Code Collaboration with Git

Lecture 12

Outline

- GitHub Review
- GitHub Workflows:
  - Trunk Based
  - Branching Based
- Specifics for Data Science Teams

Great tutorials everywhere.

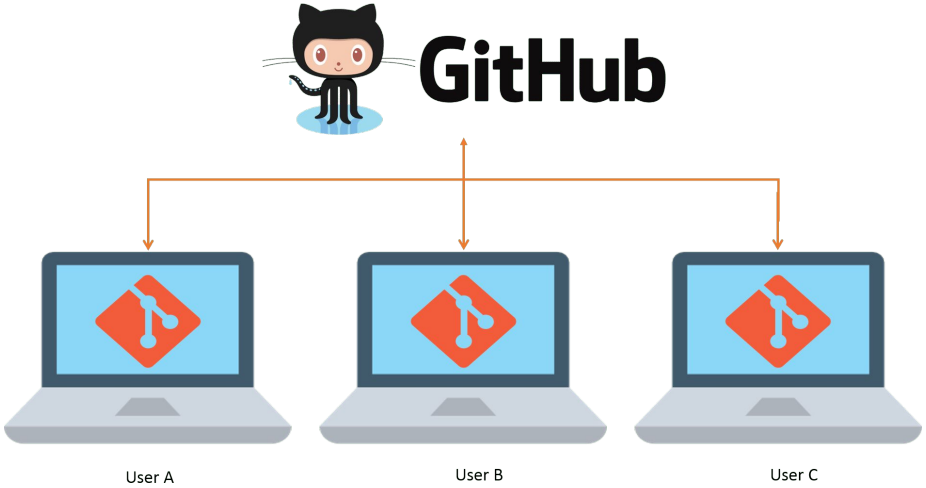
E.g. https://www.atlassian.com/git/tutorials/syncing

GitHub Review

- We will review some *basics*.
- You *will* mess up version control working with others on your project; knowledge of the basics will help fix it!
- Using Git properly will:
  - Version and backup your work (no need to save multiple versions)
  - Makes reverting to a previous version easy! (if used correctly!)
  - Allow collaboration without worry of ruining others work.

Git vs GitHub

- Git
  - Distributed Version Control
  - Code history kept on local computer.
- GitHub
  - Remote Server that allows syncing of distributed git-versioned projects.
  - Only one git-based remote platform. Others: Bitbucket, GitLab



Snapshots: the building block of Git

- Git keeps track of your code history through snapshots
- Snapshots records what you files look like at a give time.
- You decide when to take a snapshot (“save” or “commit”).
- You can revisit old snapshots (“checkout” or “revert”).
- Git only stores *differences between* the snapshots!
    - Makes the version history much smaller
    - Beware of changing, programmatically generating content on Git (e.g. notebook output).

