Git Overview

Git Overview

What Is Version Control?

- management system that manages changes that you make in your documents, project, computer programs, large websites, etc till end
    - i.e: adding new files, modifying old ones by changing source code
    - creates snapshot of entire project every time you make a change in project
        - snapshots/changes: "versions" - entire state of project at particular time
            - what kind of files your project is storing/what kind of changes

Why Version Control?

1. collaboration
    A. avoids conflict in isolated modification between multiple developers
        a. w/ VCS - provides shared workspace & continuously shares details about changes
2. storing versions
    A. essential to save version of project post-changes is essential; w/o VCS can be confusing
        a. how much to save? entire project or only changes?
            1. if only changes - hard to view whole project
            2. if entire project - huge amount of redundant data
        b. how to name versions?
3. backup
    A. if central server crashes, backup is available in local servers
4. analyze
    A. VCS provides info regarding proper description of every changes (what, when, where)

Distributed VCS

- repository: data space where you store all files related to your project
- central & local repos
    - first developers make changes in local before pushing to central
    - update local repo w/ new files that're pushed into central repo by operation called pull

GitHub

- Git: VC management tool allowing you to operations like fetching data and push local files into central server
- GitHub: code-hosting platform for VC collab
    - allows you to host central repo in a remote server
    - "social network"

What is Git?

- distributed VC tool that supports distributed non-linear workflow
    - tool required to make all version control tasks i.e: local repo, help access remote repo to push/pull

Repository

- directory or storage space where your projects are stored
- can be local to folder on your computer or can be storage space on Github-like online host
- can keep code files, text files, image files, etc

1. central repo
    A. typically located on remote server (Github)
    B. exclusive consists of .git repo folder
    C. meant for team to share & exchange
2. local repo
    A. typical located on local machine
    B. resides as a .git folder inside your project's root
    C. only admin of machine can work w/ this repo

<u>Git Operations & Commands</u>

1) Git
- Git: distributed version control system w/ purpose to manage a project, or set of files, as they change over time
    - VCS: storage system that records changes to a file or set of files over time so you can reference specificities
- **git init**: (**command**) initialize
    - sets up all tools needed by Git to commence tracking project's changes
    - creates local repo
    - don't need to initialize if cloned repo

2) Git Workflow
- Git workflow consists of editing files in working directory, adding files to the staging area, and saving changes to a Git repo:
    A. **working directory**
        a. where all the work is done
        b. i.e: creating, editing, deleting, organizing files
    B. **staging area**
        a. where you'll list changes you make to the working directory
    C. **repository**
        a. where Git permanently stores those changes as different versions of the project
- **git status**: (**command**) checks status of changes; inspects contents of working directory & staging area
    - result - "untracked files" bc Git sees file but hasn't started tracking
- in order to commence tracking, files need to be moved to staging area
    - **git add** [**filename(s)**]: (**command**) add files from w.d. to staging area
    - **git add '*.txt'**: (**command**) wildcard that adds multiple files ending in .txt at once
    - git rm '*.txt': removes
- **git diff** [**filename**]: (**command**) checks difference between working directory and staging area
    - can use HEAD as [filename] to check most recent commit
    - options:
        - --staged: see changes you just staged
- **git commit**: last step (**command**) permanently saves changes from staging area -> repo
    - commit messages must:
        - "be in quotation marks"
        - be written in the present tense

        - be brief (< 50 characters) when using -**m option**
- head commit: the commit you're currently on
    - **git show HEAD**
        - output of this displays everything the **git log** command displays + all file changes that were committed
    - **git checkout HEAD** [**filename**]: discards changes in w.d. & restores to o.g.
        - restores file in w.d. to revert to before last commit
    - **git reset HEAD** [**filename**]: "unstages" unwanted file(s) from staging area
    - **git reset** [**commit_SHA**]: resets to previous commit in commit history
        - use **git log** to find SHA
        - use first 7 letters for [commit_SHA]
        - HEAD is now set to that prev commift
- **git log**: shows list for viewing all previous commits
    - option:
        - git log **--summary**
    - results:
        - SHA: 40 character unique commit code
        - commit author
        - date & time of commit
        - commit message

8) Git Branching: separate line of commits from master
- **git branch**: shows current branch (i.e: *master, or more)
- **git branch** [**new_branch_name**]: creates new branch
- **git checkout** [**branch_name**]: switches to branch
- **git merge** [**giver_branch_name**]: merges changes to master (receiver) branch
    - switch (git checkout) to master first
- **git branch** -**d** [**branch_name**]: deletes branch

9) Git Teamwork
- push local repos to Github distributive server
- **git remote add** [**origin/remote_name**] [**repo URL**]: creates a remote repo on Github central server
    - [remote_name] = usually 'origin' bc it's main one
- **git clone** [**remote_location**] [**clone_name**]: clones replica of central repo as a local copy
    - clones Github central repo from [remote_location] into local directory called [clone_name]
        - [remote_location]: tells Git where to find remote (website or filepath); usually the origin
        - [clone_name]: name given to local directory
- **git remote -v**: lists a Git project's remotes
    - *cd* into appropriate local directory first
    - results: fetch/push
- **git fetch**: fetches any new changes made to remote origin
    - *cd* into local directory first
    - doesn't merge change to local repo
- **git push** [**origin/remote_name**] [**local_branch_name**]: pushes branch up to the remote
    - option: adding option between after *push*
        - **-u**: tells Git to remember parameters (next time can simply run *git push*)
- **git pull** [**origin/remote_name**] [**local_branch_name**]: checks for changes others pushed to Github

    - git stash: stashes changes that you don't want to commit to just yet
    - git stash apply: re-applies changes after your pull
- **git stash**: sometimes when you go to pull, you may have changes you don't want to commit just yet..
    - one option (other than committing) is to stash the changes
    - use git stash command to stash changes and **git stash apply** to re-apply your changes after your pull