CMOR 420/520
Computational Science

Git version control

Informal version control

- Tracking changes: looking for the newest version of codes in your email, Dropbox, Google Drive, etc.
- Collaboration: pair coding, adding new changes one-by-one.
- Releasing new versions: post something on your website
- Continuous integration: running test cases by-hand

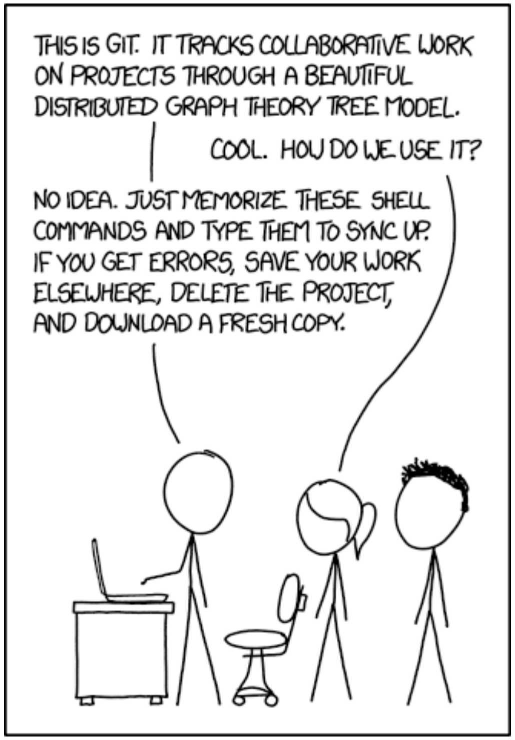
Modern version control

- Tracking changes: just view your “commit history”
- Collaboration: branching, merging, pull requests
- Releasing new versions: through a repository hosting service
- Continuous integration: automatable

Modern version control: aims to formalize and automate many steps in software engineering

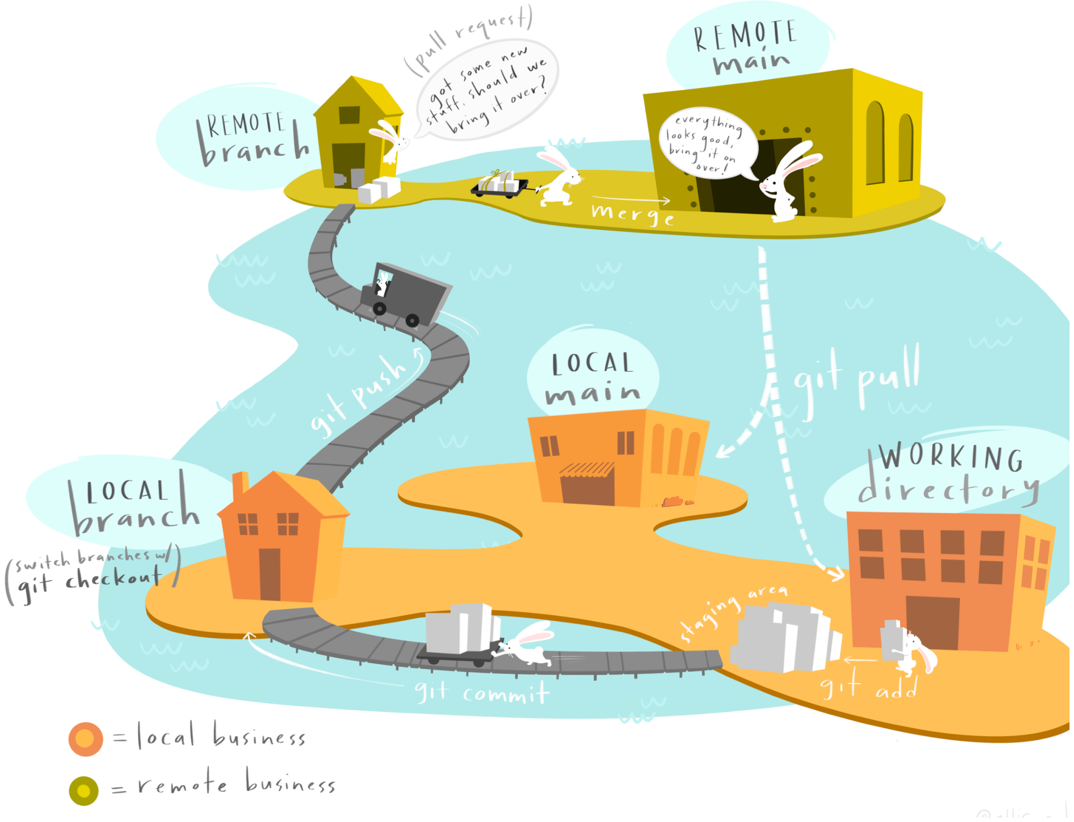
Git version control

- Conceptually: Git tracks changes to a *repository* by modeling its history
  - Mathematically: Git history is a directed acyclic graph (DAG)
- In practice, because Git's syntax is confusing, you often end up memorizing a few commands.
- It's important to have a conceptual understanding of what Git does


