git

What is git?

A version control system.

It allows you to:

- retain a history of your work
  - you can undo/ `reset`
- collaborate with others
  - you can `pull`, `push`, and `blame`
- more complex workflows
  - `checkout`

Note: we're not talking about GitHub yet, it's something different.

Getting information

- `git status` - current state
- `git log` - history
- `git branch` - location

Global setup (once per machine)

```
git config --global user.email "pkw3@duke.edu"
git config --global user.name "Patrick Wang"
git config --global init.defaultBranch main
```

Note: to set the default branch name, you need `git --version` >= 2.28.

If you want, `ssh <NETID>@biostat821.colab.duke.edu`.

Repository setup (once per project)

```
mkdir 20220110 && cd 20220110
git init
```

Note: if using GitHub, you'll probably initialize the repo there.

Idealized workflow (per "change")

```
echo "# 2022-01-10" > README.md
git add README.md
git commit -m "Add README"
```

Less ideal: undoing things

```
touch bad.txt
git add bad.txt
git commit -m "Add bad things"
```

```
git reset HEAD~1
```

More on `reset`

```
touch bad.txt
git add bad.txt
git commit -m "Add bad things"
```

`git reset ...`

- `--soft` - undoes `commit`
- `--mixed` (default) - undoes `commit` and `add`
- `--hard` - undoes `commit`, `add`, and edit!

Aside: commit references

SHA

- Each commit is identified by a 40-character SHA-1 hash.
- You can also reference it by a unique prefix (at least 4 characters).
- e.g. `git reset 979c`

Relative (usually from HEAD)

- `~X` goes back X commits (default 1)
- `^Y` does interesting things involving merges - don't use it right now
- e.g. `git reset HEAD~2`

Collaboration

- You can copy a repository from somewhere else, history and all
- Then you can sync two "forks" of a repo in various ways
  - `pull` and `push`

Collaborative repository setup (once per project)

```
git config --global pull.rebase true
git clone ssh://<NETID>@biostat821.colab.duke.edu:/s2023
cd s2023
```

Simplified workflow (per change)

```
touch <NETID>_WAS_HERE
git add <NETID>_WAS_HERE
git commit -m "Make my mark"
git pull && git push
```

`pull`

A `pull` is a `fetch` followed by a `merge` or `rebase`

-- `merge` by default, but I prefer to `rebase`

Remotes

`git remote -v`

A remote is a link to another repo - "origin" is the default name when we clone

Branching (per "feature")

change < feature < project

```
git pull
git checkout -b <NETID>_grade
vim grades.txt
...
git add grades.txt
git commit -m "Add grade for <NETID>"
git push -u origin <NETID>_grade
```

Task

Edit the grades.txt file to assign yourself (and only yourself) a grade.

Merge or rebase

`merge`: create a "merge commit"

- git generates a directed acyclic graph, but not a tree

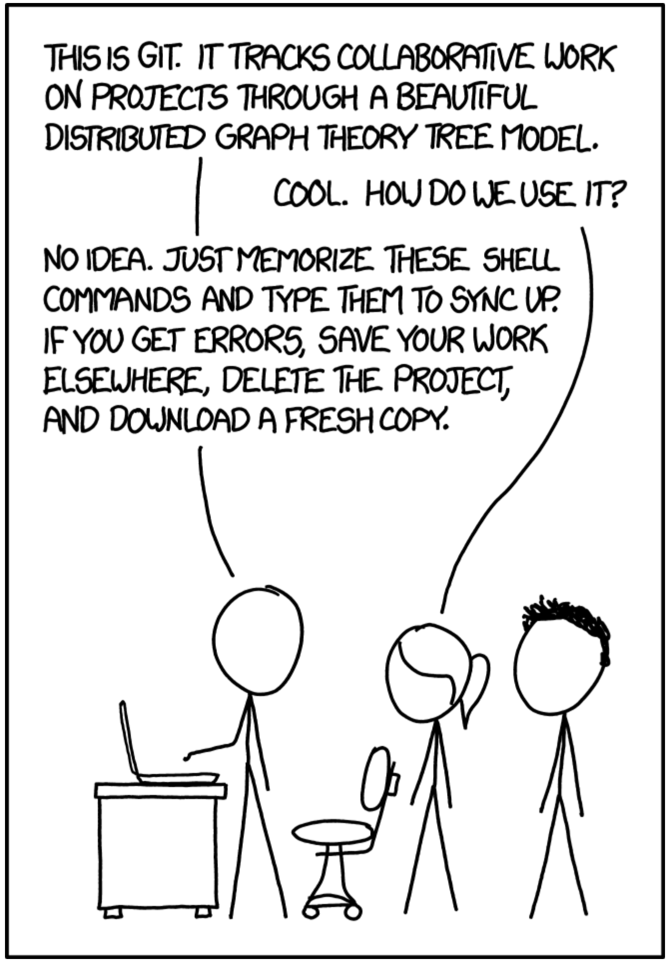
`rebase`: appends commits

- if you avoid `merge`, you get a tree!

My preference is to rebase exclusively.

Danger

- `push` cannot destroy history, unless you use `-f/--force`
  - DO NOT use `-f/--force`
- You have write access to the central repo
  - You can rewrite history or destroy the entire thing
  - How can we enforce more constraints, e.g. don't push directly to main?
  - Add a tooling layer: GitHub!
  - Bonus: we get loads of other handy development and project management tools!



https://xkcd.com/1597/