Review

- What were the terms we learned on Monday?
  - How do they fit together?

Why Version Control Systems?

Why Version Control Systems?

- Collaborate on code with a team
- Roll back/restore older version of code
  - Granularity: Individual files or collection of files
- Store ownership of files/changes and when occurred
- Record reasons for changes
- Track progress
- Each developer has own sandbox of code
- Maintaining multiple branches

Centralized vs Distributed VCS

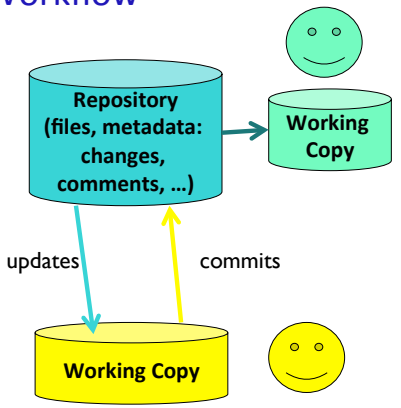
- What are their characteristics?
- What are examples of each?

Centralized vs Distributed VCS

- Centralized (CVS, Subversion)
  - One central repository: the gold standard
  - All updates made against central repo
  - No access to repo? No updates
  - Must sync with central repo before adding updates
- Decentralized (git, mercurial, bazaar)
  - Multiple copies/clones/forks of repositories
  - You can always have a local repo
  - You can optionally have a central repo
  - More distributed sharing options

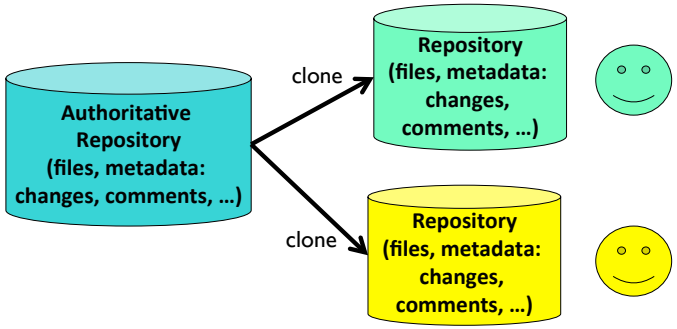
Centralized VCS Workflow

1. Pull changes other people made
2. Make your changes, and make sure they work properly
3. Commit your changes to the central server



Distributed VCS Workflow

1. Clone copies of repository



Distributed VCS Workflow

1. Clone copies of repository
2. Work (mostly) locally
   - Optionally push to remote but not necessarily an authoritative repository

Discussion

- CVCS: Mostly remote operations
  - Requires network connectivity for updates, commits
    - More expensive operations
  - Less space for each client
- DVCS: Mostly local operations (faster)
  - Does not require network connectivity
  - Whole copy of the repository
  - More space for each "client"

What Should Be Under Version Control?

What Should Be Under Version Control?

- Yes:
  - Text-based things made by humans
  - Source code
  - Scripts
  - Files that aren't going to change
- No:
  - Automatically built things
    - (executables, object files, jar files)
  - Temporary files
  - Sensitive data: passwords, private ssh keys
  - Most VCSs have ways to Ignore these

https://git-scm.com/book/en/v2

Configuring git

Configure git to identify your code modifications as belonging to you

- Check the configuration
  - `git config -l`
- if no `user.name`:
  - `git config --global user.name "My Name"`
  - `git config --global user.email "me@place.com"`
- `git config --global color.ui auto`

Using git: initializing a repository

- Repository: Project
- Go into your cs397 directory
- Execute `git init git_repo`
  - Creates a new directory named `git_repo` and initializes a repository
  - Creates a new subdirectory named `.git` that contains all of your repository files
  - View the contents of the `.git` directory

Using git: checking the status

- `git status`
  - show modified files in working directory, staged for next commit

git File States

- **Committed**: the data is safely stored in your local repository
- **Modified**: changed a file but have not committed it to repository yet
- **Staged**: marked a modified file in its current version to go into your next commit snapshot.

git Project Sections

- **.git directory**: stores your project's metadata and object database
  - Copied when you clone a repository from another computer.
- **Working directory**: *single* checkout of *one* version of the project
  - Files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
- **Staging area**: stores information about what will go into your next commit