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Git terminology

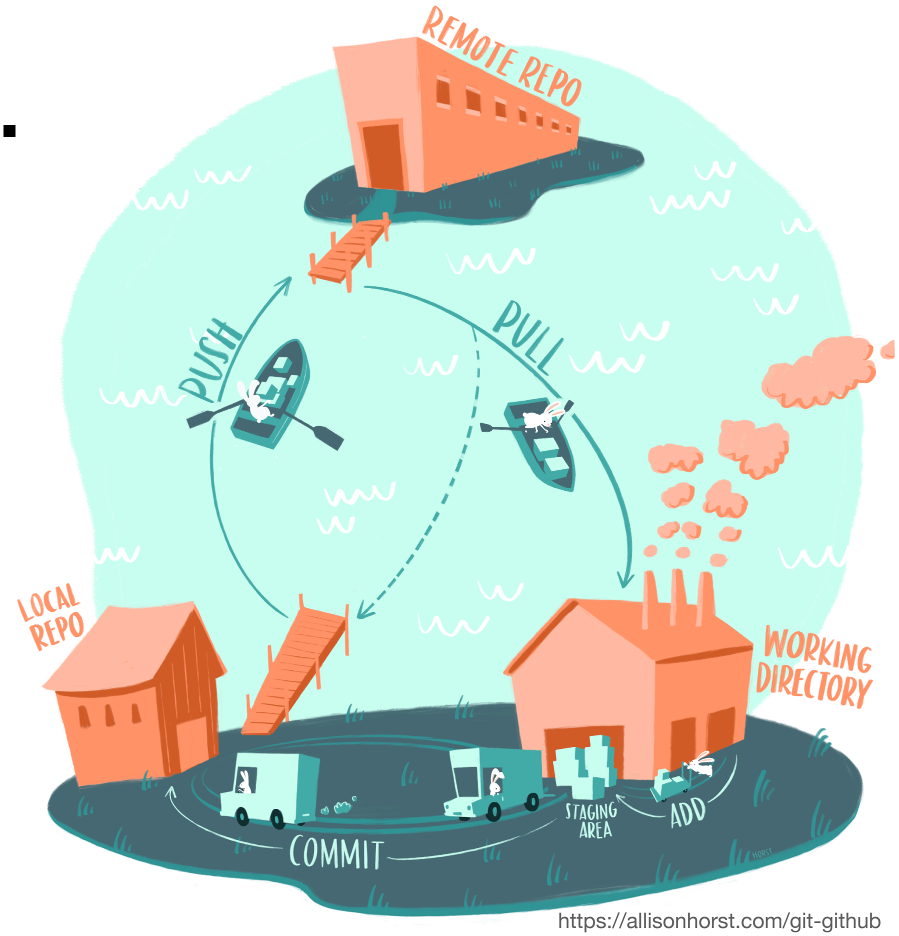
- You have a “local” branch on your computer
- You can optionally sync to a “remote” repo hosted online
- You add “commits” to a local branch and “push” or “pull” commits from the remote branch



https://allisonhorst.com/git-github

Git terminology, cont.

- git add: *stages* some changes (easy to undo)
- git commit: commits these changes (adds a new node in the git graph)
- git push: syncs the updates with a remote (online) repository
- git pull: syncs your local repository with a remote



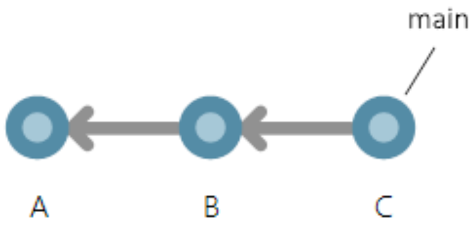
https://allisonhorst.com/git-github

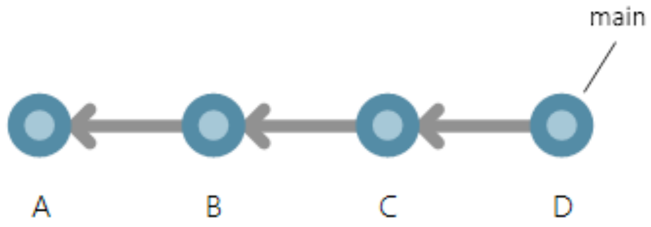
Git commit graphs

- Git tracks a *directory* and all files and subdirectories inside it

- Each commit is a snapshot of the directory, and depends on a parent (e.g., A <— B means that the repo state at B was created by modifying the repo state at A)

