# Lab 1 – Intro/Git

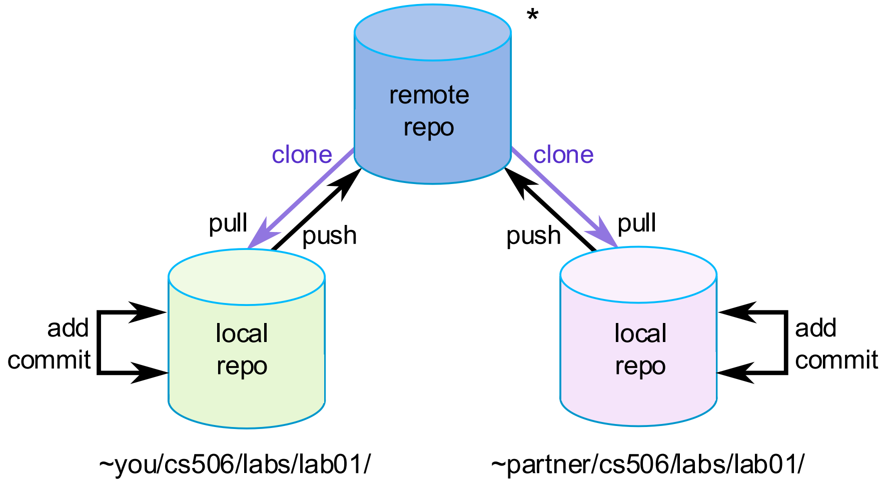
CS 506

## What are Git and GitHub?

- Git is used to manage timelines of a project (a.k.a. the repository)
- Git is a version control system
- GitHub is a website to backup and host the timeline of your project!
    - We utilize Git to push updates to our projects to GitHub as a means of backing up our work
    - For example, if my computer's SSD becomes corrupted (beyond repair), then I can still access my projects via GitHub!
    - Additionally, we use Git and GitHub to collaborate on and share projects

## Use case motivation

- For each repo/project, I want to write code where:
    1. Iterating on (and keeping track of) different versions of my code is easy
    2. Work is backed up to and hosted on the cloud
    3. Collaboration is seamless!

