GitHub Tutorial

Foundations of Machine Learning

More GitHub

- Mostly we've been using GitHub as a public place to store files, with Google Colab as a way of editing them by making `commits` and using `clone` to grab up-to-date versions of the repo
- This is pretty limiting
- We want to make the jump to Rivanna, and use more fully featured versions of Jupyter and GitHub
- These notes are sketching out "minimum viable Git/GitHub" for people who are learning about these tools this semester
- In general, at this stage, I think you should do most key Git operations on GitHub, and constrain your local actions to `commit`, `add`, and `clone/pull/push`
- There is always more Git to learn

GitHub Repositories

- A Git repository or **repo** is a directory, like any directory on your computer
- It is special because someone has initialized a software called **Git** that performs sophisticated version control tracking on the directory
- It is not like Dropbox or Box: You have to interact with Git directly
- It is a very powerful tool, and Git is not GitHub: GitHub is a popular service that hosts Git repos, so that anyone can make copies or contribute to them

Bash commands

- What is happening when we clone a repo? This line

```
! git clone https://www.github.com/DS3001/intro
```

has the `!` symbol, which escapes from Python/Jupyter directly to the command line for the Linux virtual machine you're using

- For example, if you type `pwd` inside Jupyter, it tells you the present working directory. If you type `! pwd`, you escape to Bash, the "Born Again SHell", and get the same output, but from Linux and not Python.
- When we `! git ...`, we are interacting with Git directly

GitHub Personal Access Tokens

- A **Personal Access Token (PAT)** grants you the rights to clone a private repo and push work back to public or private repos
- Let's make a PAT now on GitHub
  - Click your icon in the upper-right corner, `Settings`, scroll all the way down to `<> Developer Settings`, then "Personal Access Tokens"
- Create a classic access token, not fine-grained, unless you are working on a specific project
- Your groups all have PAT's that I'll email for your group project repos

1. Creating a Private Repo, Cloning to Rivanna

- If you are going to work on GitHub, it is "easiest" to initialize your repos there. Let's create a private repo.
- Everyone can see and clone public repos, but private repos are visible only to you
- You can turn any directory on your computer you like into a Git repo by using `! git init`, but getting that repo onto GitHub then requires replicating some of these steps anyway
- To clone your repo to Rivanna, you use the Git command
  - `! git clone https://<Username>:<PAT>@github.com/<Username>/<Repo name>`

2. Working on a Repo

- When you make a **commit**, you take a snapshot of the repo and store it; subsequent changes then need a new `commit` to snapshot the repo again
- Commit early and often, it is very low cost
- To make a commit from Jupyter, you type `! git commit -am <Commit Message>`. Your commit message should be useful to your future self and coworkers, and the message appears in a log of changes
- To see the log, type `! git log`
- Let's edit a file and make a commit

