CS104 Appendix A

Github

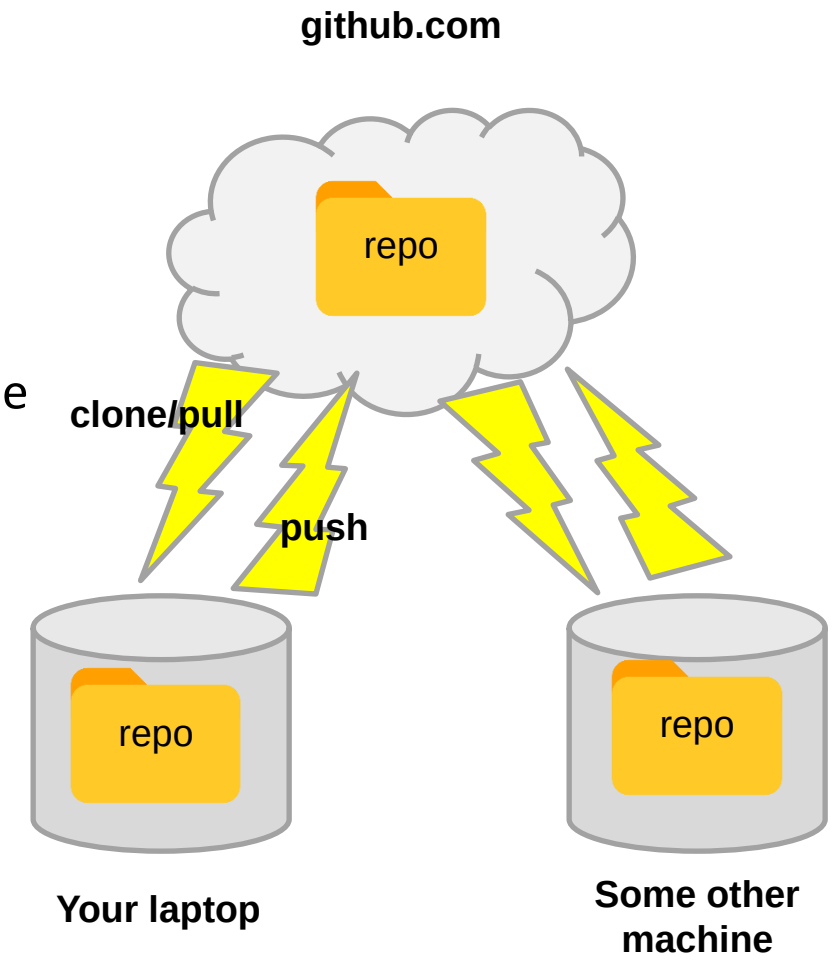
GIT AND GITHUB

Source/Version Control

- Have you ever made backups of backups of source files to save your code at various states of development (so you can recover to an earlier working version)?
- Have you ever worked on the same code with a partner and tried to integrate changes they made?
- These tasks can be painful without help
- Source/version control tools make this task easy
  - Allows one codebase (no separate folders or copies of files) that can be "checkpointed" (committed) at various times and then return back to a previous checkpoint/commit if desired
  - Can help merge differences between two versions of the same code
- Common source/version control tools are:
  - Git, Subversion, and a few older ones (cvs, rcs, clearcase, etc.)