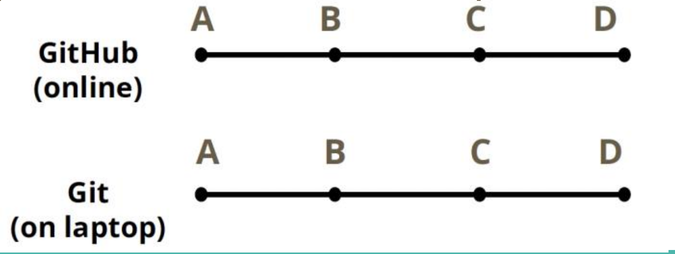


*Taken from Swarthmore College CS31

## Why do we care about iterating over versions?

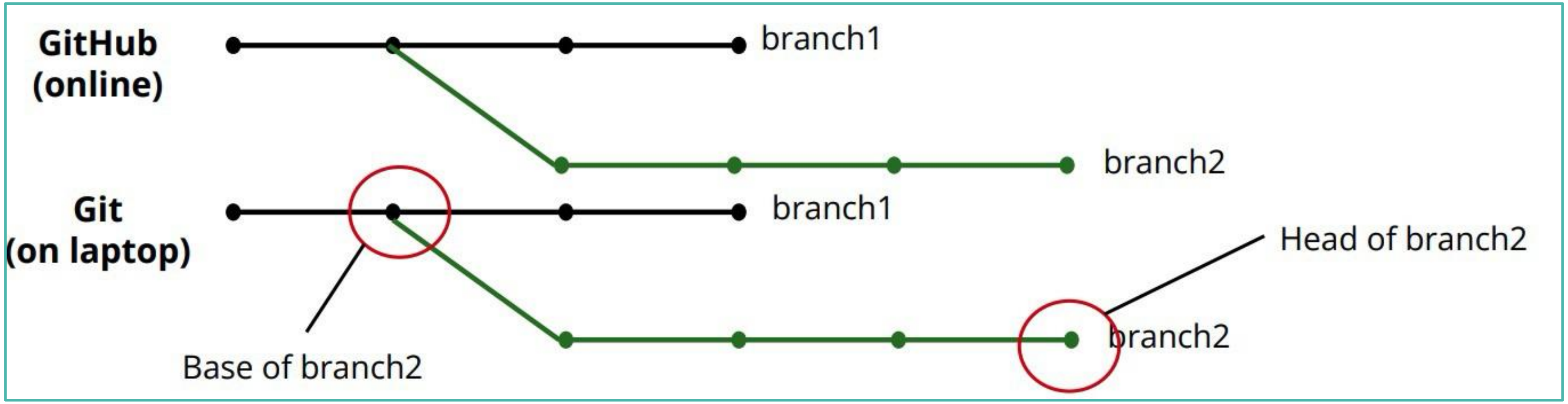
- Small, incremental commits allow developers (us) to easily rollback or revert to previous, stable versions if errors are introduced.
    - Commit point: a point in time where we save or “snapshot” our code base
        - Each commit point is assigned a unique ID, known as an SHA or hash
        - Unique ID allows us to refer to each state at any time
- The ability to iterate over versions also allows us to reconcile our local codebase with the GitHub timeline.
- Additionally, we may want to implement a feature on a different version of our codebase.

Question: how do we implement a feature at a previous commit point? The answer is branching!



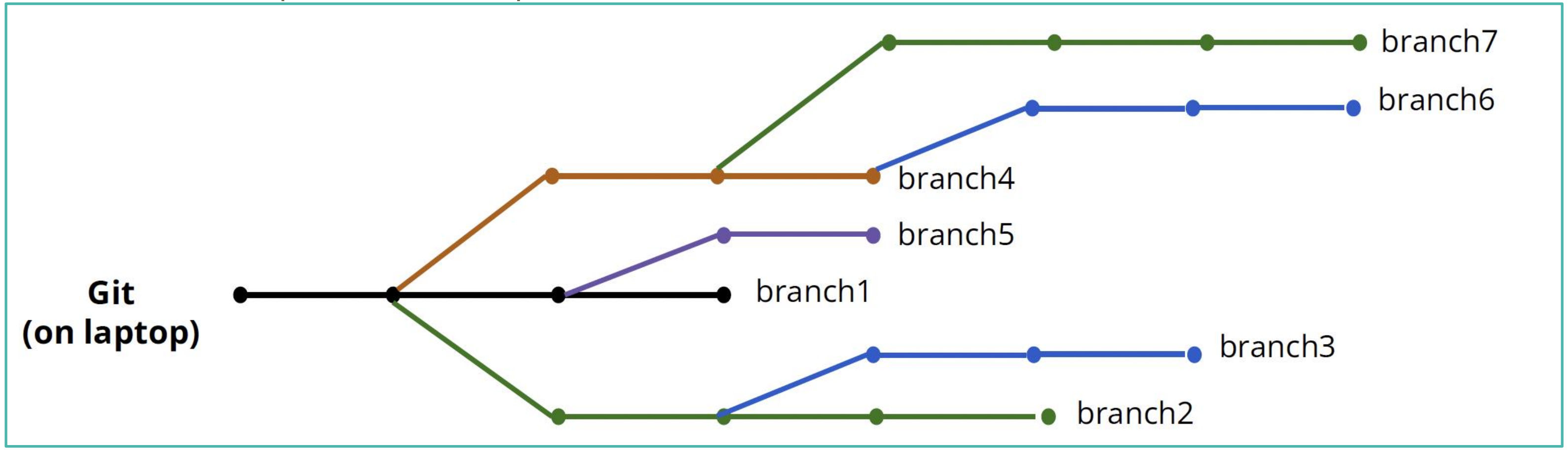
## Branching

- When iterating on different versions of our code base, we need:
  1. A way to preserve both versions of the history
  2. A way to overwrite history if we choose
- Branch off a particular commit point to create a new timeline
  - We can push commits per branch!