- Commits cannot be erased; you can undo a change but this will still result in new commits (e.g., even undo actions are tracked)

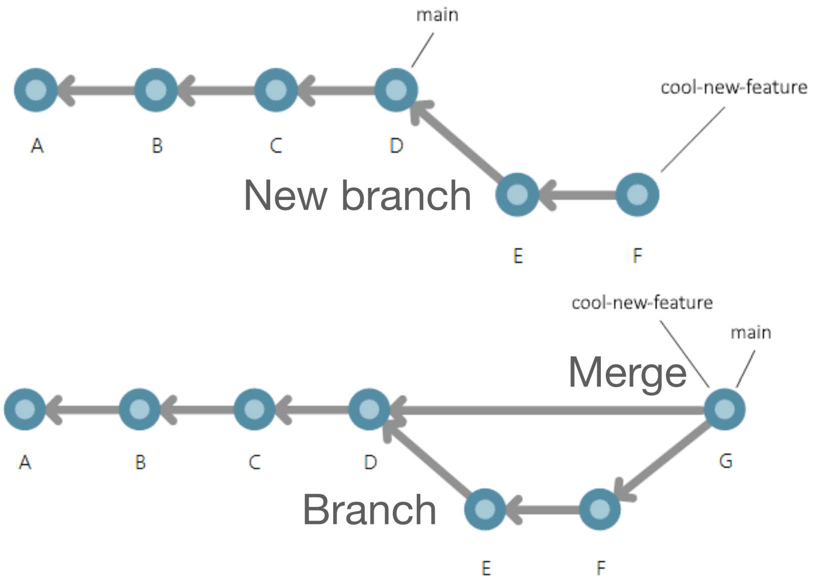




Git branching and merging

- Common use of Git: using branches to try out new features

- You can create a new branch with its own commit histories
- Can switch back and forth between the old and new branch (git checkout)
- When you *merge* a branch back into the main branch, it has two parents (G <— D, F)



Merge conflicts

- Merging doesn’t always happen automatically; sometimes you encounter *merge conflicts* where Git can’t merge one branch into another without destroying or overwriting in one branch
- If you encounter a merge conflict, Git will just ask you to manually merge two branches together (e.g., specify what you want to keep and what you want to get rid of)
- Github has options for different merge strategies; these don’t impact the final result

Git in practice

- First install git (check if it’s already installed)
  - On Linux terminals, “sudo apt install git” should work
  - On Mac, installing Xcode (via “xcode-select --install") should work. This will also install other tools that we’ll use later.
  - You can also install Git via Homebrew (https://brew.sh/) on Mac
- Git uses for your identity (name and email) to sign commits
  - git config --global user.email “your_email@hostname”
  - git config --global user.name “Your Name”

Your first (local) Git repo

- Create a directory (e.g., “my-first-git-repo”) and “cd” into this directory
- Run “git init” inside this directory and add a file (try “echo “This is some text” > First_file.txt”)
- Run “git status” to track changes (none should be tracked)
- Stage any changes you plan to commit, e.g., “git add First_file.txt”.
- Run “git status” again - the file should be *staged*

Your first (local) Git repo, cont.

- Once you’ve staged your change, you can run the following:
  - git commit -m “My first commit”
  - The “-m” flag is your commit message; every commit requires a message. If you don’t specify this, Git will open a text editor for you to write the message.
- “git log” will list the commit history

Connecting to a remote repository

- This is all local so far; usually you want to also connect to a remote repository. I'll focus on Github for this course. Two approaches:
  - You can create the local repo and then connect it to the remote via e.g., "git remote add origin https://address "

    * address is given to you by repository owner, or found on github
  - You often want to set "git push --set-upstream origin main" so that you don't have to specify *where* to push changes.

Creating a local repo from a remote repo (easier)

- Easier: create the remote repo on Github and use e.g., “git clone https://address “
- This sets remote, upstream, origin, … automatically.
- Can now “git push” committed changes to your remote repo, and “git fetch” or “git pull” changes in the remote repo to your local repo!
    - Note: “git pull” will automatically bring in changes from your remote, while “git fetch” just downloads the changes into a new branch (usually “origin/branch_you_fetched_from”)

Exercise (set up Github)

- Create a Github account at https://github.com if you don't already have one.
- Register for the Github Student Developer Pack at https://education.github.com/pack. This provides Github Pro for free,

* Make a Github Token, if not done the first day

- Create a ***private*** repo named "cmor-420-520 "