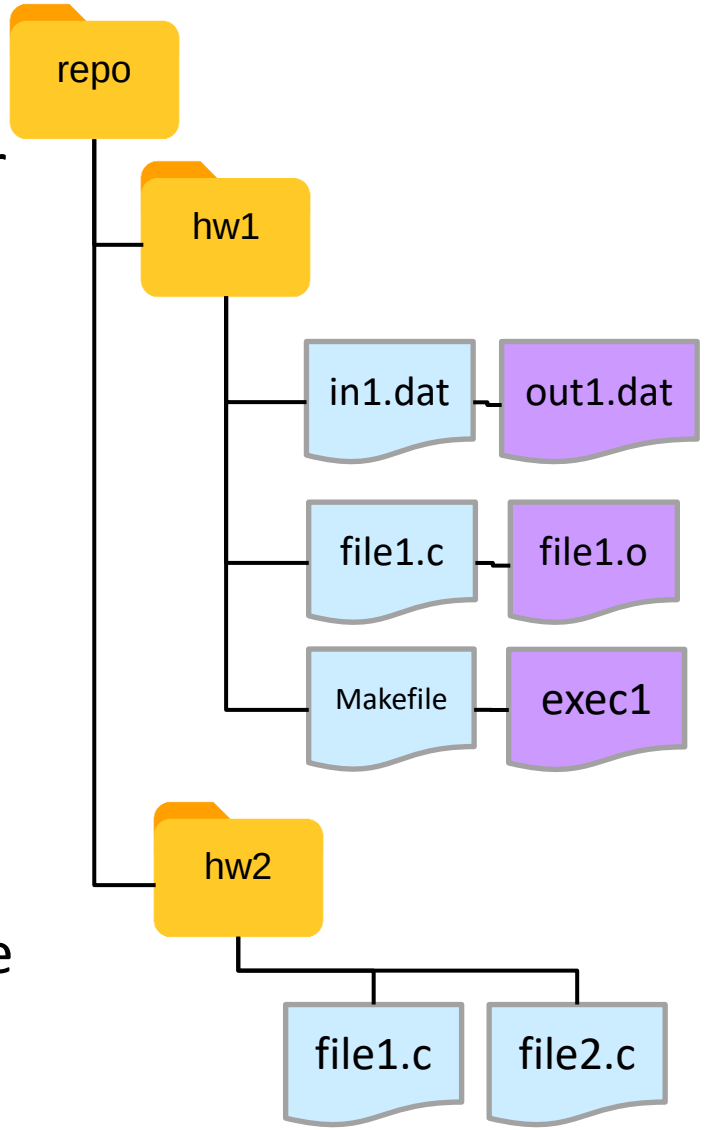
Git

- Git is a version control system
  - Stores "snapshots" of files (usually code) in a repository (think folder) at a explicit points in time that you choose
    - No more making backup copies
  - Allows easy updates to a view of the code at some historical point in time
- Git is "distributed" (often via Github)
  - Allows the repository to exist on various machines and each store new updates (aka "commits")
  - Github holds the central repository
  - Updates can be communicated to each "clone" of the repository by "push"-ing updates to and "pull" updates from the central repository on Github



Repositories

- We generally organize our code and related files for a project in some folder
  - We will use the term "repository" for this ***top-level*** folder when it is under "version-control"
- Your repository can have some files that ARE version controlled…
  - Source code, Makefiles, input files
- …and some that ARE NOT
  - Object files, executables, output files
- Version controlled (aka 'tracked') files have their version history saved and are uploaded to Github

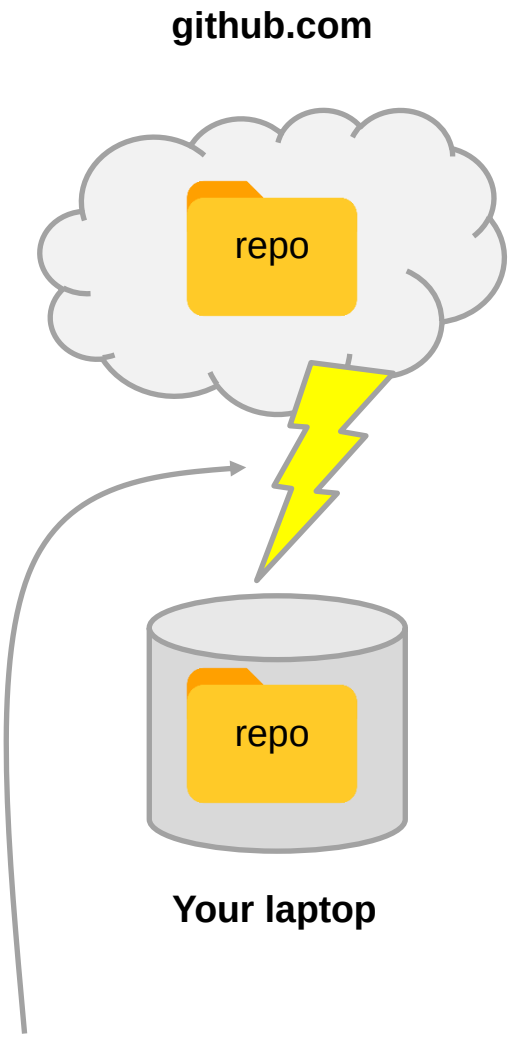


Keys

- Each time you upload or download from Github to/from your repository you will need to authenticate
  - By default you can provide your username/password
  - But since you should be uploading often it's easier to setup an SSH key
- To setup a key on your VM or other machine at the terminal:
  - `$ ssh-keygen -t rsa -b 2048 -C ttrojan@usc.edu`
- Then open the contents of `~/.ssh/id_rsa.pub` in an editor
- Login to Github, go to your Settings (upper right) and find the "SSH Keys" tab
  - Click New SSH Key
  - Provide a name (your choice) for this key and then paste the contents of id_rsa.pub into the Key textbox
  - Click Add SSH Key