# Branching

- When iterating on different versions of our code base, we need:
  1. A way to preserve both versions of the history
  2. A way to overwrite history if we choose
- Branch off a particular commit point to create a new timeline
  - We can push commits per branch!



- We can branch off branches too!

## Branching conventions

- One branch must be chosen as the primary stable branch, typically called main
  - Other branches are typically named after the feature that is being developed or the major/minor version of the software/product
    - E.g., login-page version or v1.2 of a game

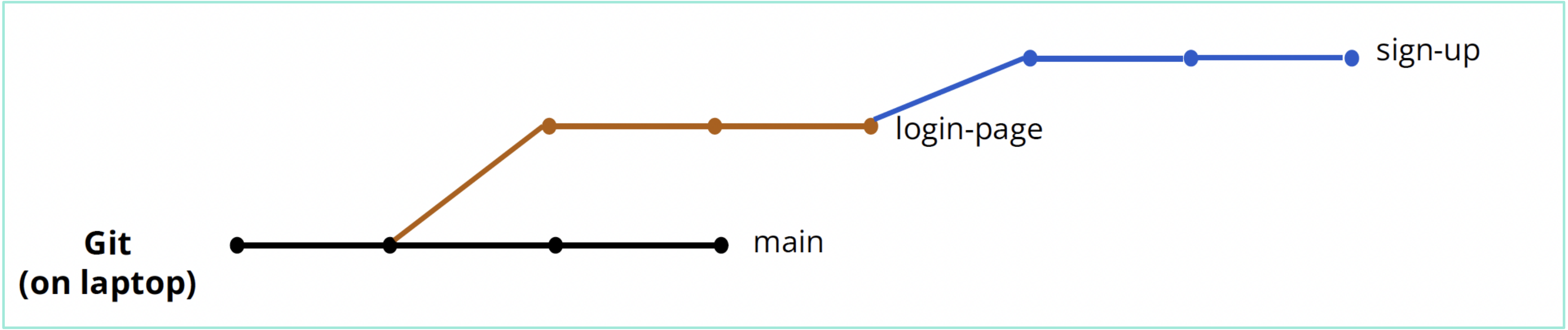
- Beware! Branching can get messy. Eventually we will want to clean it up

## Merging

- At some point, we will want to clean up the branches by merging them with the master/main branch or with each other
    - Merging is trivial if the base of one branch is the head of the other. Changes are simply appended
    - In many cases, this behavior does not apply, and we will likely deal with commit conflicts a.k.a merge conflicts
- A merge conflict arises when GitHub cannot automatically combine commits because they make incompatible changes to the same part of the code.
    - The programmer (you!) decides

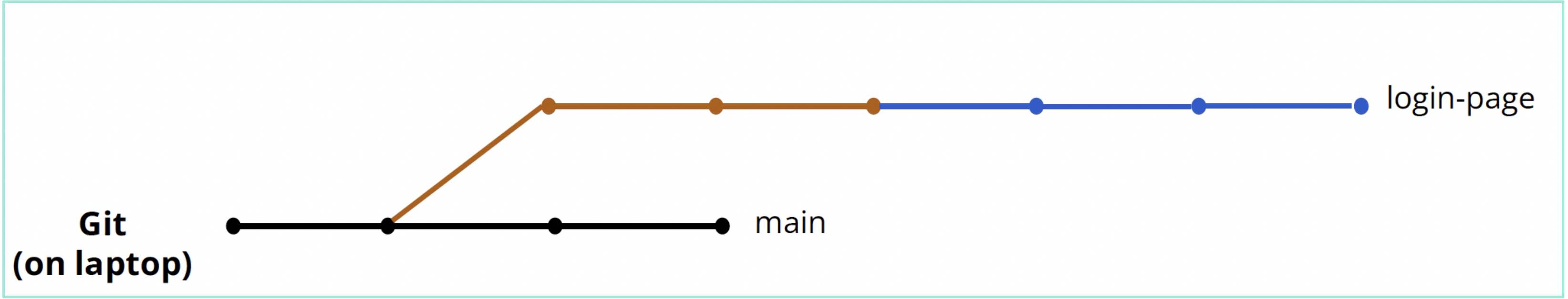
## Merging

- At some point, we will want to clean up the branches by merging them with the master/main branch or with each other



