Git

Trust me, it’s awesome

Welcome!

- 2 parts to this workshop: this presentation and exercises / general questions afterwards.
- Best cheatsheet out there: https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf
- Anything you see in < > should be replaced with what it refers to

The most useful command

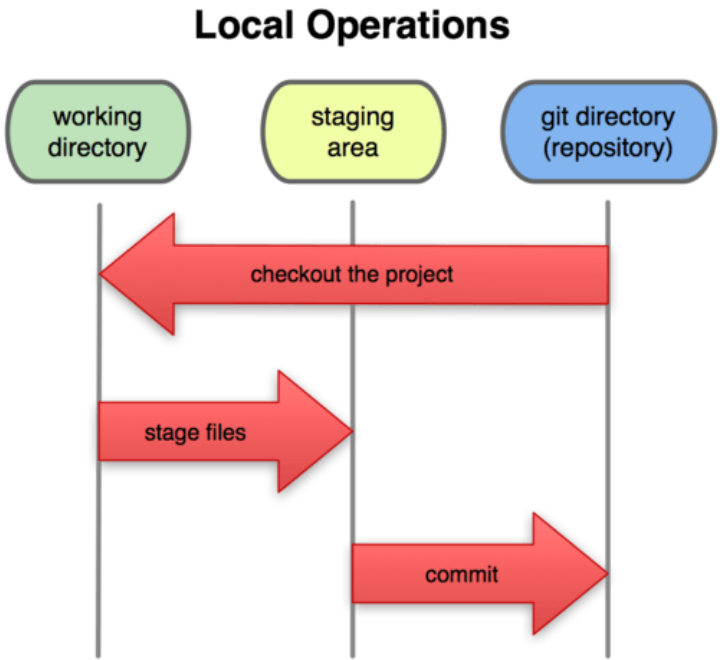
- man git-<command> - opens up the man page for <command>, giving you detailed information about everything you can do with it.
- If you ever have a question about git, this has your answer.

Git is NOT Github

- Git = Distributed Revision Control Software, used to keep track of changes within a project by taking snapshots of files over time
- Github = Online service that allows you to upload and share your git repositories.
- **You can use Git without Github!**

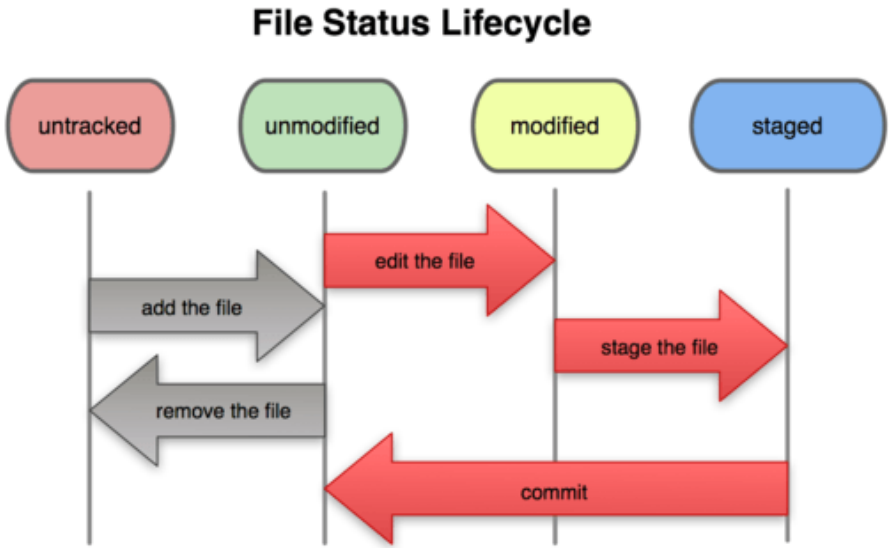
The Basics

- 3 areas: working directory, **staging area**, and the **git repository**.
- Can think of a remote repository as an optional 4th that's a clone of your local git repo
- Working directory is just your local file system's version of the project



The Staging Area

- Consists of files that are tracked by git but that have NOT been saved into the repository
- When you commit, everything in the staging area gets saved into the git repo