Cloning Repos

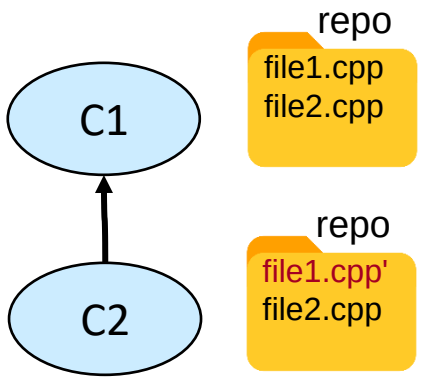
- Cloning a repo brings a copy of the specified repository onto your local machine
  - `git clone` *`url-of-repository`*
  - Only needs to be performed once per machine
- You can now perform additions, modifications, and removals locally (without being connected)
- Allows the two repositories to be synchronized in both directions via `git push` and `git pull`




```
git clone git@github.com:usc-csci104-summer2021/hw-ttrojan.git
```

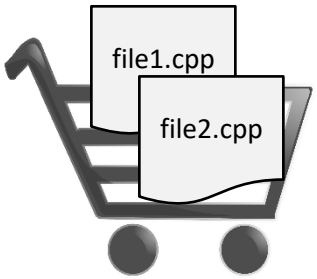
Adds and Commits

- Repositories are updated by performing commits
- We first indicate all the files we want to commit by performing one or more adds via `git add`
  - Like adding things to your cart
- Then we perform a `git commit` of the added files
  - Like checking out…this is when the snapshot is taken
- Note: Don't add folders, just files…folder structure will be added automatically

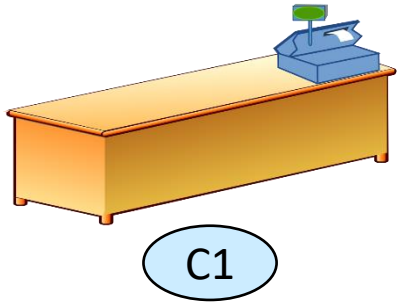


Sample Sequence:

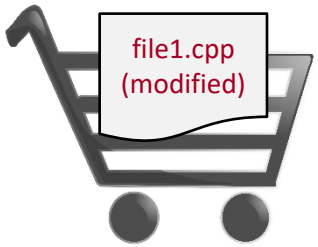
1 `git add file*.cpp`



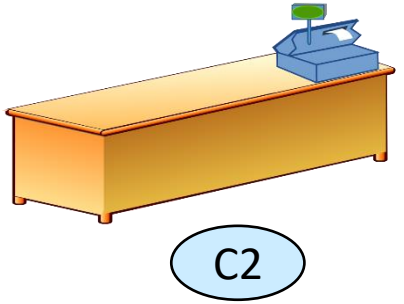
2 `git commit -m "Initial"`

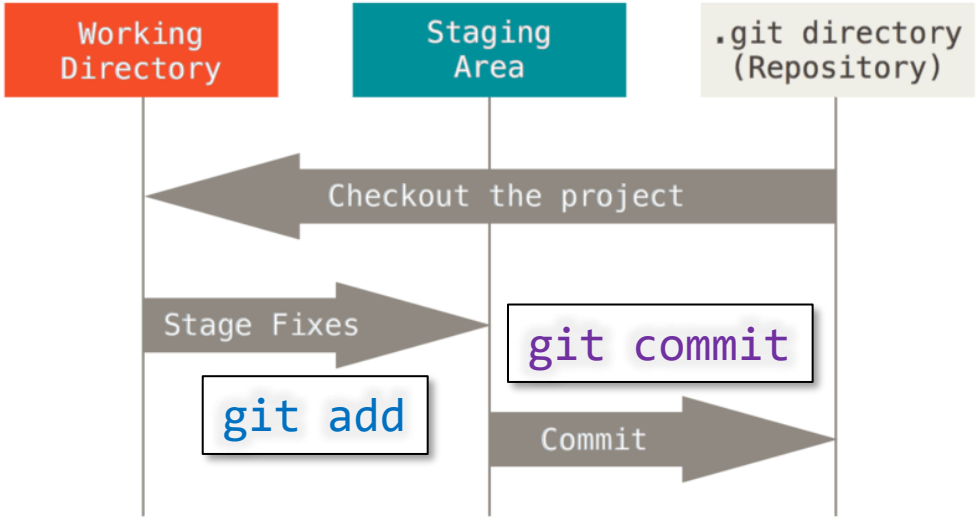


3 `git add file1.cpp`



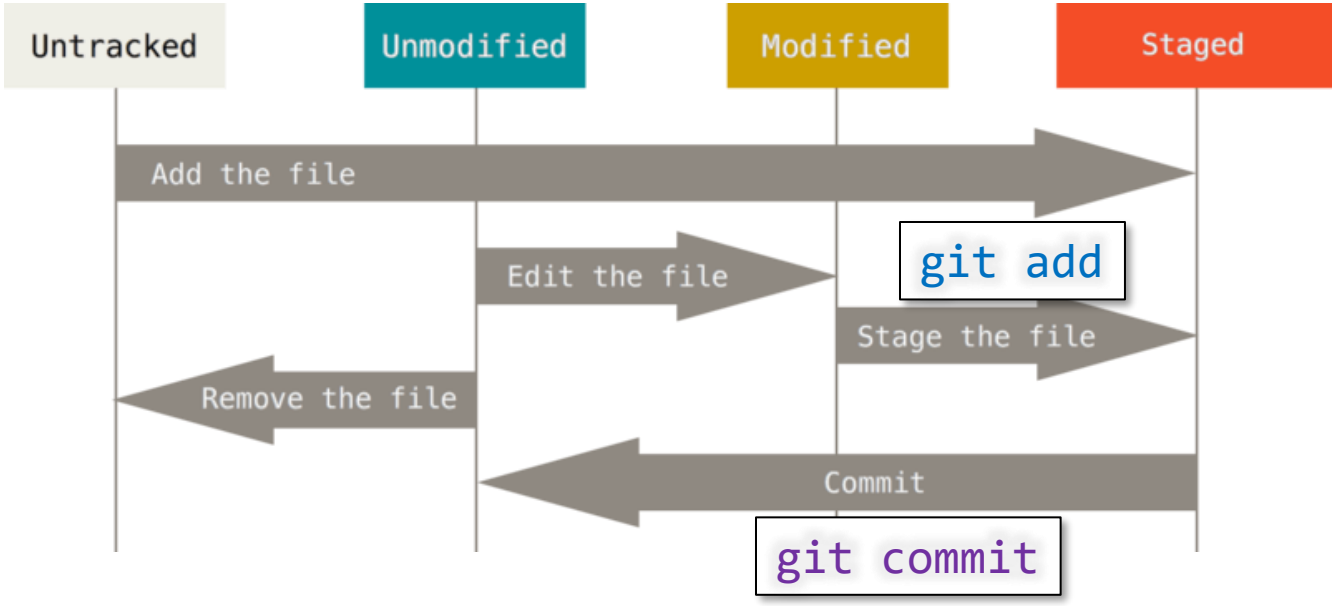
4 `git commit -m "Updated"`





Git "Locations"

