`, if not, you may need to edit the environment variables

Using homebrew to install packages makes it easy to update them (including git). All you need to do is type `brew update` and all your brew-installed packages are updated in one command!

1.3.2 Windows

Getting set up on Windows requires a bit more work as Windows doesn't come with a good terminal (command prompt doesn't count!). If you want to explore using Windows Subsystem for Linux (WSL), then go for it as it'd make your life easier as you get into more advanced things like cloud computing and remote servers, but for the moment, you can use the following steps to get started.

1. Install Git for Windows

- This gives you Git Bash, which is a much nicer way of interfacing with Git than the command line.
- **Note:** when asked about "*Adjusting your PATH environment*", be sure to select "*Git from the command line and also from 3rd-party software*". The other default options should be fine. For more details about the installation settings, please click here

2. Open up Git Bash and enter `where git`. Open up the command line and enter `where git`. Depending on whether you have administrator privileges, the outputs should look something like this, respectively

   1. `which git` : `/mingw64/bin/git`
   2. `where git` : `C:\Users\owner\AppData\Local\Programs\git\bin\git.exe` (User privileges)

      1. `where git` : `C:\Program Files\git\bin\git.exe` (administrator privileges)

   - If you see `cmd` instead of `bin`, then you need to edit the PATH in your environment variables.

1.3.3 Troubleshooting

1.3.3.1 Environment Variables

If you are not able to access git appropriately (i.e., from the terminal/git bash), you may need to edit the environment variables. In Windows you do this by navigating to `Environment Variables` from the Windows key/Start prompt and editing the PATH in `User Variables`. To do this, scroll to the PATH section of User/System variables (depending on whether you have administrator privileges), and changing `cmd` to `bin` in the `git.exe` path. In Mac, you should open the terminal and use `vi`/`touch`/`nano` command to edit the `~/.zshrc`/`~/.bashrc` file (depending on how old your Mac is - OSX switched to zsh in 2020) e.g., `vi ~/.zshrc`, which opens (or creates if missing) the file that stores your PATH. From here, type `export PATH="path-to-git-executable:$PATH"` to add the executable to the path. For me, using a Mac and homebrew, my git path is `export PATH="/opt/homebrew/bin:$PATH"`. Now save and exit the text editor, source the file if on Mac (e.g., run `source ~/.zshrc` in your terminal), and you're good to go.

2 Summary

In summary, this book has no content whatsoever.

References