Missing Semester Lecture 5 Notes

Version Control tracks changes to source code, facilitates collaboration
- How familiar is everyone with git? (Usually it varies a lot by person)
- Git tracks snapshots of things, and useful metadata (time machine)
- Useful for LOTS of different kinds of work (even by yourself)
- *Understanding* git's data model makes it easy to use its interface
- Model: Trees (folders) and blobs (files), like your OS structure
- Git thinks in history snapshots (what you tell it to know when/where)
- Its graph-theory model also has metadata about authors, notes, etc.
- Git snapshots == commits, which have parents, data, and metadata
- Git **object** types: blob, tree, or commit. That's it!!!! (mostly)
- Note the underlying memory model, links with pointers (fancy, refs)
- Note that git history is immutable, but git **references** ARE mutable
- Objects + references = a git **repository** (that's pretty much it)
- ALL git **commands** manipulate references's data OR objects' data

Using git through a simple **demo**, working with the actual git interface
- Does everyone understand the staging area vs. commits?
- Note that it can be REALLY useful to control what you add/commit
- HEAD is a special reference in git to "what you're looking at now"
- Github and GitLab are hosts for git repositories (remote vs. local)
- `git checkout` lets you *move around* in your commit history
- Remember that `git checkout` moves HEAD and can destroy things
- `git diff` is a useful way to understand how git is thinking on the fly
- Thinking of it as snapshots is great, but note that it's more complex

Git **branches** are REALLY powerful and useful, but can be tricky to use
- Please remember to write useful commit messages (helps branches)
- Tip: `git checkout -b dog` == `git branch dog; git checkout dog`
- `git merge` is really common with branching, but THINK about it
- Merges try to work automatically, but you might run into conflicts
- Working w/ git in a web interface vs. command line is DIFFERENT!!!

Git **remotes** are "remote" copies of existing repos (collaborating = easy)
- Note: "origin" typically refers to whatever (1) remote you are using
- Format: `git push <remote> <local branch>:<remote branch>`
- Note: need to use `git branch --set-upstream-to=origin/myBranch`
- Pro tips: try to pull before you push, and commit *early* and *often*
- Advice: understand full commands before you get happy w/ shortcuts
- Note: `.gitconfig` is a thing! Can play w/, or steal people's online
- Trick: `git add -p`, then short command, then `git diff --cached`

FOR LATER REVIEW AND EXERCISES:

Guys, there's SO much that you can learn about git. You should DEFINITELY take the time to get familiar with it asap. Knowing how to really use it will save you SO much headache.

There's a ton of more info and extra practice in the Missing Semester page **lecture notes** for this (especially the **Resources** section at the bottom). Also, I highly recommend the ~3 **LinkedIn Learning** tutorials on git in Larry's HEP Starter Kit playlist here: https://www.linkedin.com/learning/collections/6702106055167504384?trk=share_collection_url&accountId=42574436&u=42574436&success=true&authUUID=4DA9IJc1T8KSFp1q9HMChQ%3D%3D

Also, really **useful talk** on git's philosophy from Larry is here: https://www.youtube.com/watch?v=4XpnKHJAok8

Just practice the stuff as it comes up, and everything will fall into place!!

We potentially will be going through the Git Branching tutorial here (with Jordan): https://learngitbranching.js.org/

OR -- Here's a nice little repo that we can all play with harmlessly (I hope): https://github.com/ldishman/git_demo
Note: Do whatever you want here, but be respectful and conscientious!