https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

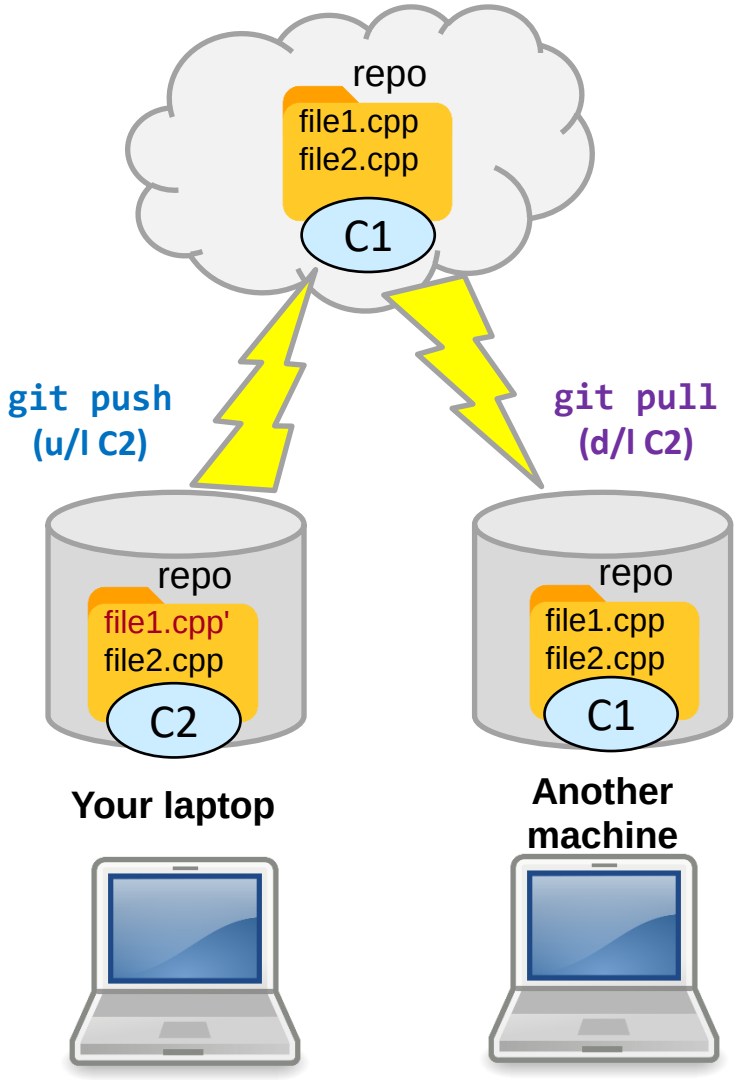


Git File Lifecycle

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

Push and Pull

- Suppose we make changes to our local repository
  - `git add file1.cpp`
  - `git commit -m "Added func2"`
- We upload the updates to the remote repository via a push operation
  - `git push`
- Another clone of the repository can download any updates from the remote repository via a pull operation
  - `git pull`



Summary

- `git add` *`file(s)`*
  - Stage a file to be committed
- `git commit -m "Change summary"`
  - Makes a snapshot of the code you added
- `git checkout -b` *`branch-name`*
  - Create a branch and switch to it
- git pull
  - Download commits from your remote repository
- git push
  - Upload your local commits to the remote repository
- `git checkout` *`branch-name`*
  - Switch to a new branch
- `git merge` *`other-branch-name`*
  - Merge the commits from other-branch-name into current branch
- HEAD is synonymous with the (current branch's) latest commit
- `origin` is usually the remote name for your repo on github
- `upstream` is usually the remote your repo was forked from (must be added)

Helpful Links

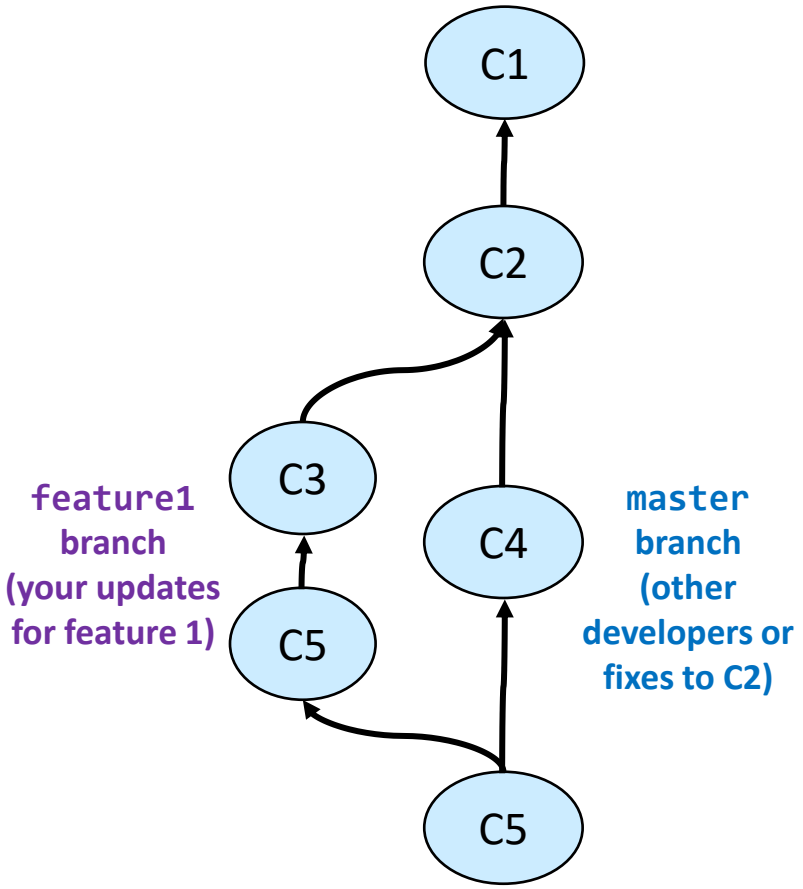
- https://help.github.com/
- Tutorial
  - https://learngitbranching.js.org/ (Do only the lessons below)
    - **Main Tab: Level 1 - Intro to Git Commits**
    - **Remotes Tab: Level 1: Push & Pull – Git Remotes**
- Cheat Sheets
  - https://services.github.com/on-demand/downloads/github-git-cheat-sheet/ (web version)
  - https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf (print version)
- FAQ for common Github Issues (when you encounter a git issue doing your HW check this FAQ first)
  - http://bytes.usc.edu/cs104/cs-faq.html