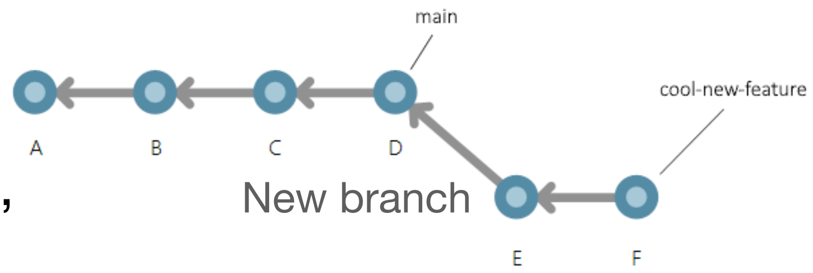
Connecting to a remote repo in practice

We are going to connect to the remote repository you just made:

- git clone $https : //address$
- Create a file *Exercise_file.txt*, add, commit, and push to the remote repository

Branching

- To create a new branch, run “git branch *new_branch_name*”
- To switch between branches, use “git checkout *branch_name*”



New branch

- “git diff main ” will show the difference between the current branch and the main branch

* “git push origin branch_name” will push to the remote repo, to branch_name

Stash

- If you have local changes but want to switch branches this is a problem.
- ’git stash’ allows you to store local changes
- ’git stash pop’ will access the stored changes and insert them back into your files

Exercise

- Make your own remote repo, call it *CMOR_Exercise*
- connect to it
- create a file in the main branch
- make a new branch, call it *new_branch*
- switch between the branches
- create a different file in *new_branch*

- Extra: use git status to check what file, use 'git diff main' to check the difference between branches

Exercise

- make a new branch, call it *new_branch*
- switch between the branches
- create a different file in *new_branch*
- add, commit, and push to the repo. Use the push command:

```
git push origin new_branch
```

- Extra: use git status to check what file, use 'git diff main' to check the difference between branches

Merging

- Merging combines branches into one branch
- If a file has been edited in different branches, a merge combines those changes
- If the edits conflict, the terminal will tell you. You must resolve the conflicts yourself!

```
> git merge troublesome-branch
Auto-merging conflict.txt
CONFLICT (add/add): Merge conflict in conflict.txt
Automatic merge failed; fix conflicts and then commit the result.
```

```
> cat conflict.txt
<<<<<<< HEAD
Foo baz bar
=======
"Foo bar"
>>>>>>> troublesome-branch
```

Reset, revert, and checkout

- You can unstage files for commit using ’git reset’. These changes are not deleted.
- You can unstage and delete changes for commit using ’git reset ——hard’

- ’git revert’ will revert a commit

- ’git checkout’ can be used to switch to a specific version (commit) use ’git checkout *commit_number*’
    - the commit number appears in the output of ’git log’

```
> git log
commit 74357614b358d648c9a12cb5c2741077aef7c88e (HEAD -> master)
Author: Joey Huchette <jah9@rice.edu>
Date:   Sun Sep 1 15:23:17 2019 -0500

    Part 3 of 3.
```

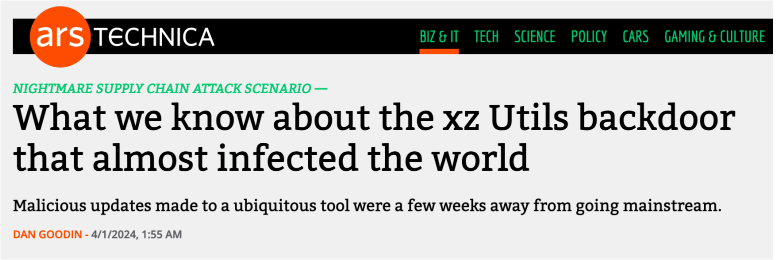
Tagging git commits

- Developers often release new versions of a library or codebase (e.g., v0.1.0, v1.0.0, etc).
    - It can be helpful to “git tag” a commit as a release

- Example: you can tag the current commit that you are on. For example, “git tag v1.0.0” creates a “v1.0.0” tag.
    - Once a tag is defined, it’s like a commit alias, e.g., “git checkout v1.0.0” will work. “git tag” list current tags.
    - Note that you have to “git push” a tag, e.g., “git push origin v1.0.0”

Open source software and pull requests

- Open source software relies on contributions from people you don't know. How do you safely test their contributions?
  - Pull / merge requests: a contributor makes a copy of the repository, tests some changes, and proposes the changes through a pull request.
- PRs provide safeguards, but still relies on developers to check the submitted code.



NIGHTMARE SUPPLY CHAIN ATTACK SCENARIO —

What we know about the xz Utils backdoor that almost infected the world

Malicious updates made to a ubiquitous tool were a few weeks away from going mainstream.

DAN GOODIN - 4/1/2024, 1:55 AM