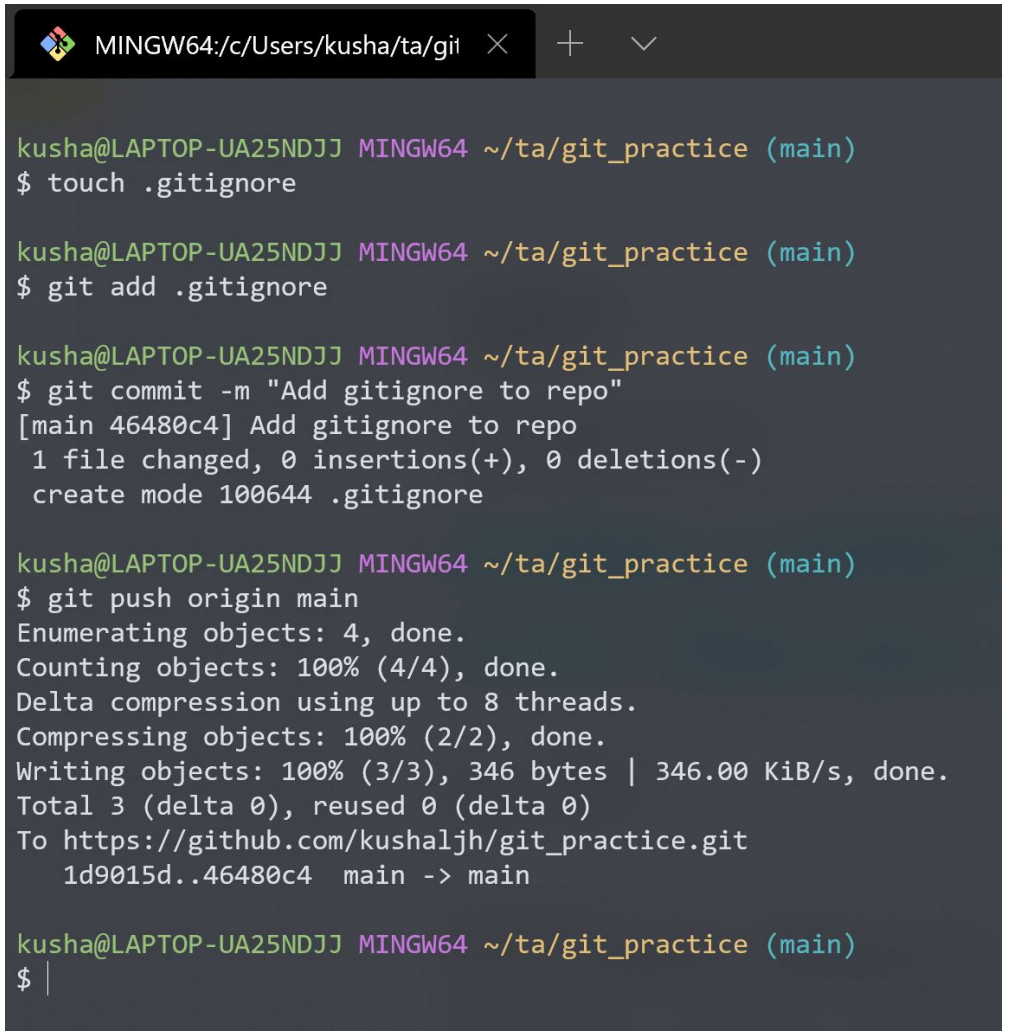
# Lecture 19: Git!

