Intro to Git and GitHub

PSU Center for Infectious Disease Dynamics Workshop

Callum Arnold

2/15/23

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

Part I

Pre-Requisites

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! In fact, this book was created using git and GitHub, so you can explore how it was put together and iterated by going to the GitHub page. 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. In git terminology, GitHub is your **remote**. 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? As an academic or student, you can get a free **PRO** account, meaning unlimited collaborators on private repositories and a bunch of other useful things that we'll touch on more later. 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 probably make your life easier as you get into more

advanced things like cloud computing and remote servers (see here for more details), 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.

You could also install git using Chocolatey, as this would provide you with a package manager that you can use to install other useful software, much like homebrew on Mac OS.

1.3.3 Final Git set up steps

Now that you have Git running, you need to tell it who you are. This allows multiple people to make changes to code, and the correct names will be attached to the changes.

Open up the Git Bash or the terminal and enter

```
git config --global user.name 'Firstname Lastname'
git config --global user.email 'my_email@domain.com'
```

Typing in `git config --global --list` is a way to check that your details have been saved correctly.

> **Note:** the email you use with git will be published when you `push` to GitHub, so if you don't want your email to be public, you can use the GitHub-provided no-reply email address instead. The key points are that you need to turn on email privacy in your GitHub settings, and then using that address in your git config.

On another note, if you would prefer to use a different user name than your GitHub user name you can. This would help show you which computer you completed the work on, but it is not important to most people.

1.3.4 Troubleshooting

1.3.4.1 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 2019) 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.

1.4 Installing a Git Client

You've now installed git, and you're ready to get going! Much like most pieces of software, we can interact with git via a command line or using a graphical user interface (GUI). There's a small vocal minority of people that proclaim that you can't learn git with a GUI (aka git client), but don't listen to them! There are plenty of good git clients out there that make the basic commands simple, and provide a visual for more complicated ideas. I prefer to use the GitKraken client, which is free to use for students and academics if you sign up to the GitHub developer pack, but only allows access to a limited number of private repositories otherwise, so you may want to explore other options if that's you. GitHub for Desktop is made by the GitHub team, so as you can imagine it is tightly integrated with GitHub. Whether this is a pro or a con will depend on whether you think you'll explore other remote hosting services, but the reason I've avoided it is that it doesn't have a method of visualizing branches. You'll have to decide for yourself if this is a deal-breaker for you and your workflow. If you use VSCode as your development IDE, there's a built in git client (the **Source Control** panel), and you can install the **Git Graph** extension to visual branches. The **GitLens** VSCode extension is also worth installing, and now offers paid features to further extend its use, although if you have a GitKraken Pro account you get these for free. In case you come across it in

other recommendations, SourceTree was another good alternative, but I have had some issues connecting to some GitHub accounts, and it has limited support, so I have since moved away from it.

As you get a better understanding of git, you may want to use the command line as it can be quicker to use, and is more powerful. If you decide to go this route, I'd recommend looking at the lazygit plugin which brings a GUI to the terminal, enabling you to get (most of) the best of both worlds. It's a little out of the scope of this workshop, but there are some useful video tutorials linked on lazygit's GitHub page.

1.5 Setting up a GitHub Account

It's very easy to get set up on GitHub. Just click the link above and select the package you'd like. If you have an academic email address, consider making this your primary email address on the account, as it gives you a **PRO** account for free with access to more features. See here for more details about the differences. If you are a student, you should also sign up for the GitHub Student Developer Pack as this gives you free access to a bunch of useful tools, such as the GitKraken git client mentioned earlier. If you are a teacher, you get access to a GitHub Teams account, which also comes with its own set of benefits (see here for more details).

Be sure to choose a user name that is easy to remember, and easy to find. I would suggest just using your name, or the username you have for other work-related accounts (e.g., Twitter). It's quite annoying to try and change this later, so spend a little time thinking about it now.

> **Note:** You can choose a public-facing name that is different to your username, so you can just use your full name here if yours is long and you don't want to use it as your username.

Now you have a GitHub account set up, this is your *remote*. If you work on a project with collaborators, this can be shared with them. That way, collaborators can work on their own versions of the code on their *local* machine (computer), and when it's ready for other people to use/help write, they can `push` it to the *remote* where others can access it. Don't worry if you don't know what `push` is - we'll cover that soon.

1.6 Linking Git to GitHub

Now you have a GitHub account, you need link it to your local git installation. There are a couple of different methods for doing this. The first is to use HTTPS and Personal Access Tokens (PATs). This is somewhat easier to set up, but it's a little less secure and ultimately is more annoying to use as it often requires entering your username and password each time you connect to GitHub. The second is to use SSH, which is a little more complicated to set up,

but is more secure and easier to use once it's set up. Unlike Jenny Bryan, I think it's worth the effort to set up SSH as you front load the work (though it's not too bad), and then you can just forget about it. Knowing SSH basics is also really useful as it's the basis for many other things, such as connecting to remote servers and computing on clusters, so will serve you well moving forward.

Instead of trying to cover all eventualities for all operating systems, please work through the excellent GitHub docs on PATs and SSH. If you are having issues after working through the steps, then try reading Jenny Bryan's excellent git guide, which covers both PATs and SSH. If nothing here works, let me know and I'll try to help!

2 Basic Git Commands

References