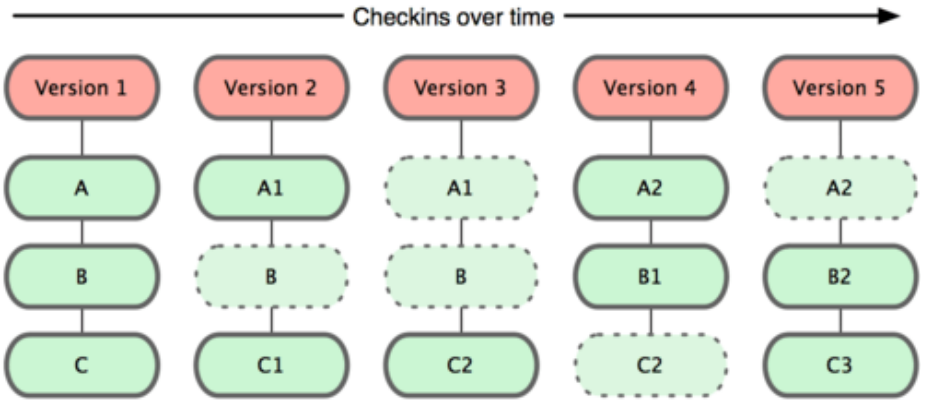


Staging Area Operations

- git add <file> = adds <file> to the staging area, ready to be committed
- git status = shows all files in the staging area and files that aren't being tracked
- There are different ways to remove a file from the staging area based on its state. Running git status will give you the commands to remove a file from the staging area

The Git Repository

- A miniature file system consisting of snapshots of your project over time
- Holds all of your committed files and their changes over time
- You can revisit ANY commit that's in the git repository

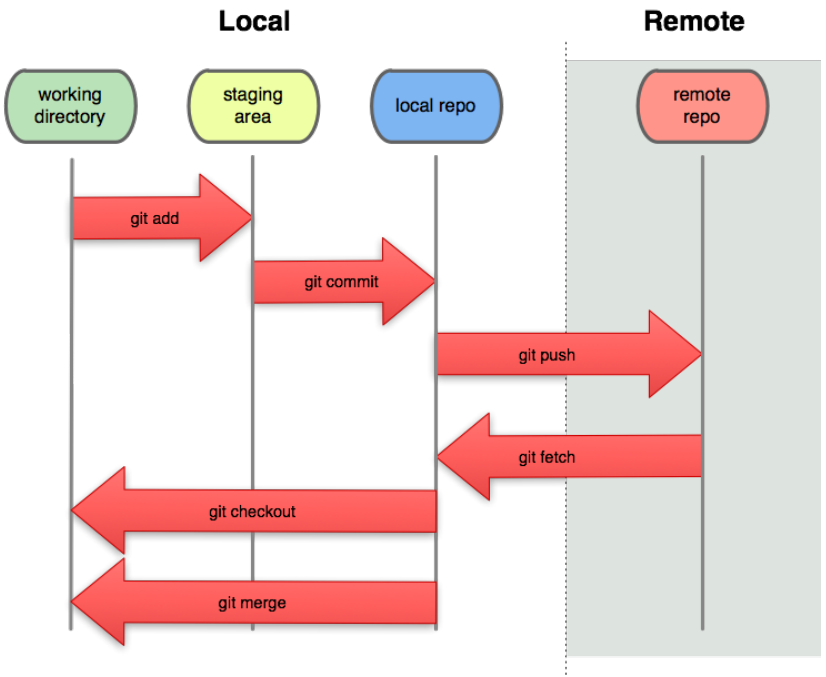


Git Repository Operations

- git commit -m <message> = commits all files in the staging area to the git repository. Write a sensible message!
- git log = lists all of the commits in the repository in chronological order
  - --oneline = compresses the information to one line per commit
- git reset --hard <commit-id> = reverts back to the commit referred to by <commit-id>.
  - use HEAD~n instead of <commit-id> to go back the nth most recent commit (ex. HEAD~2 goes back two commits)

Remote Repositories

- Remote repositories are git repositories hosted on a server run by you or a service like Github.
- Used to share code bases among developers



Working with Remote Repositories

- git remote add <name> <url> - sets up a remote tracking branch in your repository (AKA tells git about your github repo)
- git push <remote> <branch> - updates <branch> that is located in the repository pointed to by <remote>
  - Most common use: git push origin master, which pushes your code changes to the master branch in the origin remote repo
- git pull <remote> <branch> - fetches and merges changes from <branch> located in <remote>

Other Useful Commands

- git init = Creates a new git repository
- git clone <git-url> = Creates a local repo that is identical to the repository located at <git-url>
- git stash = Caches all changes to files tracked by git and removes these changes from the working directory
  - Use git stash pop to re-apply these changes. This is really useful when you accidently make changes on one branch that are meant for another

Rules of Thumb to Avoid Problems

- **Do not make a git repository inside another git repository.**
- Don’t edit files directly on Github.
- If you mess things up, its often easiest to just clone your remote repo again and start fresh.
- Always write sensible commit messages.
- Commit often, but not too often. You’ll want to go back if you break something, but too many commits makes your git logs a mess to read!