CSE 374: Intermediate Programming Concepts and Tools

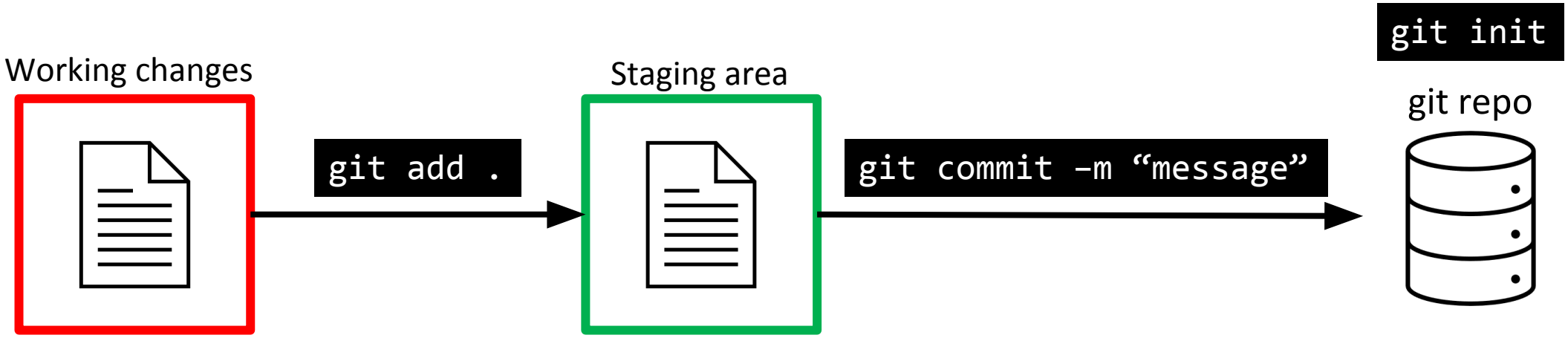
# QUICK RECAP

# Some "git" commands

- `git init`
  - Create a new empty git repo or convert an existing folder to a git repo
- `git add`
  - Preparing edited files to be saved (committed) to a repo
- `git commit`
  - Records (saves) changes to a repo
  - Accompanied by a short descriptive message
- `git push`
  - Update the remote copy of the repo with the local changes and commits

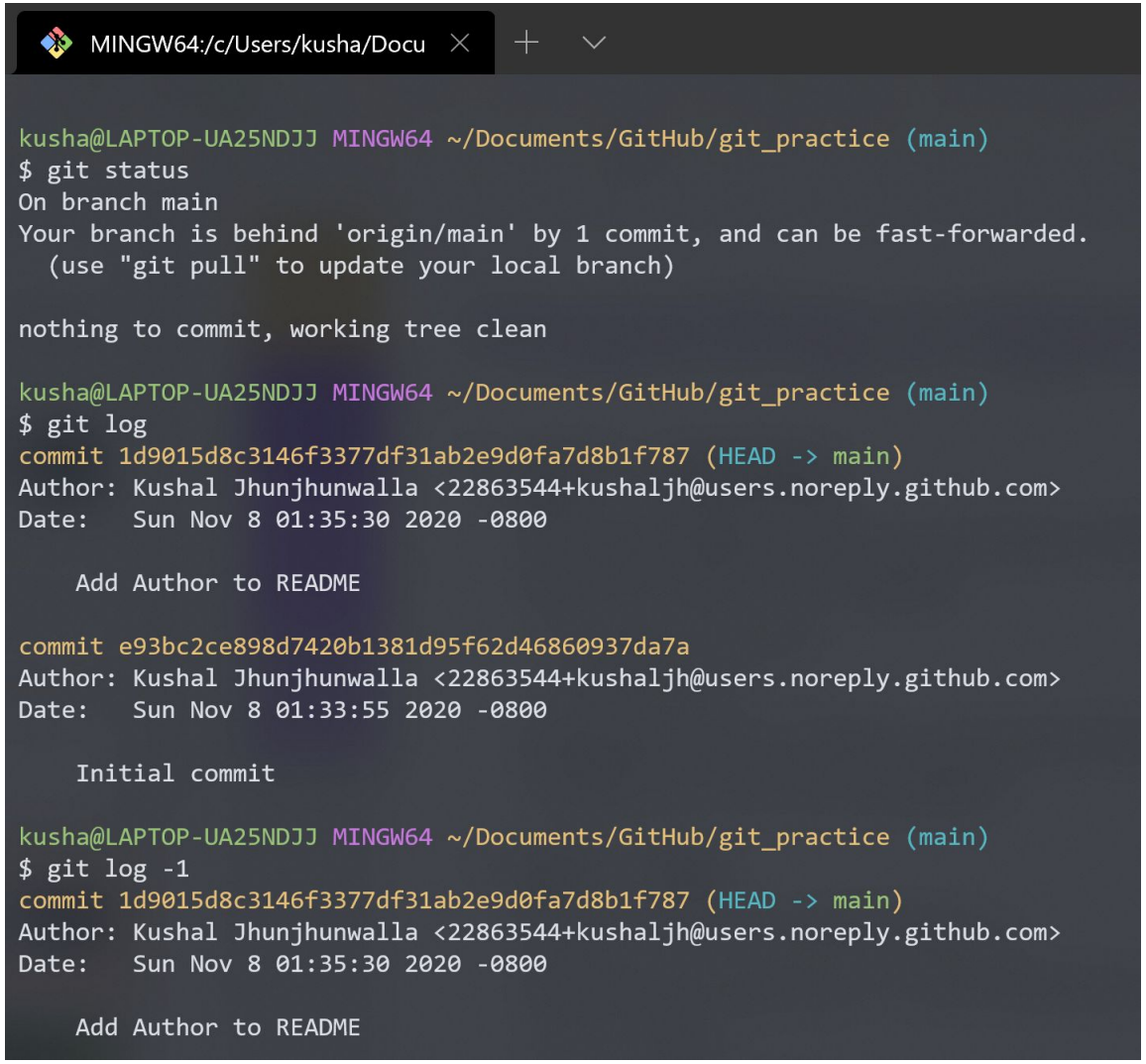


# Staging and committing overview



# Inspecting a repository

- `git status`
  - Lists the files which you have changed but not yet committed
    - Working directory
    - Staging area
  - Indicates how many commits have made but not yet pushed
- `git log`
  - Shows the commit history
  - `git log –graph --oneline`
    - Shows branch info as a graph



