# Introduction to Git

EDH7916

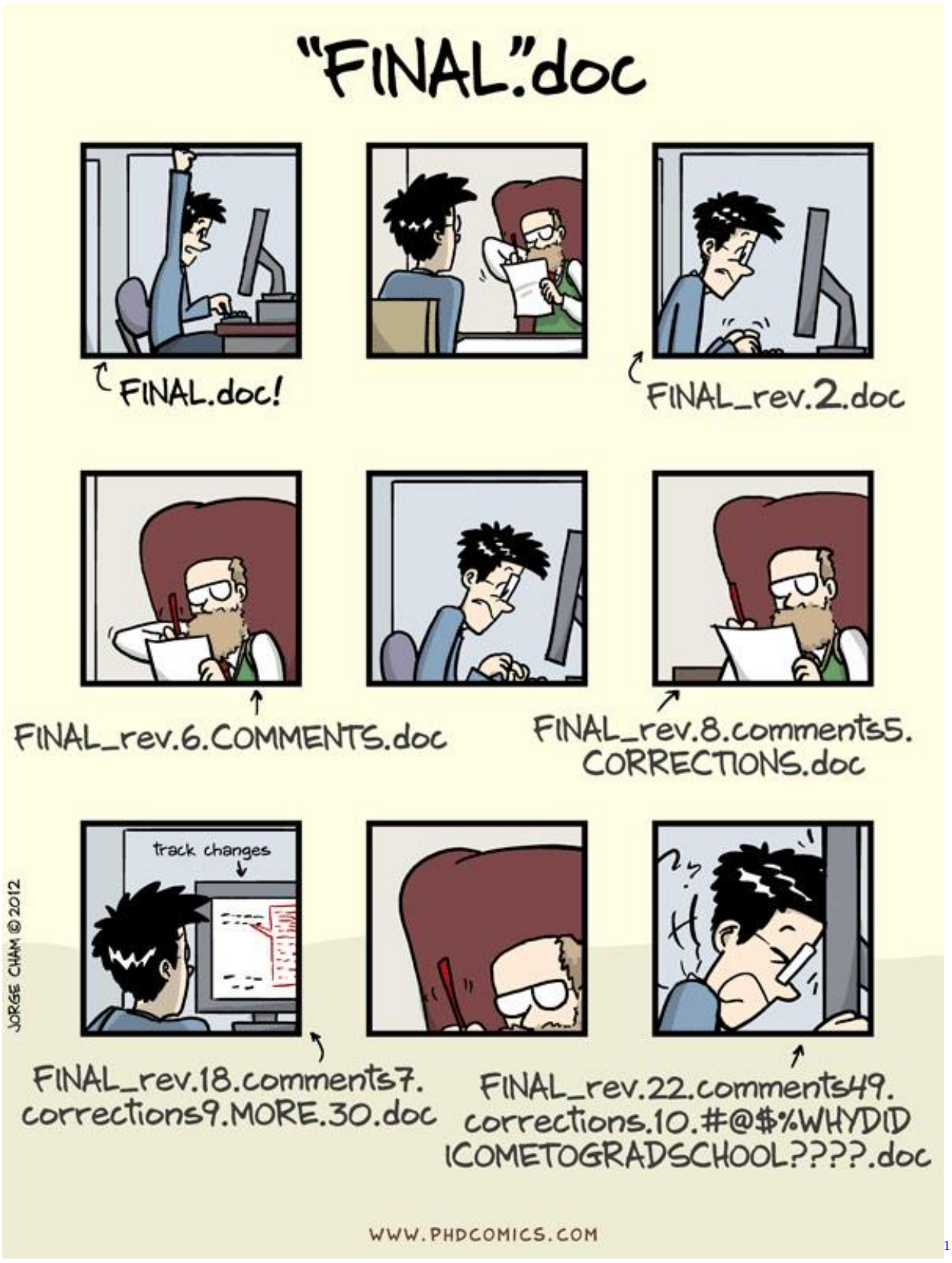
Benjamin Skinner

From the git website http://git-scm.com/

> *Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.*

## Why use git?

With so many other (read: easier!) ways to share and collaborate on documents, why use git? Isn't it a bit overkill to learn an entirely new syntax? Why not just email files or use something like DropBox? Because it is very easy to end up with something like this:




[1]

[1] http://www.phdcomics.com/comics/archive.php?comicid=1531

*Credit: Jorge Chan*

**As complexity increases, so does the need for git**

$$Project = f(size, scope, collaborators)$$

As any part of that function grows, so too does the need for a work flow that:

- Allows for many moving parts
- Allows for peaceful collaboration (no overwrites)
- Allows for timely collaboration (synchronous)
- Allows for distant collaboration (centralized file location)
- Allows for version control (so you can go back if necessary)

Git was designed to do all these things, so it works better than other systems.

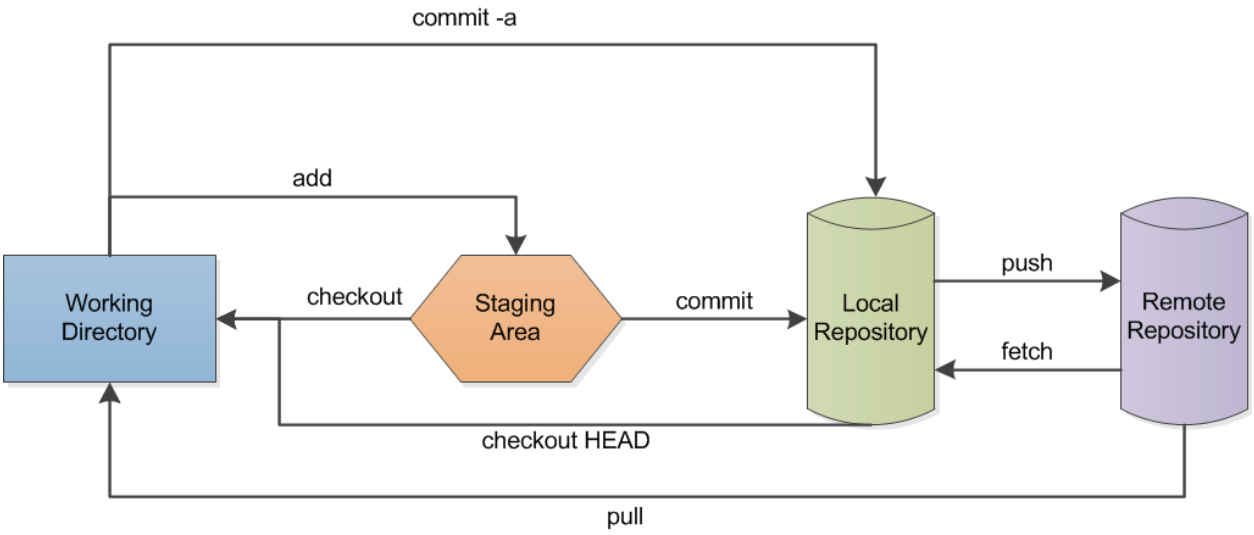
**What's the difference between git and GitHub?**

Yep, you guessed it. Git is the system/language that supports version control. GitHub is an online service that will host your remote repositories, for free if public or a small fee if private. (Students get an education discount and some free repos. Check out https://education.github.com/.)

RStudio is nice because it provides an nice point-and-click interface for git. (Sourcetree[2] and GitHub Desktop[3] are also really nice GUIs.) If you want to run git at the command line, go for it! But using RStudio or another GUI is fine.