Commits: saving the state of your project

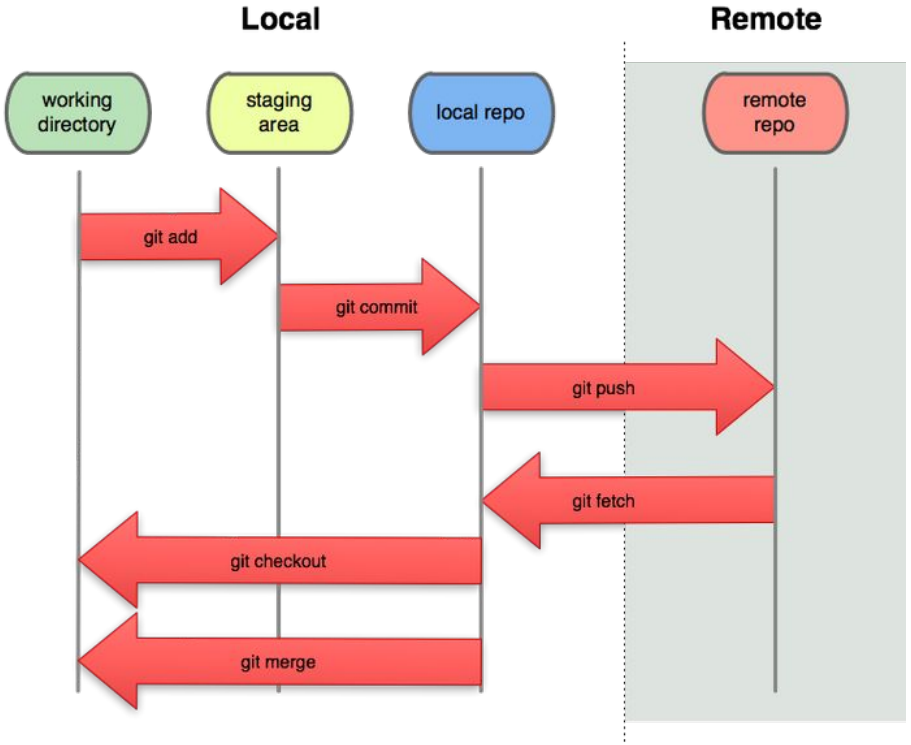
- The act of creating a snapshot.
- Roughly translates to “saving a file” in git
- A project is made up of commits.
- A commit consists of:
  - Information of how the files changed from previous commit
  - A reference to the previous commit
  - A unique identifier (hash code)
- Command: `git commit -m “commit message”`

The anatomy of a repository

- A repository is a collection of files and their history.
- A repository is a collection of commits.
- Can live on a local machine or a remote server (e.g. github)
- The totality of a Repository is in a hidden directory called `.git` in your project's base directory (created with `git init`)
    - This directory contains an INDEX of all files being tracked by Git
    - Commit hashes and diffs
    - The remote address of a repository on GitHub (called `origin`)

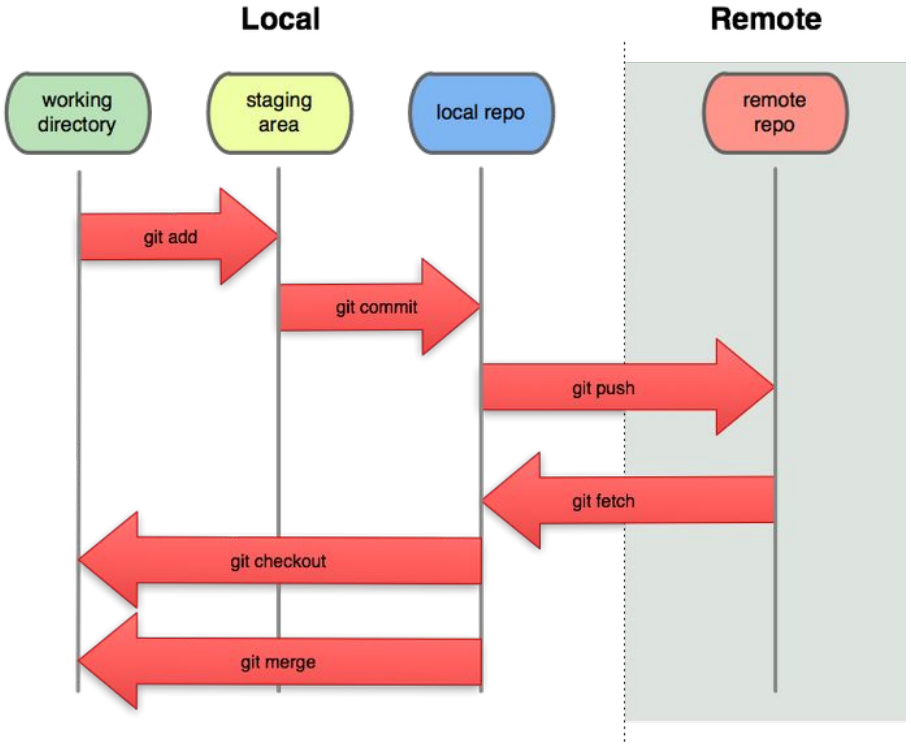
Working vs Staging vs Snapshot

- The **working directory** are the files that you are currently working on.
- The **staging area** are files that are being tracked by git, and ready to commit. You must add a file to staging to commit the file.
- A **snapshot** is the state of the staging area when you commit your work.