# Working with remote

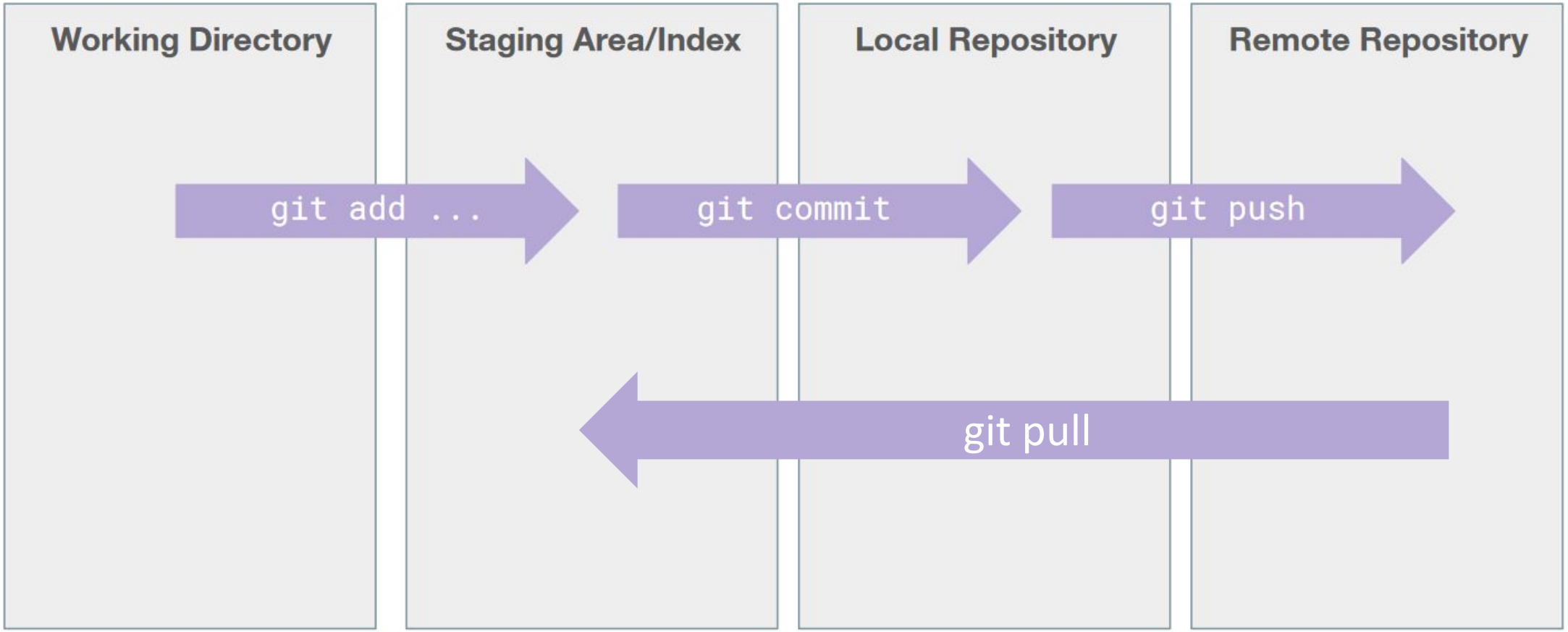
# `git` commands for interaction with remote

- **`git clone`**
  - Cloning is the process of creating a working copy of the remote or local repository by passing the following command.
  - **`git clone username@git_server_hostname:/path_of_repository`**

- **`git pull`**
  - If we have already cloned the repository and need to update local (only code) respect to the remote server
  - **`git pull origin main`**

- **`git fetch`**
  - Fetching is the process of updating (only git information) the local git structure and information from remote repository
  - **`git fetch`**

# Updating changes using pull




**NOTE**: There are way more git commands than what is listed here - this is a simplified model to get us started.

# What's next!

- Branching, checkout
- Merging, Merge (Pull) Requests
- Conflicts
- Interacting with a Git Server (GitHub / GitLab)