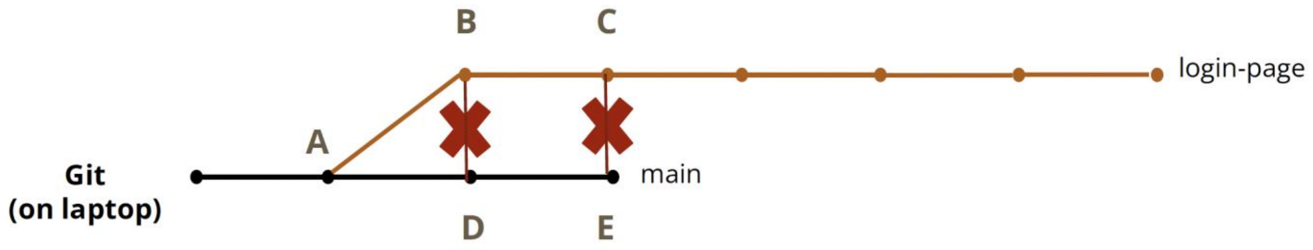
## Merging

- At some point, we will want to clean up the branches by merging them with the master/main branch or with each other



- A merge conflict arises when GitHub cannot automatically combine commits because they make incompatible changes to the same part of the code.
  - The programmer (you!) decides

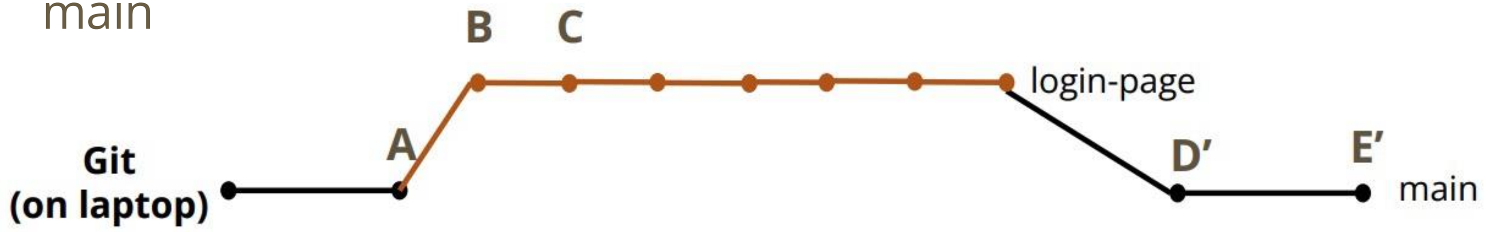
## Merge conflict



- It is possible that line 42 of **D**’s *server.py* is different from **B**’s *server.py*
    - As managers of the code base, we must resolve this conflict and decide which *server.py* we want so we can merge login-page with main
- After we resolve, the merge conflict, what does the actual merge look like?
    - This is the question of rebasing