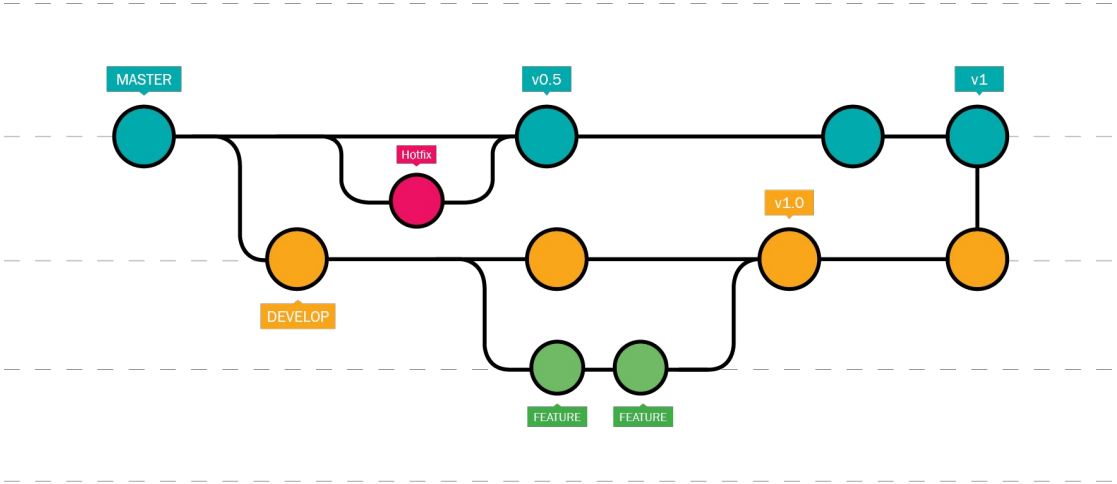
Repositories: local vs remote

- The process of downloading commits that don't exist on your machine from a remote repository is called **fetching** changes.
- Related to fetch is **pull**.
- The process of adding your local changes to a remote repository is called **pushing** changes.
    - `git push origin main`



Git Workflows

- Code development changes can *branch* off from each other.
- In figure:
  - Each color is a different branch
  - Each node is a commit
  - Work on the code moves from left-to-right.

