Git Workshop Outline

Companion notes to "Git Workshop: Syncing, Branching, and Merging"

- Target Audience
  - Have tried Git for a small project or individual research
  - Have used git add, commit, push/pull, but run into occasional errors that don't make sense
  - A follow-up to the beginner tutorial in https://swcarpentry.github.io/git-novice/
- Recommended setup
  - Git UI that shows history as a graph, such as GitKraken
- **Git/Version Control Goals**
  - I want to:
    - Back-up my work
    - Build off previous work
    - Collaborate with teammates on new analyses
    - Re-generate figures 3 years from now
    - Share my code so others can validate my research
- Sync changes across different computers
  - Motivation: push/pull between personal PC and lab/experimental computer
  - pair-up activity:
    - Add a new file
    - Commit
    - push it on one computer
    - pull it on the other
    - will run into "pull before you push"
      - stop and use as a learning moment: show GitHub history vs my local history: which should go first?
  - If you pulled stuff you didn't want: **git log/show/diff HEAD^** and **git checkout <SHA>** file
    - demo with real git repo
    - ~~learngitbranching **"Ramping Up"**~~
- To summarize (What's going on underneath?)
  - https://rachelcarmena.github.io/2018/12/12/how-to-teach-git.html
  - add vs commit: "staging area" (ie "index")
    - (Show this side-by-side with Git CLI / GitKraken
    - Why have a staging area offline?
      - Allows us to group changes to different files
    - Merge conflicts: trade-off you get for being able to work on the same file at the same time... possible conflicts
- Single-branch merge conflicts
  - say we're both working on the same file without knowing it. what happens?

    - **demo**
        - Edit file locally, then edit same file on GitHub directly.
        - Try to push, need to pull first
        - Try to pull, results in merge conflict. There's not an obvious way to combine your changes
        - **git mergetool**
    - **pair up (show slide)**: in that new file, both modify the same line (e.g. , then try to push/pull.
- Collaborating on code
    - Up until now, all our work is the same workspace. Anything we want to share with 1 person, we're changing for **everybody**. Now, maybe we want to share changes that are half-finished, without disrupting other people's workflows
    - **enter branching (bolded = show in branch tutorial)**
        - Ex. want to run this tutorial, but with Python audience
            - Change hello_world.m to hello_world.py
                - Or (in MATLAB repo), 2 versions. in-session task + updating version simultaneously. This is a very real-world example.
            - Options
                - Commit change (need to commit each time)
                - Create new repository (heavyweight)
                - Branch ("save as...")
        - branch = history tree branch
        - https://learngitbranching.js.org/?locale=en_US
            - ~~lesson 1 for intro to environment~~
            - lessons 2+3 for real branching, then
                - Lesson 3 objective
    - **pair up: branching in a real git repo**
        - call on people to make it interactive
        - Show it again in GitKraken to make it real
    - **merging -** join the changes from 2 branches together
    - cherry-pick
        - learngitbranching **"moving work around" 1**
        - to see change: git log
        - **pair up: see slides**
    - "branch early and often"
    - A quick walk through GitHub
        - History Tree
            - Insights > Network.
            - In GitKraken, adjust the column size to see the tree
        - Pull Request
            - https://github.com/charlesincharge/tutorial_git/pulls
            - PR = "I request that you pull my changes into your branch"
            - good way of reviewing code changes

                - common practice in software orgs - all changes go through pull requests
- Step back and ask: why bother with branches/GitHub?
    - **Show workflow slides**
    - single branch unsustainable and untrack-able
        - single-branch = backup strategy (like Dropbox, not a collaboration strategy)
        - Difficult to collaborate without trampling other people's work
        - Need explicit collaboration strategies
    - branches allow multiple possible backups (ie "save as")
        - like virtual directories. analysis-20190426, analysis-20190321
    - Only way for groups to collaborate on large code projects (which we're doing)
- Tips and Tricks
    - **stash**
        - Working on something partway
        - git stash {pop/list/drop/show -v}
    - tag
        - Good code tagging helps for: re-creating figures 2 years from now
        - Basically like a "commit"
        - Need to version-control both data and different repositories!
    - bisect
        - Finding the commit that broke your code
    - blame
        - Help find rationale behind change - see what other changes were associated
- What's the diff? (Git commands that seem similar but are different)
    - merge/rebase learngitbranching intro4, fetch/pull : learngitbranching remote7
    - reset/revert learngitbranching rampup4
    - branch/tag/commit, equivalent

Resources:
- Atlassian Git Tutorials
- GitHub Git Tutorials
- https://learngitbranching.js.org/
- https://betterexplained.com/articles/aha-moments-when-learning-git/
- https://buddy.works/blog/5-types-of-git-workflows
- https://rachelcarmena.github.io/2018/12/12/how-to-teach-git.html