Demo Time

Branching - The what and the why

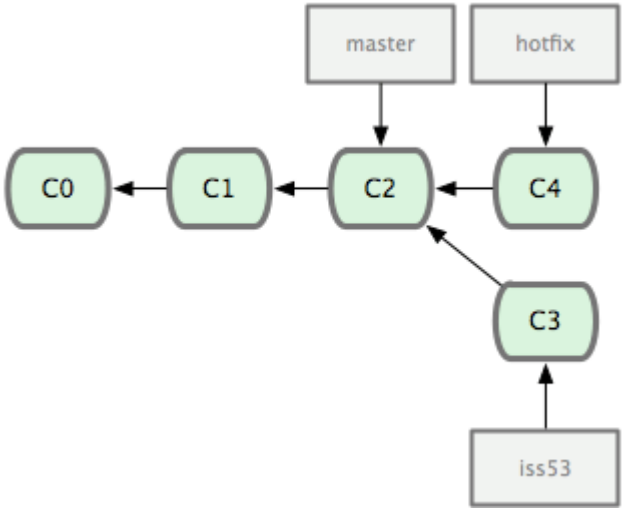
- Branching is a way to separate your work logically by partitioning off areas of your work from the master branch
- A change made on one branch does NOT show up on another branch until the branches have been merged

Branching - How to use it

- git branch <name> - creates a new branch. Omit <name> and use git branch to list all branches
  - -r = list all remote branches
  - -d <name> = delete a branch
- git checkout <branch> - makes <branch> your current branch
- git merge <name> - merges branch <name> into your current branch. Will create a new commit merging the two branches together if necessary.

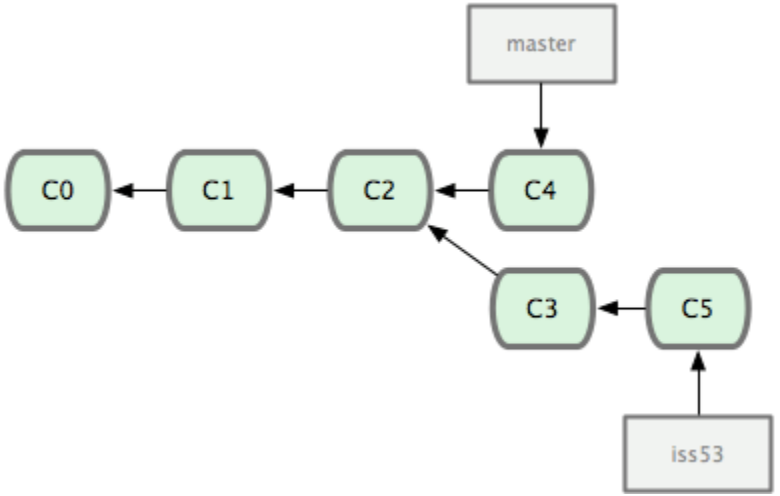
Merging - Example Part I

- 2 branches, hotfix and iss53
- Both branches diverged from master at c2
- hotfix has fixed some critical bug in server.rb
- iss53 is dedicated to improving server.rb’s response time