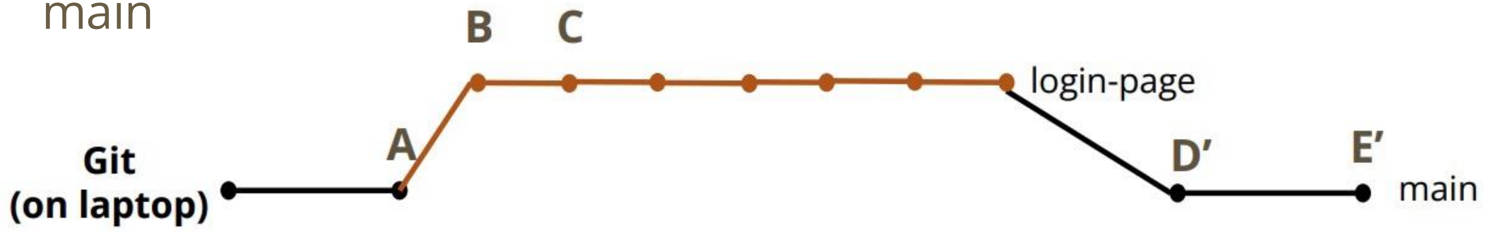
## Rebasing

Option 1: Incorporate the base of login-page at the “common ancestor” of main

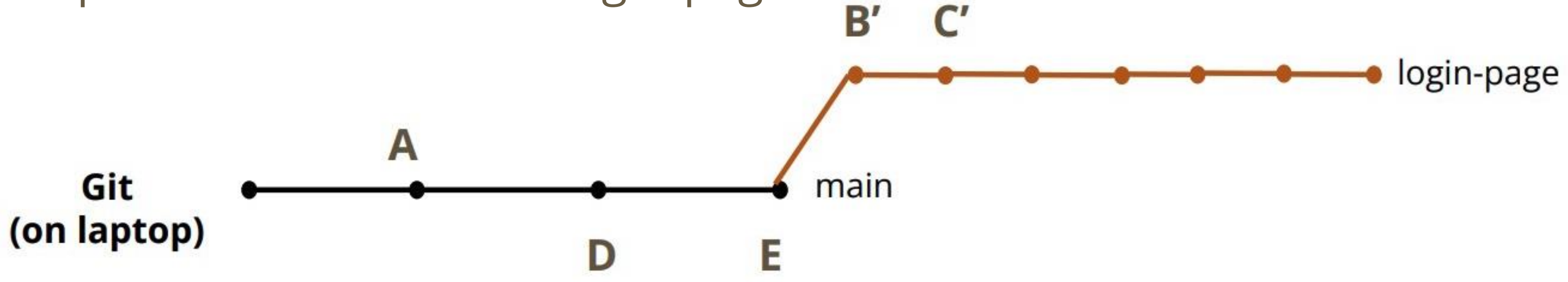


# Rebasing

Option 1: Incorporate the base of login-page at the "common ancestor" of main



Option 2: Move base of login-page to head of main



## Collaboration

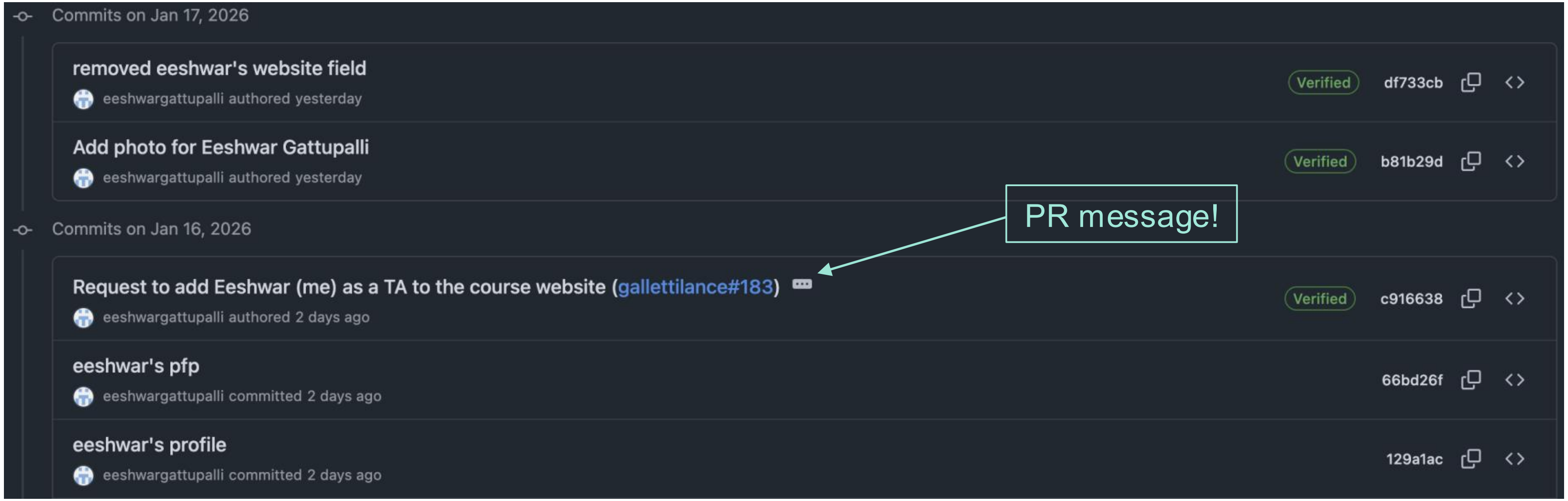
- Other repositories can be thought of as other branches
- SOP for collaborators to contribute code:
    1. Collaborator makes a copy of (forks) the main repository
    2. Collaborator makes changes on their fork
    3. Collaborator requests that part of their copy be merged into the main repository via a pull request (PR)
- This is the standard (almost) everywhere!