(Probably not necessary for 104)

ADVANCED GIT (FOR REFERENCE)

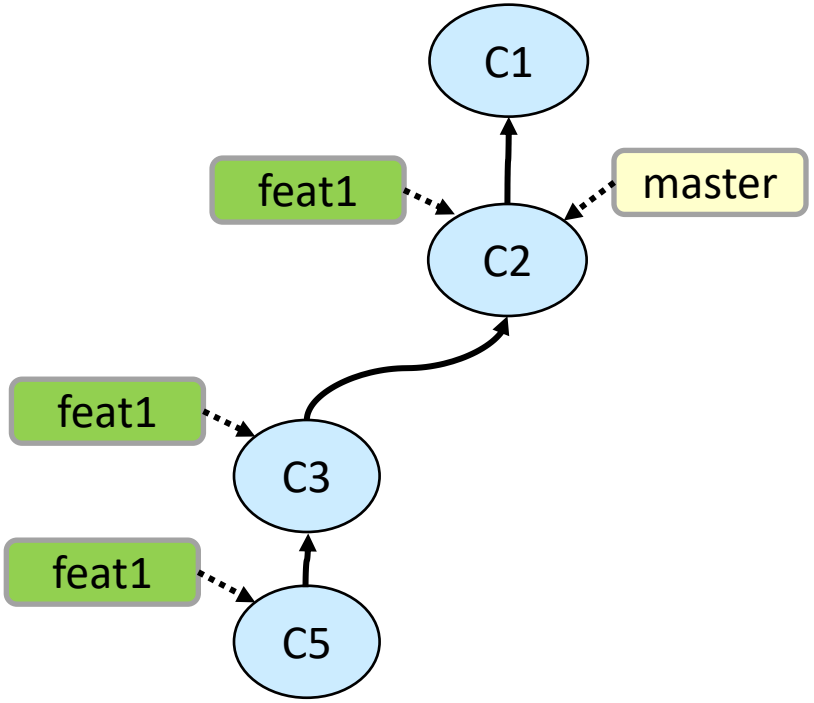
Branches Motivation

- Branches are useful when you are adding some new feature/fix, especially when others developers may also be doing the same by giving a separate sandbox to work in
- Branches allow you to
  - Grab the code from a particular starting point (i.e. commit)
  - Modify code, add, delete and commit
  - Merge the code back into the master branch



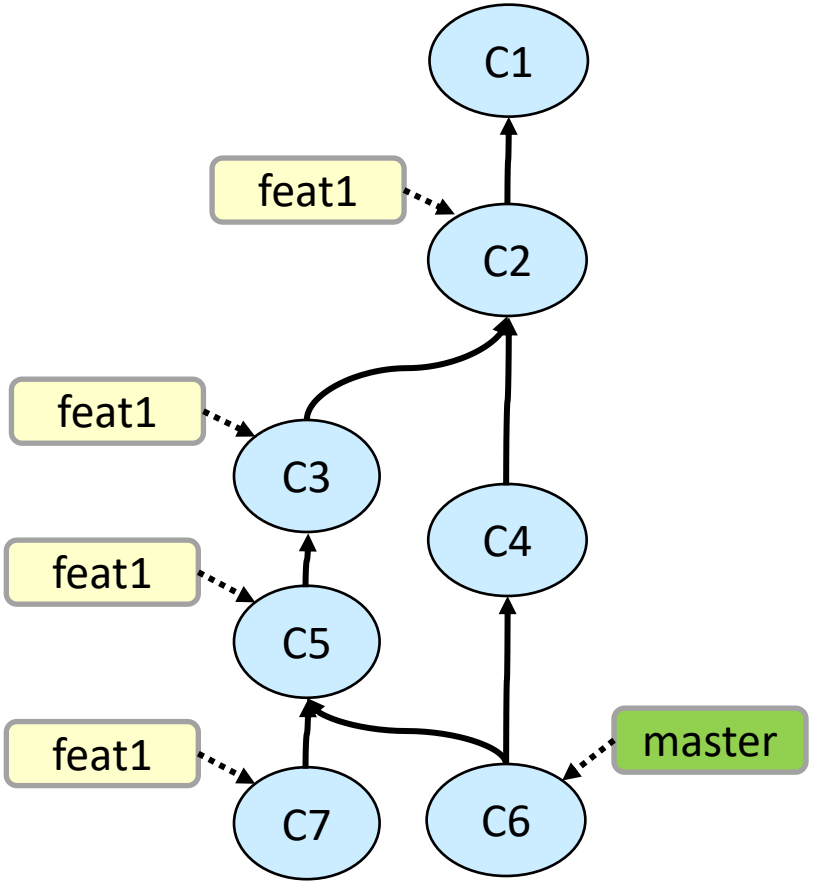
Branches (1)