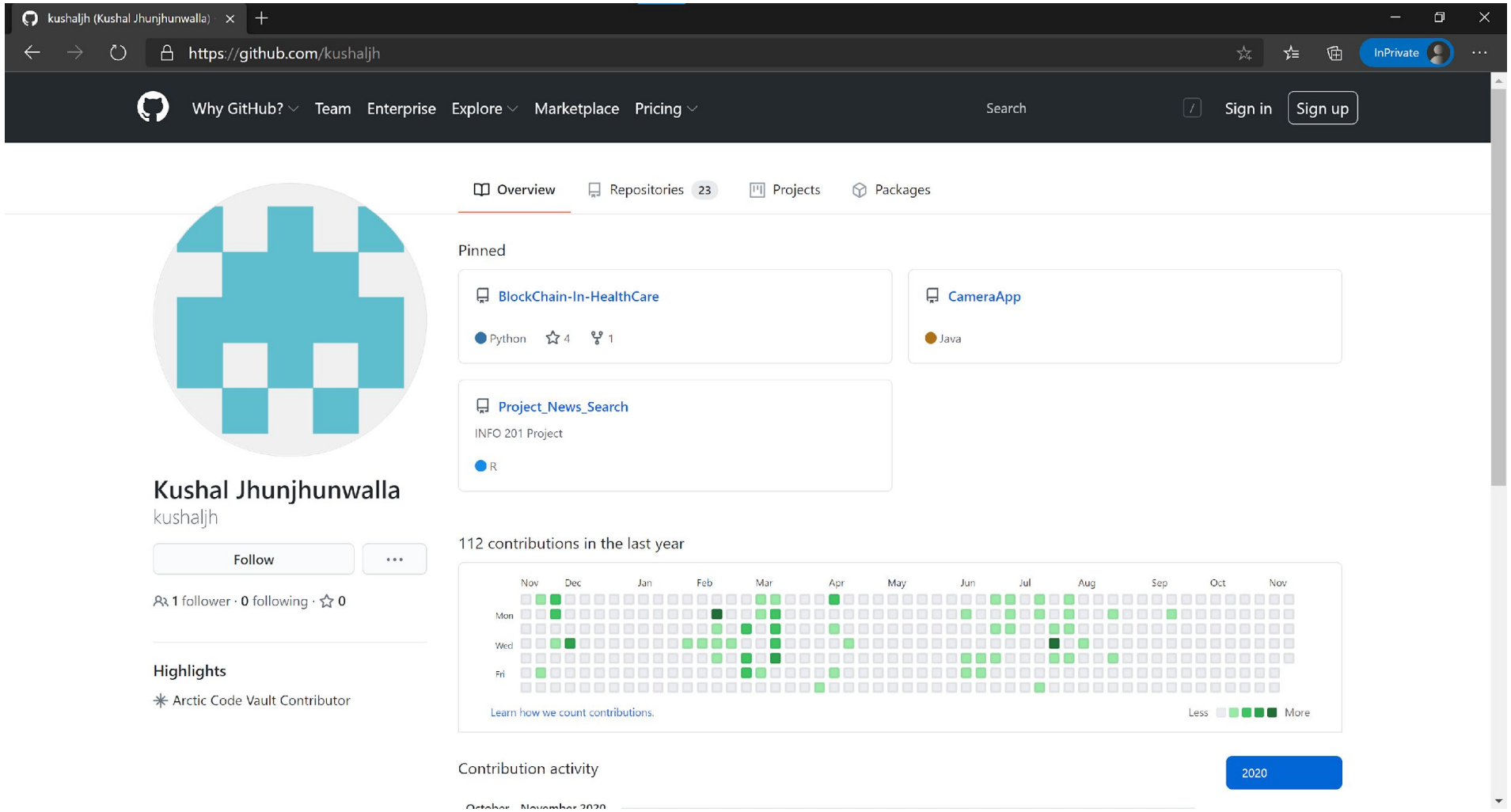
# Skipping files using `.gitignore`

- As a recap, generally we should not have these files in a git repo:
  - Object files (i.e. .class files, .o files) and executables
  - Huge media files (e.g. videos)
  - Credentials and system files (e.g. .DS_Store in Mac)
- Its tedious to mention all the files you want to add into a repo every time you add your changes (and ensure you skip the others)
- To skip above files you can create a file in the root of the git repo folder called `.gitignore`
- This file should contain patterns in file names you would like to skip being added
- https://github.com/github/gitignore

