BF582 - git

Challenges of Writing Code

- We write lots of code
- Code evolves over time
- Other people need to see/run our code
- We work on teams and contribute to the same code base
- Employers want to see your coding skill

git

“Git is a distributed version-control system for tracking changes in source code during software development.”

git

- **Distributed** - no centralized location; every git repository contains entire change history
- **Version control** - track changes over time, control who can make changes
- **Source code** - instructions a computer can understand, and corresponding documentation (NB: *NOT DATA!*)

Core git concepts

The repository

- **Repository -** directory containing a set of files git knows to track
- Contains the current state of code, and a record of *all previous states*

```
mkdir my_git_repo
git init my_git_repo
ls my_git_repo/.git
```

What repositories track

- **TEXT FILES, AND ONLY TEXT FILES:**
  - Scripts/source code
  - Documentation (e.g. README.md)
  - Specific git files (e.g. .gitignore)
- Do not track:
  - Empty directories
  - Large files (i.e. > 1Mb)
  - Non-text files (images, binary files, etc)

The commit

- **Commit** - atomic operation for tracking code changes
- Both a noun (*make a commit*) and a verb (*commit your changes*)
- Contains one or more code changes to one or more files
- You choose what and when to commit

How to make a commit

1. Make changes to code (e.g. create new file)
2. Stage changes with **`git add`**
3. Commit those changes with **`git commit`**, adding a brief description of what changes were made

When to make a commit

- **Often** - small, incremental changes are ideal
- A logical change to the code:
  - You add a new feature/script
  - You changed one script which required changing another script
- Sharing functionality:
  - Your team mate needs to run one of your scripts
  - You want to share/debug a piece of code

How to check which files have changed

- git notices untracked files and uncommitted changes
- Examine current state of repo: `git status`
- Display exact differences: `git diff`

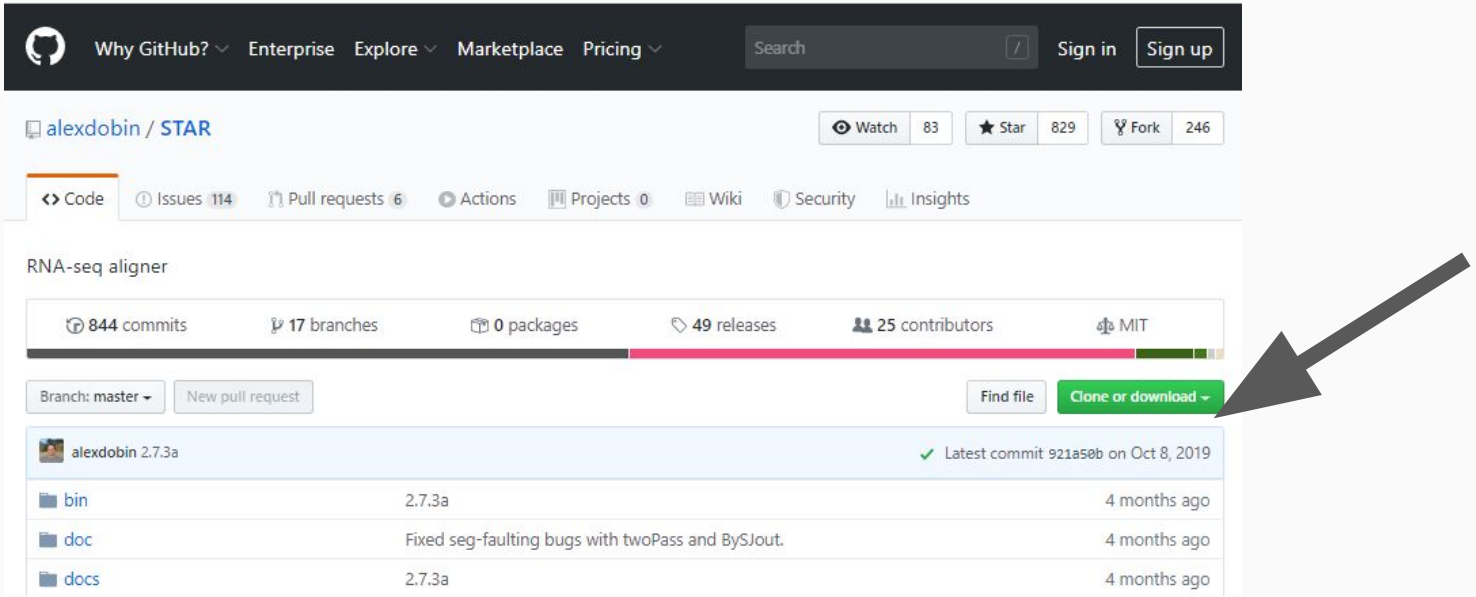
Collaborating with git

Github, Bitbucket, and GitLab

- Free, publicly available git hosting services
- All three provide essentially same functions:
  - Code hosting
  - Issue tracking
  - Access control
- Communities drive open source software development
  - Many bioinformatics tools hosted on github

Cloning projects on github

- **Clone** - make an exact copy of a repo; maintains link back to hosted repo



Creating your own projects on Github

1. Create a blank repo on Github
2. Clone your blank repo to your local computer/SCC/etc
3. Add files, make changes, local commits
4. *Push* your local changes to remote repo to sync up your changes

Integrating others' changes

- Your team pushes to the same github repo
- To get others changes: `git pull`
- git can figure out if changes to the same file can be *merged*
- If changes cannot be merged, *you have to fix it yourself*
- *Merge conflicts* can be difficult to sort out

git command cycle

1. **`git pull`**, if needed
2. **`git add`** to add new changes
3. **`git status`** to examine staged changes
4. **`git commit`** to commit those changes
5. **`git push`** to sync github with your changes
6. Repeat

git in your projects

Each project will have a github repo

- The **data curator** will manage the repo
- All code from the project should be committed to the repo
- Add a README file describing what each script does in the repo root
- We will look over your code as part of your assessment

Practices and conventions

- You may organize your repo as you like
- **Do not commit data files!**
- Each member may clone repo, or you may let the data curator manage a single directory on SCC

git workshop