- Each commit has one parent
- Branches are just names that can be associated with a commits
  - 'master' is the default branch
  - Created using:
    ```
    git checkout -b branch-name
    ```
- You can only be working on one particular branch at a time
- Any commits are applied to the current branch
- Example:
  - `git checkout -b feat1`
  - `git commit -m "Added part1"`
  - `git commit -m "Added part2"`



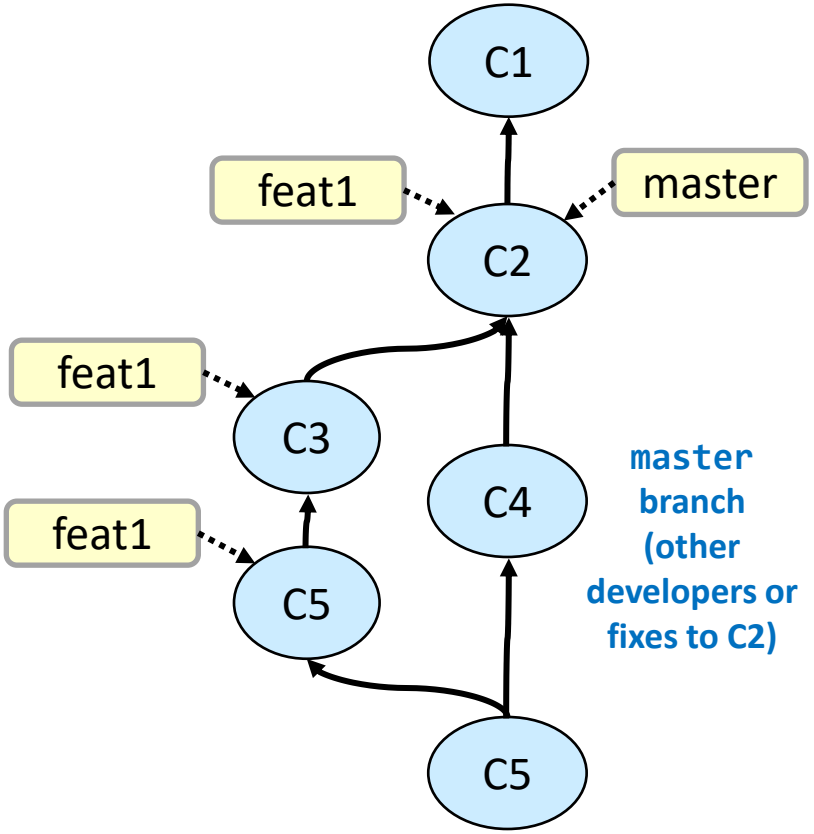
Branches and Merging

- We can switch between branches using `git checkout` *`branch-name`*
- Example:
  - `git checkout master`
  - `git commit -m "Fix bug 1"`
- Two branches can then be merged together via: `git merge` *`branch-to-merge-in`*
- A merge is a special commit with two "parents" and combines the code
- Example:
  - `git merge feat1`
- Note: You must be in the branch that will be updated with the code from the specified branch
  - The specified branch remains independent (you'd have to do another merge to sync both branches)
  - `git checkout feat1`
  - `git commit -m "Separate change"`



Conflicts

- If the merge encounters updates that it is not sure how to combine, it will leave the file in a conflicted state
- Can find conflicted files via:
  - `git status`
- Contents of conflicted files must be manually combined
  - Conflicted areas are highlight with `<<<<`, `====`, `>>>>` with the contents of each branch
  - Edit the file to your desired final contents
  - Then **add** and **commit**