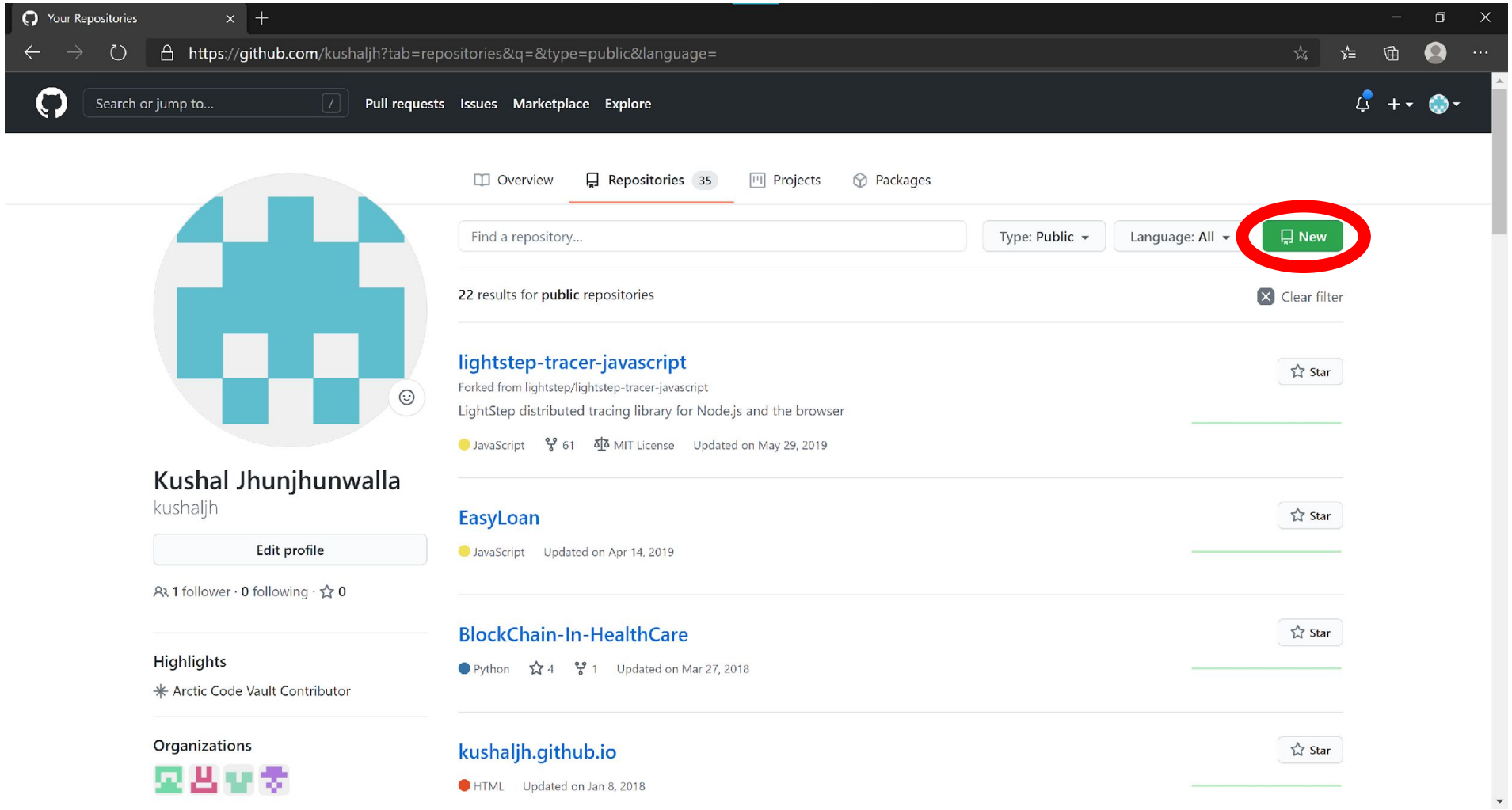
```
1  # OS generated files
2  **/.DS_Store
3
4  # Compiled source
5  *.com
6  *.class
7  *.dll
8  *.exe
9  *.o
10 *.so
11
12 # Logs and databases          You
13 *.log
14 *.sql
15 *.sqlite
16
```