[2] https://www.sourcetreeapp.com
[3] https://desktop.github.com

**How does git/GitHub work?**




commit -a: Directly commit modified and deleted files into the local repository (*no new files!*)
add: Add a file to the staging area.
checkout: Get a file from the staging area.
checkout HEAD: Get a file from the local repository
commit: Commit files from the staging area to the local repository
push: Send files to the remote repository
fetch: Get files from the remote repository
pull: Get files from the remote repository and put a copy in the working directory

[4]

*Credit: Lbhtw (Own work)*

**Some notes on work flow (good habits)**

1. Always `pull` from your local repo with your remote repo before starting any work. This makes sure you have the latest files. RStudio has a `pull` button in the **Git** tab.
2. Don't wait to `push` your commits. Just like you save your papers every few lines or paragraphs, you should push to your remote repo. This way, you're less likely to lose work in the event of a computer crash. Also, should you want to return to a prior version, small changes make it easier to find what you want.

3. Add useful `commit` messages. RStudio will make you add something. Don't say "stuff." A version history of 95 "stuff"s is pretty non-helpful. Also, I would keep it clean. Even in a private repository, work as if someone will see what you write someday.

4. Don't freak out! If you accidentally push and overwrite or delete something on the remote, you can get it back. That's the point of version control! Sometimes weird things like merges happen (two files with the same name are shoved together). They can be a pain to fix, but they can be fixed.
5. Avoid tracking overly large files and pushing them to your remote repository. GitHub has a file size limit of 100 MB per file and 1 GB per repo. The history of large files compounds quickly, so think

[4] https://commons.wikimedia.org/wiki/File:Git_data_flow.png

carefully before adding them. Usually this means that small datasets are okay; big ones are better backed up somewhere else and left on your machine.

6. Remember: even in a private repository, your files are potentially viewable. If any of your datasets or files are restricted, do no push them remotely. Use `.gitignore` to make sure that git doesn't track or push them.

Every class and work session should go like this:

1. `pull` from GitHub remote
2. *do work*
3. `add` or `stage` (if new file)
4. `commit` with message
5. `push` back to GitHub remote

If you'd like you can also use terminal commands to accomplish the same things. Many people (including me) like to use the terminal commands directly.

- Pull: `git pull`
- Stage: `git add <filenames>`
- Committ: `git commit -m "<your message here>"`
- Push: `git push`

That said, you should use whatever works best for you.

By far the most useful guide to working with R and git/GitHub is Jenny Bryan's guide[5]. In fact, we'll use it first to set up your user credentials so you can interact with GitHub.

**Quick exercise** *Using these instructions from Jenny Bryan's site[6], let's set up a Personal Access Token or PAT that you'll save to your machine.*

### Some more notes on using Git: plain text

Git works best with plain text files[7]. This is because it notes the differences across two plain text files rather than just copying and re-copying the same file over and over again as it changes. When you've only changed one word, git's method of version control is much more efficient than making a whole new copy.

It's also useful when you need to merge two files. If file B is just file A with new section, file B can be easily merged into file A by inserting the new section — just like you would add a paragraph in a paper you're writing. R scripts are plain text. Some data files, like *.csv, are plain text. This is why git works really well with data analysis workflows.

On the other hand, git does not work as well with binary files[8]. These files are stored in a format closer to what your computer understands, which comes with benefits. Data files, like Stata's *.dta and R's* .Rdata, as well as MS Office files — *.docx,* .xlsx, *.pptx, *etc* — are binary files. Git will keep track of your MS Word document, but due to its underlying structure, you won't be able to merge and every small change will just make a whole new copy of the file. This is why we generally don't commit large binary data files to Git: your repo just becomes larger and larger with each small change to your data.

[5] https://happygitwithr.com/
[7] https://en.wikipedia.org/wiki/Plain_text
[8] https://en.wikipedia.org/wiki/Binary_file

**Not-so-quick exercise** *In this first exercise, we'll practice using RStudio to pull down your student repository from GitHub. We'll also practice making a few quick file changes, staging, commiting, and pushing them back to the remote.*