## Collaboration best practices

- Pull the latest early and often
  - i.e., retrieve the latest changes from the main repository
- Always create a new branch when developing on a forked repository. Avoid committing directly to main
  - This makes it easier to keep your main branch in sync with the main repository
- Keep branches short-lived: merge after a few days vs. after weeks
- Commit frequently and meaningfully
  - Detailed commit messages aren't just so you get full points on your programming assignment!

## Collaboration best practices

- Commit frequently and meaningfully
    - Detailed commit messages aren't just so you get full points on your programming assignment!



Commits on Jan 17, 2026

removed eeshwar's website field
eeshwargattupalli authored yesterday — Verified df733cb

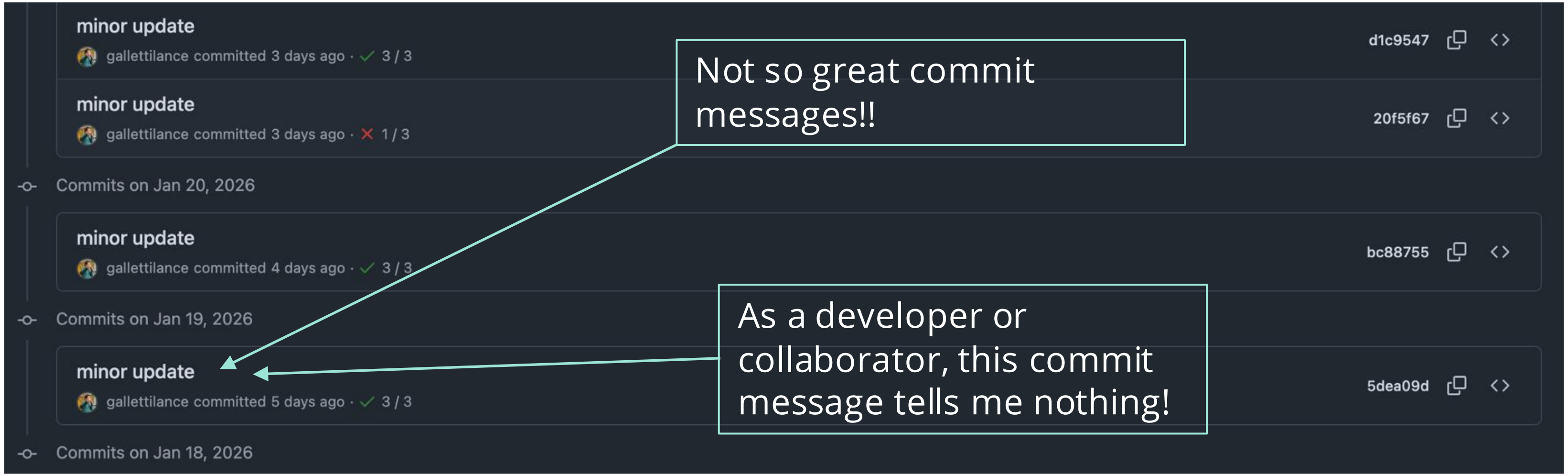
Add photo for Eeshwar Gattupalli
eeshwargattupalli authored yesterday — Verified b81b29d