# Working with a Git server | GitHub

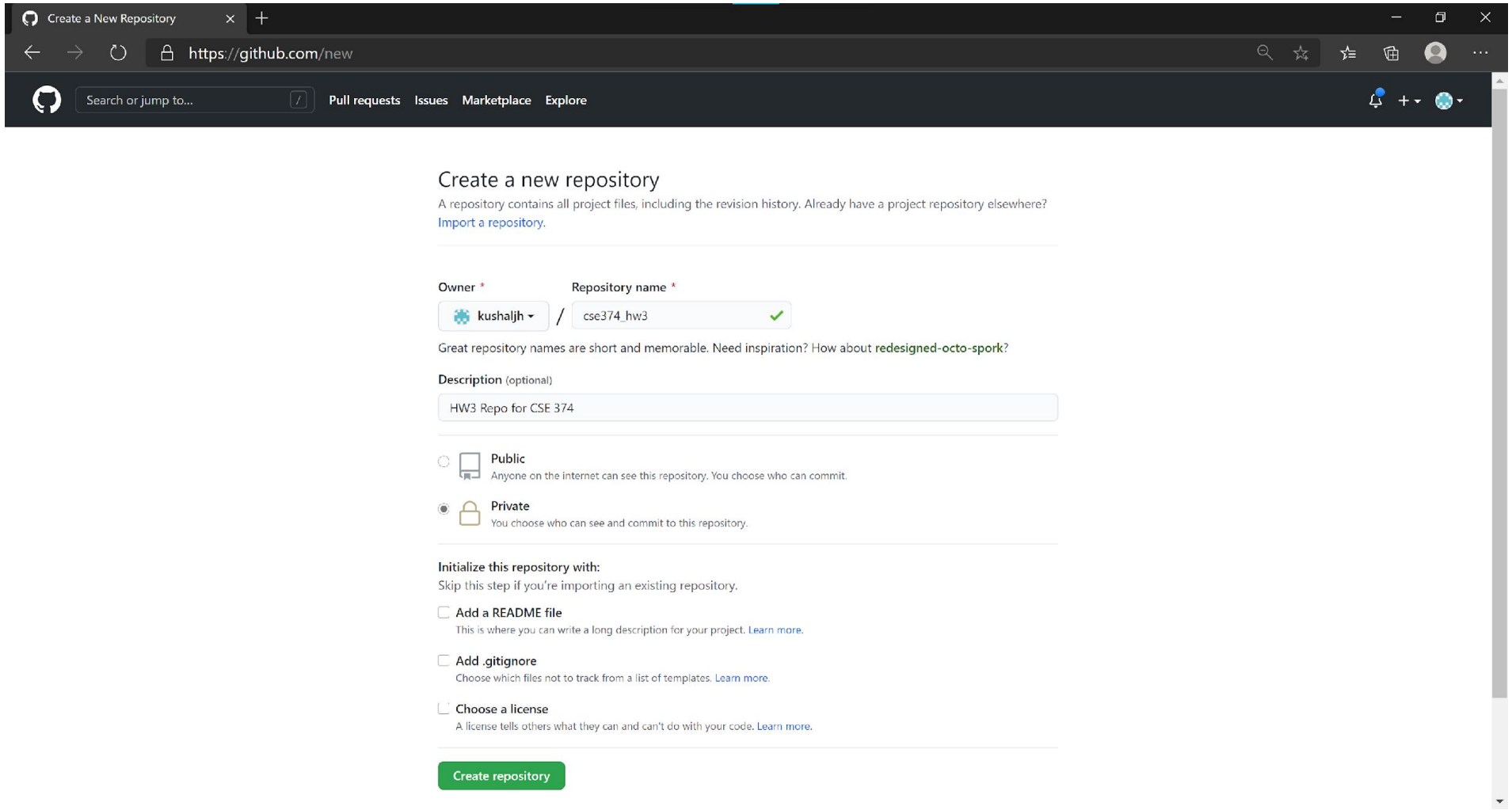
# GitHub profile



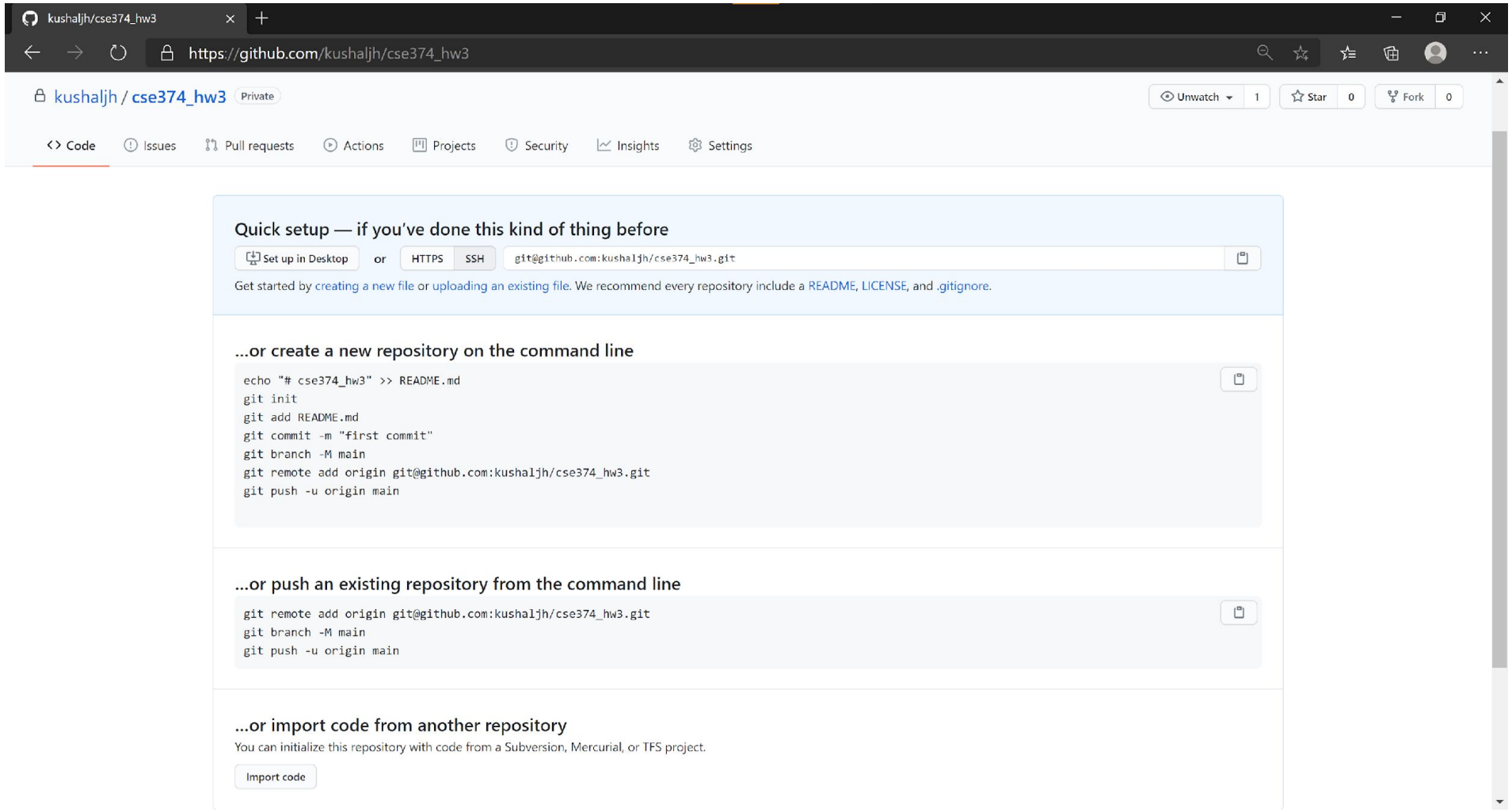