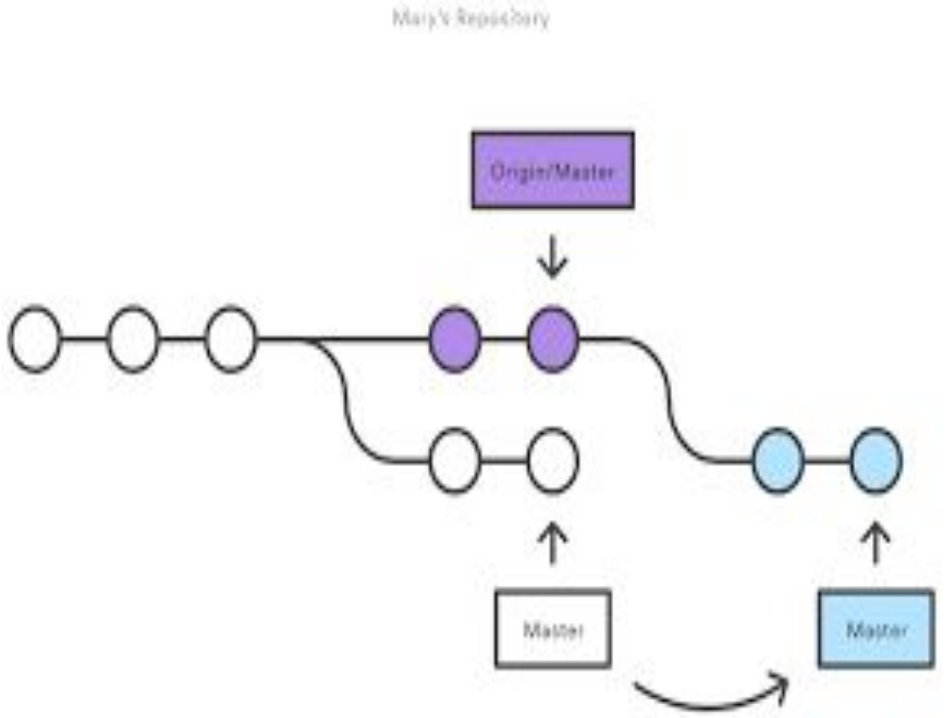


Centralized Git Workflow

- Always work linearly, from on a single ‘branch’ of code.
  - This default “main” branch is called the *main* branch.
  - You will *not* be using this basic workflow.
- Typical Workflow:
  - `git pull my_repo  # pull the latest version from remote (or clone new)`
  - `git add mainfile.py  # add changed file to staging`
  - `git commit -m “add new function” # commit the staged changes`
  - `… repeat the last two steps as you make more changes...`
  - `git push origin main # push the changes to remote`

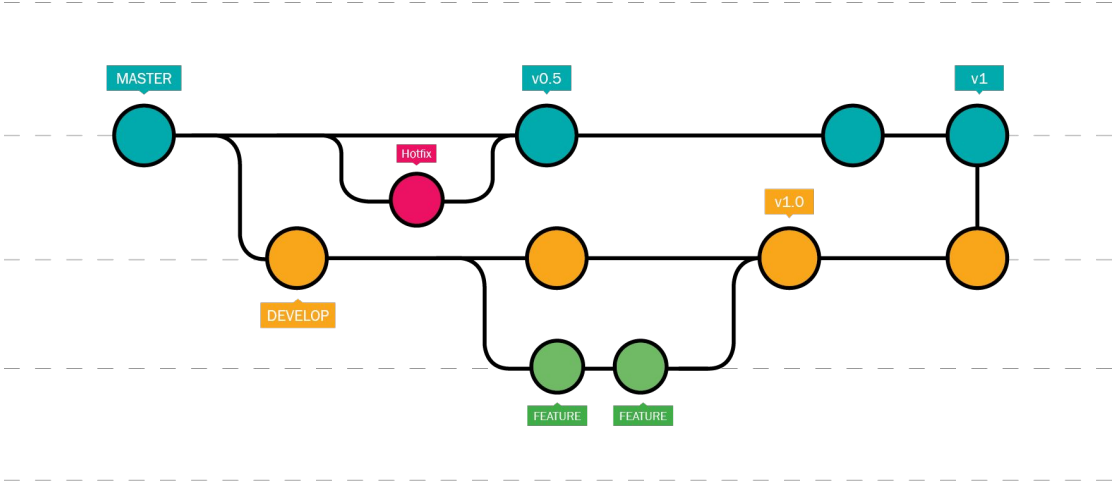
Centralized workflow: multiple developers

- What happens when you push your commit to GitHub *after* other code already been committed by another dev?
- Causes push to be rejected!
- Need to `git rebase` to “mix” the two commit histories.
- Messy when working with many commits!



Branching-Based Git Workflows

- Branches help you keep different work separate according to function.
- Multiple people working on the same code should always use different different branches!
- One person working on multiple tasks in a single repository should use different branches for each task.



Pattern for Branching Based Workflow

Suppose I want to add a new data source to my project, which has a working version in my main branch.