Using git: updating a repository

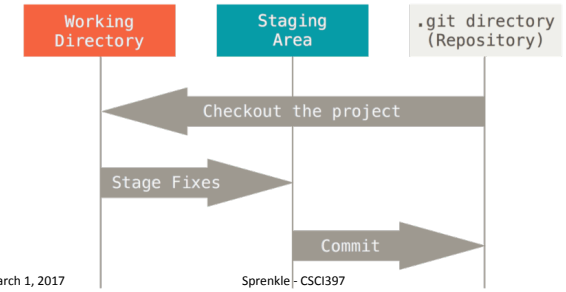
- Create a file
  - `touch myscript.sh`
- Add that script to git for tracking
  - `git add myscript.sh`
- Commit the files
  - `git commit -m 'initial version'`

Using git: updating a repository

- Edit `myscript.sh`
- Run `git status`
- Try to commit
  - `git commit`

git Work flow

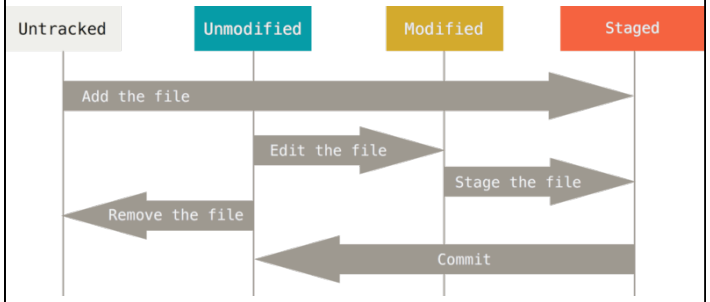
1. Modify files in your working directory
2. Stage the files, adding snapshots to your staging area.
3. Commit: takes the files in the staging area and stores that snapshot permanently to `.git` directory



File Stages

- Every file is either ***tracked*** or ***untracked***
- **Tracked**: files in the last snapshot
  - can be unmodified, modified, or staged
- **Untracked**: everything else
  - Any files in your working directory that were not in your last snapshot and are not in your staging area
- When you first clone a repository, all of your files will be *tracked* and *unmodified*
  - Git just checked them out and you haven't edited them

File Stages



- Source: `https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository`

Using git: updating a repository

- Create a new file
  - `touch otherscript.sh`
- Run `git status`

git: committing our changes

- We're happy with our files (one new, one modified) and we want to commit them
- What should we do?

Git: committing our changes

- `git add *.sh`
- `git commit –m "New version!"`

What's been happening?

- `git log`

Branches

- We've been working in the master
- Instead, let's use branches so that we can work on multiple features in parallel
- Create a new branch and switch to it
  - `git checkout –b mybranch`
  - `git status`

Update code in the branch

- Add file `newscript.sh`
- Stage and commit the file:
  - `git add newscript.sh`
  - `git commit –m "added newscript.sh"`
- `git status`
- `git branch`
  - list your branches
  - a `*` will appear next to the currently active branch

Merge our branch into the master

- `git checkout master`
  - Go back into the master branch
  - merge the specified branch's history into the current one
- `git merge mybranch`
  - Get the code from `mybranch` and merge it into here

One More Time, with Feeling

- Create a new branch and switch to that branch
  - `git checkout -b newfeature`
- Edit newscript.sh
  - Add something to it
- Stage and commit the file in **one** step:
  - `git commit -a –m "updated …"`

Go back to the master

- Go back to the master branch
  - `git checkout master`
- Edit newscript.sh
  - Add something (different) to it
- Stage and commit the file in **one** step:
  - `git commit -a –m "updated …"`
- Attempt to merge the `newfeature` branch into the `master` branch
  - `git merge newfeature`

GIT COLLABORATION

Working on a Shared Project

- I created a repository for us to use
  - `git clone /csdept/courses/cs397/shared/gitrepo/ mycopy`
- Go into your copy
- View the contents
- Transmit local branch commits to the remote repository branch
  - `git push -u origin master`

Explore git

- What other git commands are there?
  - What do they do?