# Creating a new repo on GitHub



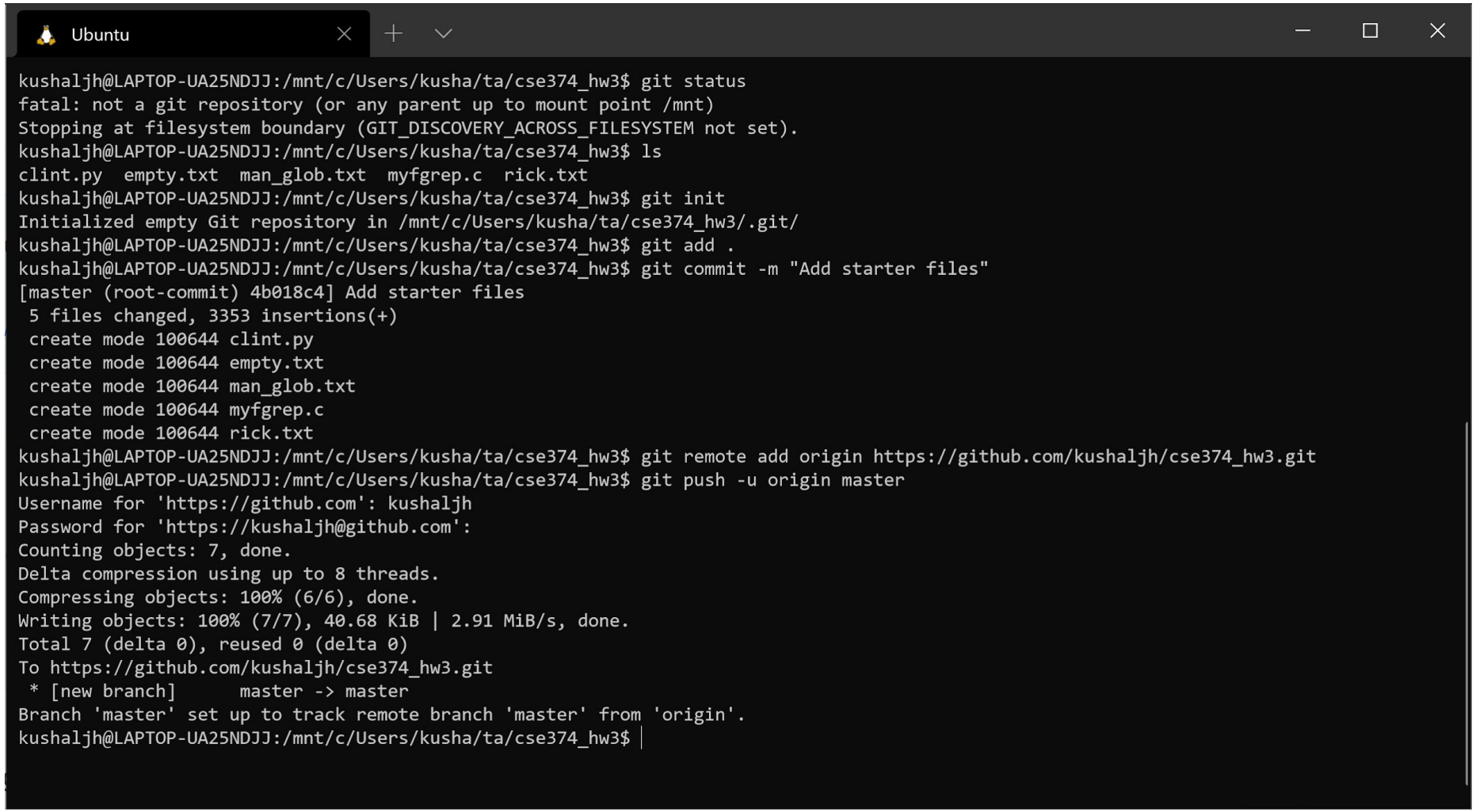
# Creating a new repo on GitHub