Commits on Jan 16, 2026

Request to add Eeshwar (me) as a TA to the course website (gallettilance#183)
eeshwargattupalli authored 2 days ago — Verified c916638

eeshwar's pfp
eeshwargattupalli committed 2 days ago — 66bd26f

eeshwar's profile
eeshwargattupalli committed 2 days ago — 129a1ac



minor update
gallettilance committed 3 days ago · ✓ 3 / 3
d1c9547

minor update
gallettilance committed 3 days ago · ✗ 1 / 3
20f5f67

Commits on Jan 20, 2026

minor update
gallettilance committed 4 days ago · ✓ 3 / 3
bc88755

Commits on Jan 19, 2026

minor update
gallettilance committed 5 days ago · ✓ 3 / 3
5dea09d

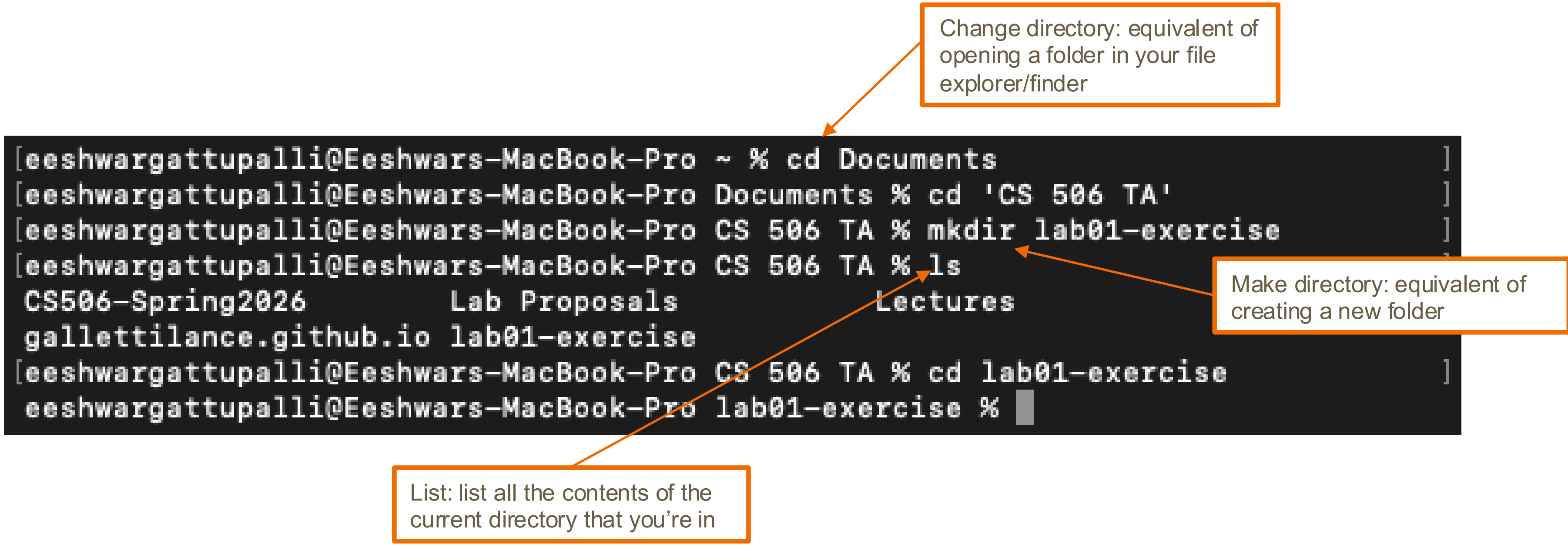
Commits on Jan 18, 2026

Not so great commit messages!!

As a developer or collaborator, this commit message tells me nothing!