Commit fatigue

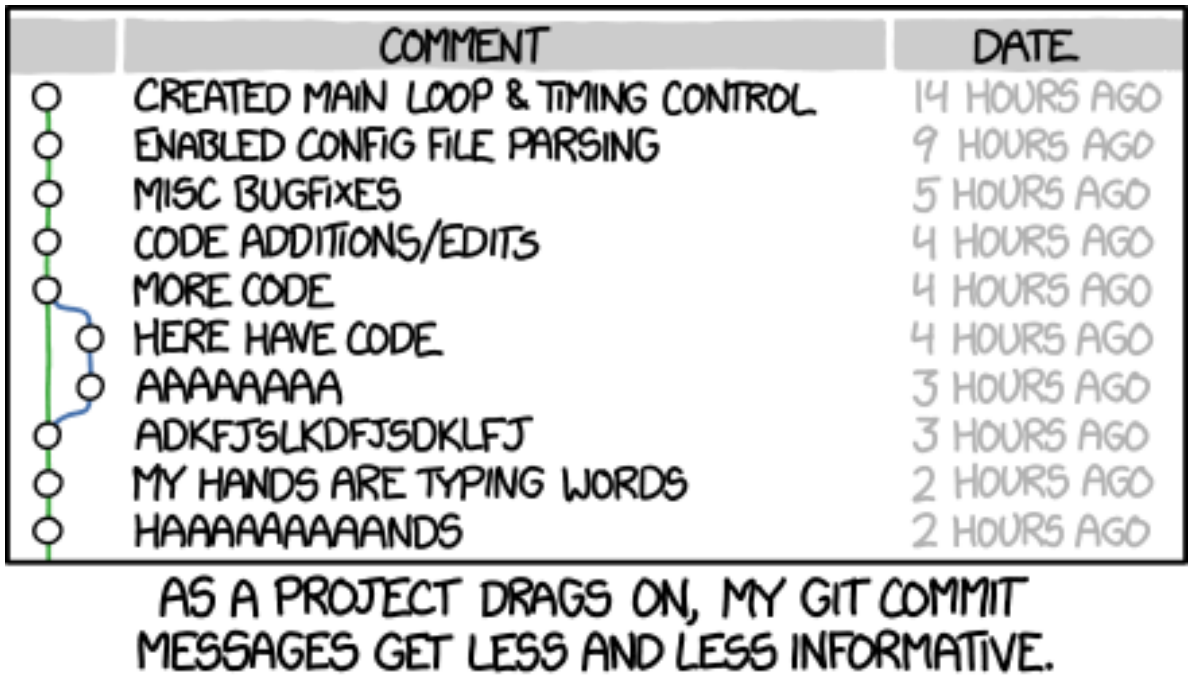

Figure 1: Git Jokes!

How does Git work?

- Whenever you make a `commit`, it saves a complete snapshot of the repo in a clever way
- As you work on your project, making `commits` as you go, it updates the snapshots, so you can take risks, knowing that you can go back

2. Working on a Repo

- Git only tracks the files you tell it to track.
  - To add a new file, type `! git add <filename>`
  - To add everything in the directory, type `! git add *`
  - To remove a file, type `! git rm <filename>`
- To see what files Git is tracking, type `! git ls-files`. This is an important command if you need to diagnose why your changes aren't making it back to GitHub

3. Pushing Work Back to GitHub

- OK, make sure you've added all your files and committed all your desired changes
- Here is how to push a repo back to GitHub (on one line):

```
! git push
https://<Username>:<PAT>@github.com/<Username>/<Repo
name>
```

- Again, you need your PAT to ensure GitHub that you're who you say you are, and are allowed to do this

4. Continued Work on a Repo

- The `clone` command tries to create a new repo, and if one already exists, you can't easily overwrite it
- Instead, you can (on one line)

```
! git pull
https://<Username>:<PAT>@github.com/<Username>/<Repo
name>
```

to update your local copy with the most recent changes on GitHub, do your work and make `commits` and `adds` as necessary, and `push` changes back

- This is the basic workflow loop

Branching

- What if you want to try making a radical change, or work in parallel with someone else without interfering with their work?
- You can **branch** the project and work on a parallel version, leaving the **main** branch where it is
- I am not talking about local branching in your repo (`git branch`, `git checkout`), but about working with branches on GitHub (this is slightly different)

Branching

- Use the repo tools on GitHub to create a new branch there
- To make a local copy of that branch, type the following (on one line)

```
! git clone --single-branch --branch <branch name>
  https://<username>:<PAT>b@github.com/
  <username>/<repo name>
```

- To push to a branch, make your commits and (on one line)

```
! git push
https://<username>:<PAT>b@github.com/<username>/<repo name>
<branch name>
```

- To see the current branches for your repo, `! git ls-remote` after cloning; all repos start with a **main** branch, and add others as the project goes along

`merge`/pull Requests

- When you push to a branch, GitHub will automatically ask if you want to merge the changes into the main branch
- This will integrate your work into the main branch automatically, if possible
- If files were edited on both branches in the same places, you get a `conflict`: Git will show you where the files don't match, and you'll have to sort out what you want to do to resolve the conflict between the files
- The solution (for this class) is just to have files for everyone and not step on each others' toes
- (A bad-case scenario is that you just manually upload a file or two that you've worked on, directly into the main branch)

Managing a Project

- In the kind of work we're doing, you can create lots of notebooks in separate branches to avoid conflicts, then harmonize them towards the end
- For example, have someone work on visualizations in a `viz` branch in a `viz.ipynb` notebook, someone work on analytics in an `analytics` branch in an `analytics.ipynb` notebook, and so on.
- As people finish, merge their work back into `main`
- Once your individual tasks have come together, branch off main again to work on your report(s), using the relevant code chunks from the notebooks to present your main results