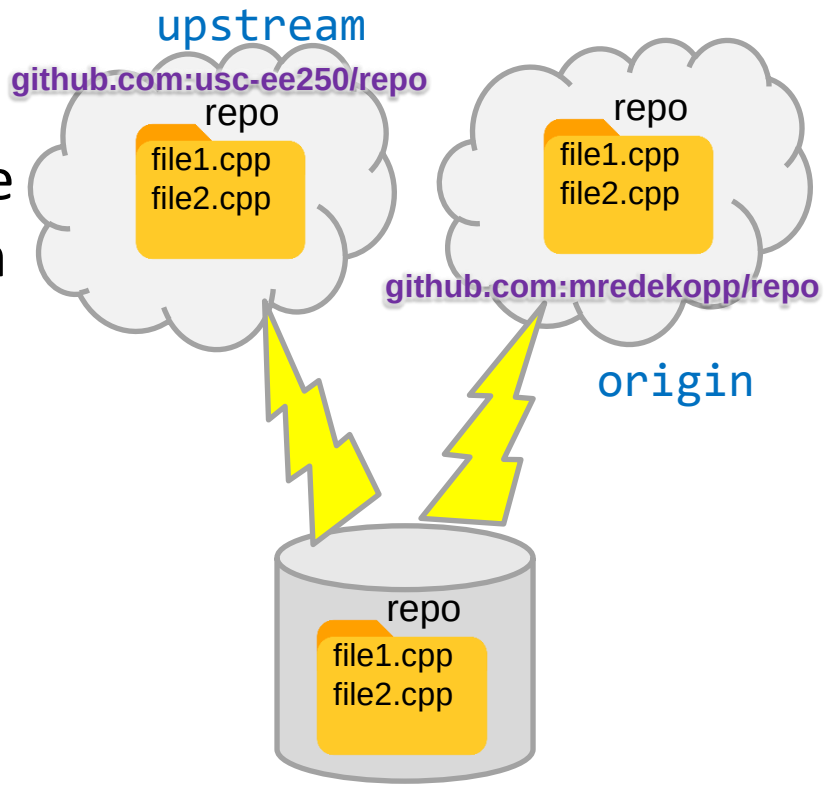


Sample Conflicted File

```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> feat1
```

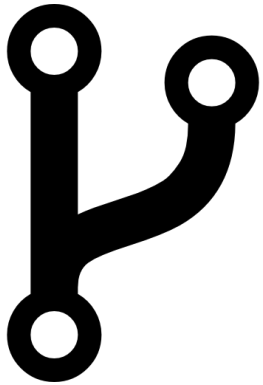
Remotes

- Remotes are just like their name indicates: remote locations where we can push and pull (or fetch) data from
- To list remotes
  - `git remote -v`
- To add a remote
  - `git remote add` *`name remote-url`*
  - `origin` is the common name for the remote repo from which you cloned
  - A remote is just an association of a name to a repo URL
- To choose & push a particular branch to a remote
  - `git push -u` *`remote local-branch`*



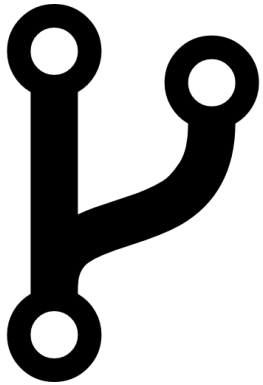
Forks

- A fork is a "copy" of a repository
  - Essentially a new repo whose starting point is the current state of the original, "forked" repo
  - Allows changes to be made (like a branch) or starting a new project based on some current codebase
    - If the original fork changes, there are means to pull those updates into your fork
  - It is possible to fork a fork ☺
- Example
  - The sensors we use have Python library support available on Github
  - We have forked that repo and made some changes for EE 250
  - You will then fork our repo (i.e. a fork of a fork) and modify it with your lab group
    - If we make changes in our repo, you can easily bring them into your fork



Upstreams

- Common definitions
  - upstream: The parent repository from which you forked
  - downstream: The forked ("child") repository (i.e. your repo)
- Common usage
  - The upstream fork can be thought of as just another remote
  - While the remote named origin usually refers to your fork on github, the remote named upstream usually refers to the parent of your fork
- Setting up access to the upstream fork
  - See https://help.github.com/articles/fork-a-repo/
  - `git remote -v`
  - `git remote add upstream` *`parent-fork-url`*
- Updating your code from the parent fork
  - `git fetch upstream`
  - `git checkout master` (can be skipped if you aren't using branches)
  - `git merge upstream/master`



An Example

- Suppose we create a repo for you: `p1-ttrojan`
  - It comes preloaded (because of actions we took) with some code that was from our own repo: `p1-skel`
  - `git clone git@github.com:usc-csci104-summer2021/hw-ttrojan`
  - `cd p1-ttrojan`
  - `# You make changes; add, commit, push`
- Now we make changes to `p1-skel`, how can you get and merge those changes in?
  - `git remote -v` `# list the remotes`
  - `git remote add upstream git@github.com:usc-csci104-summer2021/p1-skel`
  - `git fetch upstream` `# d/l changes to a temp area`
  - `git checkout master # make sure you're in your master branch`
  - `git merge upstream/master` `# Update your code`