# How to actually use Git and GitHub?

- Use your terminal to navigate to the directory where you want to develop



```
[eeshwargattupalli@Eeshwars-MacBook-Pro ~ % cd Documents                          ]
[eeshwargattupalli@Eeshwars-MacBook-Pro Documents % cd 'CS 506 TA'                ]
[eeshwargattupalli@Eeshwars-MacBook-Pro CS 506 TA % mkdir lab01-exercise          ]
[eeshwargattupalli@Eeshwars-MacBook-Pro CS 506 TA % ls                            ]
 CS506-Spring2026        Lab Proposals              Lectures
 gallettilance.github.io lab01-exercise
[eeshwargattupalli@Eeshwars-MacBook-Pro CS 506 TA % cd lab01-exercise             ]
 eeshwargattupalli@Eeshwars-MacBook-Pro lab01-exercise %
```

Change directory: equivalent of opening a folder in your file explorer/finder

Make directory: equivalent of creating a new folder

List: list all the contents of the current directory that you're in

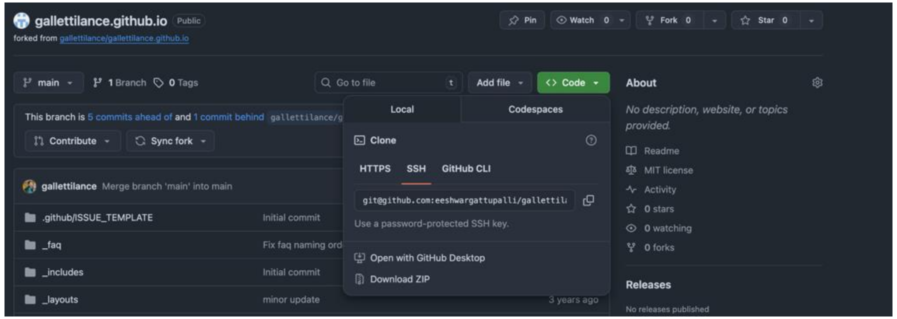
- Now that you're in the desired destination, you can begin using Git commands!

## Git commands: Local operations

- Basic commands
  - **git init**
    - Initialize a new repository (locally)
  - **git add <file-name1> <file-name2> … <file-nameN>**
    - Promotes changes from your working directory to the stages area. Tells Git which files you want to include in the next commit
  - **git commit –m "amazing commit message ☺"**
    - Saves the changes you've staged (from git add) as a permanent point in the repository's timeline, like a snapshot!
      - Annotates the snapshot with your amazing commit message

## Git Commands: Linking to GitHub

- Linking your local repository to a GitHub repository
  - **git remote add origin <SSH URL>**
    - Adds a ”remote” that points to a GitHub repository (identified by the SSH)
    - Once you create an empty repository on GitHub (without README, .gitignore, and license), copy the SSH URL and paste in the command above



## Git Commands: Interacting with GitHub

- Pushing to GitHub repository!
  - After staging your changes locally (utilizing git add and git commit), enter **git push origin main**
    - Pushes stages changes to the main branch of the GitHub repository being pointed to by the remote
- After forking the main repository (for collaborative efforts!)
  - **git clone <forked-repo-SSH-link>**
    - Locally clone the forked repository
  - **git remote add upstream <main-repo-SSHLink>**
    - Connect your forked repository to the original project
  - **git fetch**
    - Download most recent changes from remote (original) repository
  - **git pull upstream main**
    - Download most recent changes from remote (original) repository and apply to current local branch
  - **git push origin main (after staging)**
    - Push to forked repo

## Git Commands: Branching

- Creating and operating on new branches
  - **git checkout –b my-new-branch**
    - Creates a new branch then switches to it
  - **git branch my-new-branch**
    - Creates a new branch without switching to it
  - **git switch my-new-branch**
    - Switch to existing branch
  - **git push -u origin my-branch**
    - Creates origin/my-branch on linked GitHub repository
    - Links local my-branch to origin/my-branch
    - At this point, if you stage files and run git push, Git will push the staged files to origin/my-branch
      - If you run git switch main, stage your files, then push, Git will push to main. *Git always pushes to the current branch!*

## Git Commands: Merging/Rebase

- Merge: While on feature branch…
  - **git merge main**
    - Bring main’s history in the current branch. Essentially collapses the commit history from main into the current branch.
  - **git merge –abort**
    - Abort a merge with conflicts
- Rebase: While on feature branch…
  - **git rebase main**
    - Moves base of current branch to head of main