`merge`/pull Conflicts

- I would not `git add *` with Jupyter
- Jupyter has a bunch of files that it uses for temporary purposes, and they are almost guaranteed to change a lot during a session
- If you `git add *` and those files get added to your branch repo, there will almost surely be a conflict when you go to merge
- Instead, just `git add <filename>` for the files you actually want in the repo

“I’ve done a horrible thing, and my group is going to hate me”

- Fear not, as long as you `commit` and `push` often, you can just go back to a previous version
- Go to your repo on Github and click the “(Rewind Symbol) Commits” button
- You can see the branches and commit snapshots, and clicking the <> symbol for any snapshot shows you the repo at that time
- You can copy chunks of code, download individual files, browse them, or download an entire local .zip copy, depending on what regrettable things you’ve done

Git and GitHub

- I have a 500 page book about Git and GitHub! There are lots of nuances and other options I am leaving out
- Working with Git alone is not the same as working with Git and GitHub together: Some tasks are easier, some are harder
- If you think about the `main` branch on GitHub as the "real thing," and your local copy as scratch paper to work on, it helps you to avoid making common mistakes
- It's OK to make mistakes, and if you have a Git/GitHub disaster, we can thankfully use the copies on GitHub to restore your project to a previous version

You can always just start over

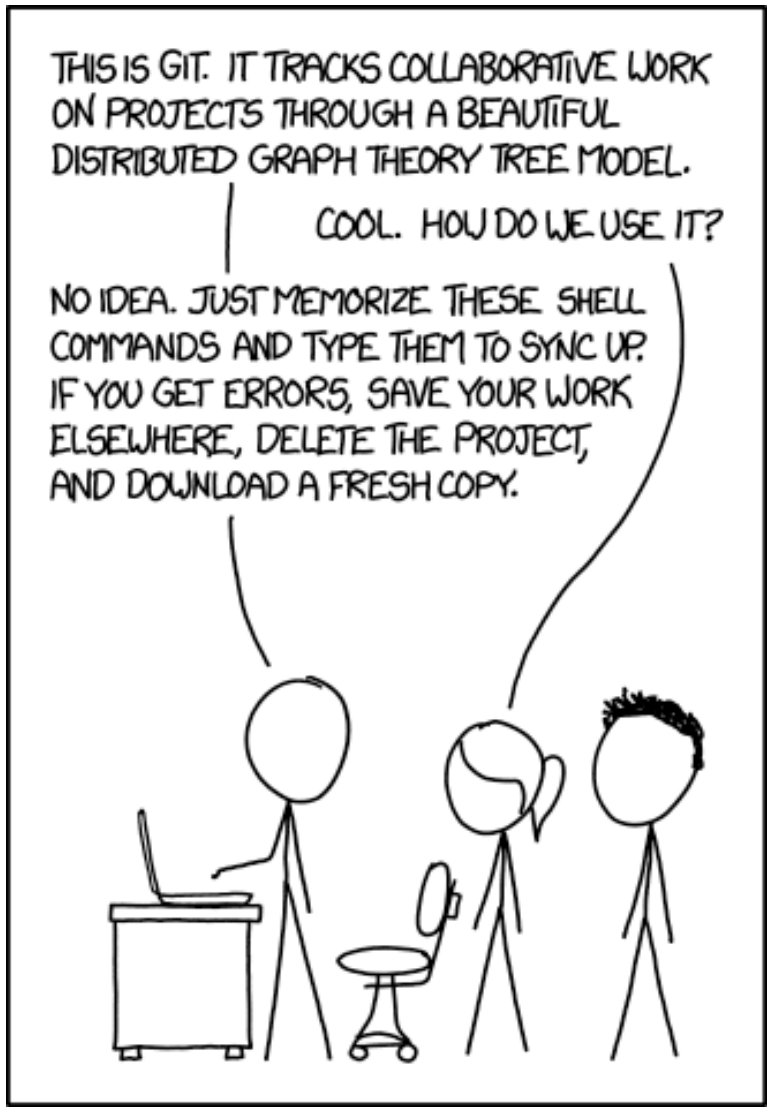

Figure 2: Git Jokes!