1. `git checkout -b new_data  # create/checkout new data`
2. `...do my work in new_data branch…`
3. `git add mynewfile`
4. `git commit -m “all my new work on the new data source”`
5. `git push origin new_data # push new branch to remote.`

Incorporating a branch into main

When the branch is “finished” it must be incorporated into main:

1. Pull newest version of main
2. Checkout branch
3. Merge main *into* the branch (to make your branch work w/newest main)
4. Push branch to remote
5. Initiate Pull Request from GitHub website
6. Have collaborators review changes and merge pull request from website
7. Locally, checkout main and pull the newest version (i.e. the work on your branch!)

Incorporating a branch into main

When the branch is “finished” it must be incorporated into main:

1. `git pull origin main`
2. `git checkout new_data`
3. `git merge main # from within new_data!`
4. `git push origin new_data`
5. Initiate Pull Request from GitHub website
6. Have collaborators review changes and merge pull request from website
7. `git checkout main; git pull origin main`

Merge Conflicts

What happens when changes were made to the same lines on both:

- Your development branch, and
- Subsequent work on main?

Merge conflict!

- Choose the version to keep...
- ... and move on.

index.html styles.css

```
<!DOCTYPE html>
<html>
<head>
  <title>GC Merge Demo</title>
  <link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
<<<<<<< HEAD
  <h1>Grand Circus Merge Demo</h1>
=======
  <h1 class="header">Merge Demo</h1>
>>>>>>> 1d46372af5a97f8ef05b9eecb82712382cc5f31c
  <p>
    Demo the merge.
  </p>
  <p class="footer">
    Grand Circus Detroit
  </p>
</body>
</html>
```

When to create a new branch?

- Development branches are for standard “development work”
  - A bit lazy!
- Better: name the branch after the task or feature you are working on.
- Different people should *always* work on different.
- Example Branch Names: `AF_dev, AF_etl, AF_newdata`
  - Putting initials first makes branches more easily searchable.

Version Control and Notebooks

- Two kinds of notebooks: “code development” and “code driven reports”.
  - “Code development notebooks” should be personal and not collaborated upon.
  - Create library code from dev notebooks if collaborating (code reuse => library code).
  - Code driven reports are versioned and worked on simultaneously; always clear output before committing.
  - Reports should *always* be quick to run (if not, create intermediate data files that make it quick).
- Notebooks version *very poorly*
  - Fixing merge conflicts is difficult in JSON
  - Git is meant to version *code*, but notebook output is *data* (and changes a lot!)
  - Always try to clear-notebook-output before committing (to commit only code)
  - If you want to commit output (e.g. a report), copy/export to a static document (e.g. html/pdf) in docs directory.

Version Control in 180B

- Appoint a team lead to create a project repository.
  - It should be *public* and have a *descriptive* name (*not* DSC180B-Project)
- Team lead should add group members as collaborators
- Team members should clone repository; checkout their own branches.
- Before Weekly Check-in, members should merge latest work into main.
  - Branches should be small in scope and correspond to weekly tasks
  - The longer you wait to merge a branch into main, the harder it becomes.
  - Reviewing Pull requests for merging is a good way to ‘get up to speed’ on everyone else’s work before the weekly check-in!

Helpful Tips

- Need to revert changes back to a specific commit?
  - https://stackoverflow.com/questions/4114095/how-do-i-revert-a-git-repository-to-a-previous-commit
- Did you add a file that you shouldn't have (e.g. a large data file) and need to remove it from the repository?
  - https://stackoverflow.com/questions/2047465/how-can-i-delete-a-file-from-a-git-repository
  - https://help.github.com/en/github/managing-large-files/removing-files-from-a-repositorys-history

There are a ridiculous number of helpful StackOverflow posts on Git (just look at the upvotes...). However, try to understand *what the answers are doing*, else you may lose unintended work!

- Always be careful before *merging changes into main* and *pushing changes to remote* -- with that, you can most always quickly fix your unintended work.