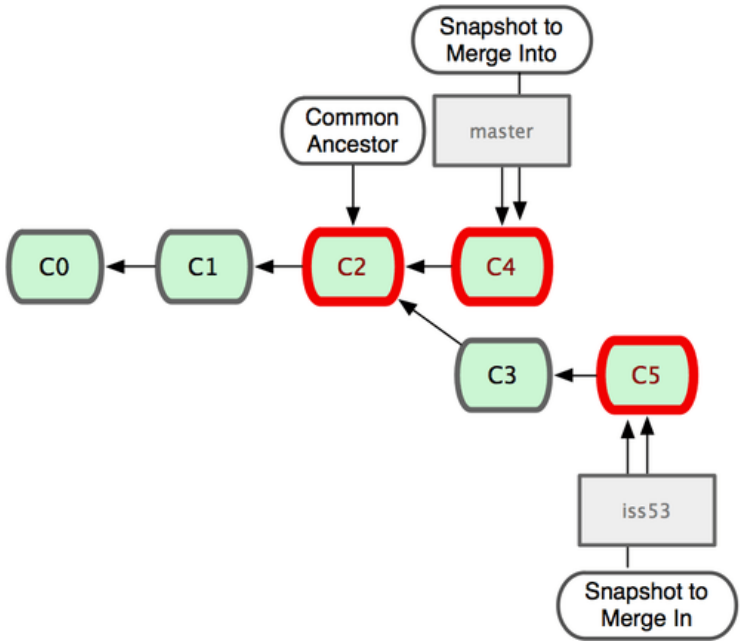
Merging - Example Part II

- Merging hotfix is simple
- Run ‘git checkout master’ and then ‘git merge hotfix’
- Run ‘git branch -d hotfix’ to remove hotfix as it isn’t needed anymore



Merging - Example Part III

- Merging iss53 is tougher
- There are 2 different versions of server.rb, the one on master (updated from hotfix) and the one on iss53
- If there are conflicts, Git will alert you and pause the merge



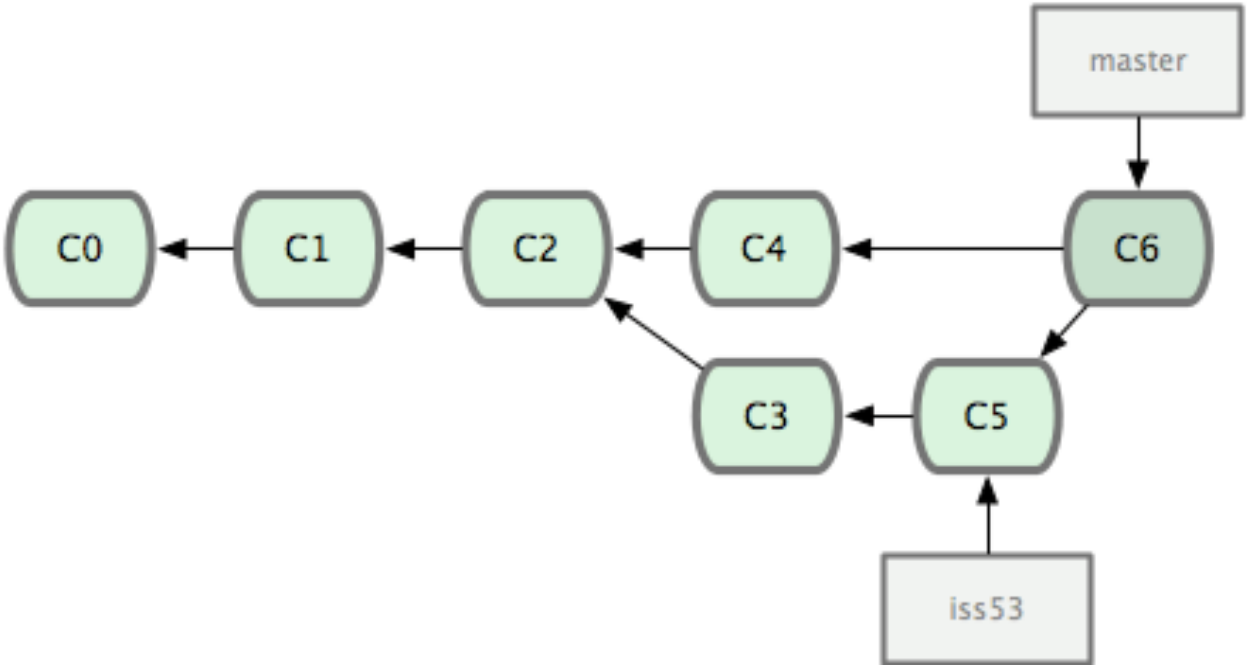
Merging - Example Part IV

- Git will mark merge conflicts in files that have problems
- You must update, readd, and recommit these files before the merge is complete

Merge Conflict

<<<<<<< HEAD
Snapshot to merge into's version
=======
Snapshot to merge in's version
>>>>>>> iss53

Merging - Example Part V



More Demos!

More on Merge Conflicts

- If you’re working on a team, someone might push stuff out before you do. Pulling their work can result in merge conflicts
- These are resolved the same way as branching merge conflicts

Exercise Time!

- Go to https://github.com/taylorc93/git-exercises and fork this repo (don't clone, you won't be able to push) if you want some practice
- To fork the repo, click the fork button located in the upper right corner and follow Github's instructions

Citations

1. http://git-scm.com/book
2. http://en.wikipedia.org/wiki/Git_(software)
3. Git man pages