Pair Programming with Git and Github

Objectives

- Describe the paradigm (what’s the master branch used for?)
- Review how to work in branches
- Talk about the benefits of pair programming
- Describe, step-by-step, how to use Git and Github in pair programming

The git paradigm

- The master branch is for “production” code.
    - You don’t work in it!
    - You work in branches, and when your code “works” you merge it into master.
    - For the DSI daily assignments, only at the end-of-the-day do you merge the branches you worked in (individual, pair, pair_morning, pair_afternoon) into the master branch.

How to work in branches

- Start in your master branch
- Checkout a new branch
  - `$ git checkout -b <branch-name>`
- Do you work, as you do it ABC (Always Be Committing).
- Push your branch up to Github.
  - `$ git push -u <remote-name> <branch-name>`
    - if the first time!
  - `$ git push <remote-name> <branch-name>`
    - otherwise

Why pair programming?

- Fundamentally: development companies feel they get better code
- Facilitates learning and mentoring (senior <-> junior developers)
- Less likely to go down tangents
- Thought given to documentation, maintainability of code base
- Code is read more than it’s written.
    - More people familiar with the code base.

Pair work

- Define roles (A - working out of the branch in his/her repo, B - collaborator)
- A and B fork and clone repo like normal.
- A:
  - needs to add B as a collaborator for that repo on his/her Github.
  - adds a branch, e.g. `$ git checkout -b pair_morning`
  - starts coding (B navigating and helping)
  - Add, commit, push the branch to Github when it's B's turn to code.
    - e.g. `$ git push origin pair_morning`
- B:
  - After A adds B as collaborator, adds A's repo as a remote:
    - `$ git remote add <partner-name> <partner-remote-url>`
  - Help A!
  - When B's turn to code comes:
    - `$ git fetch <partner-name>`
    - `$ git checkout --track <partner-name>/<branch-name>`
    - starts coding (A navigating and helping)
    - When A's turn comes again:
      - `$ git push <partner-name> <branch-name>`
        - `e.g. $ git push <partner-name> pair_morning`

Continue switching back and forth

- A:
  - `$ git checkout <branch-name>`
    - `e.g. git checkout pair_morning`
  - `$ git pull <remote-name> <branch-name>`
    - `e.g. $ git pull origin pair_morning`
  - `A works on code (B collaborating).  ABC!`
  - `$ git push <remote-name> <branch-name>`
    - `e.g. $ git push origin pair_morning`
- B:
  - `$ git checkout <branch-name>`
    - `e.g. git checkout pair_morning`
  - `$ git pull <remote-name> <branch-name>`
    - `e.g. $ git pull <partner-name> pair_morning`
  - `B works on code (A collaborating).  ABC!`
  - `$ git push <remote-name> <branch-name>`
    - `e.g. $ git push <partner-name> pair_morning`

That’s it!

- Both A and B will have all the relevant code in their branches.
- see `git_daily_dsi.md` for how to merge everything into your master branch at the **end of the day** (after the afternoon pair!)