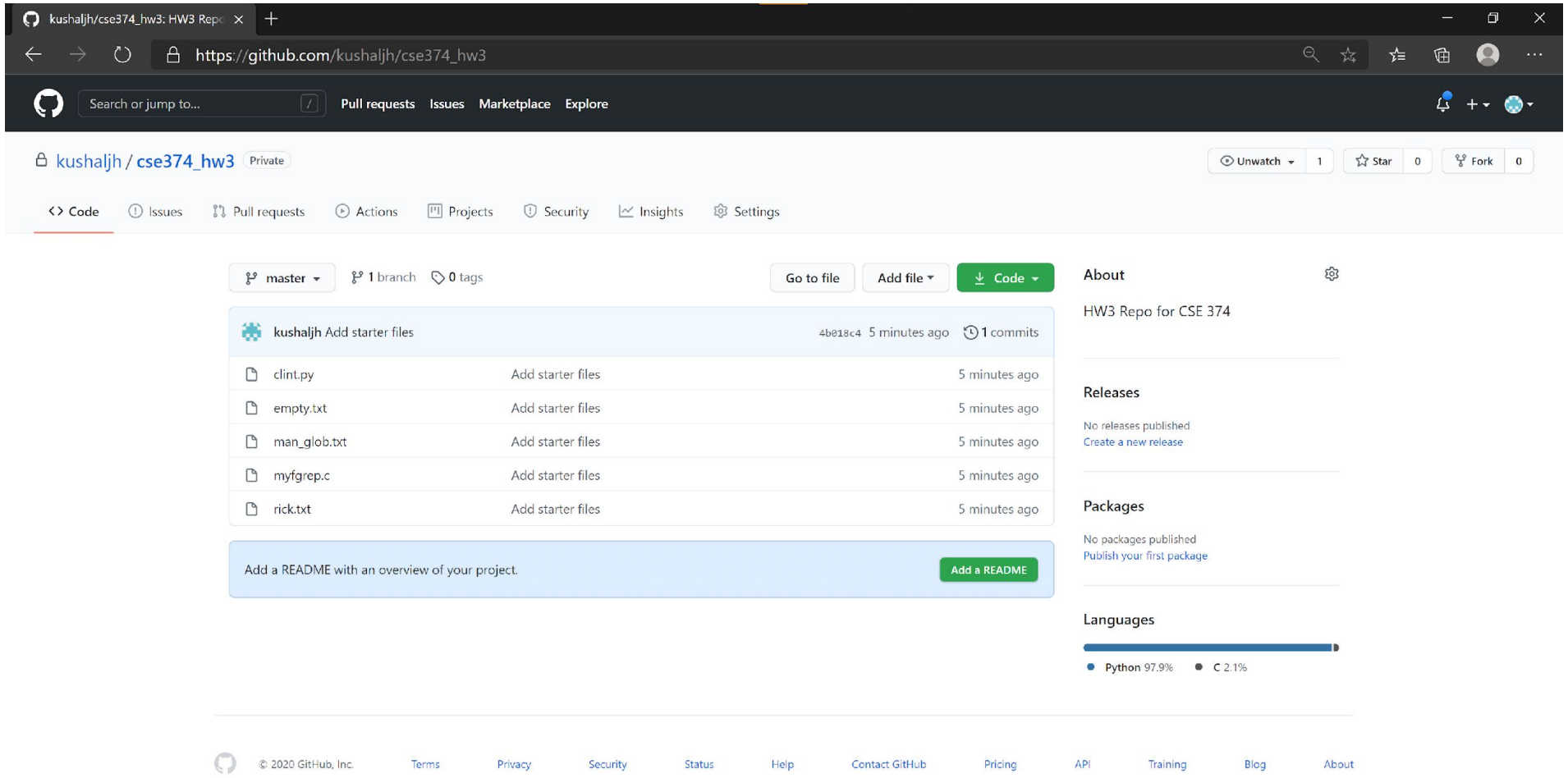
# Creating a new repo on GitHub



# Adding local files to the GitHub server



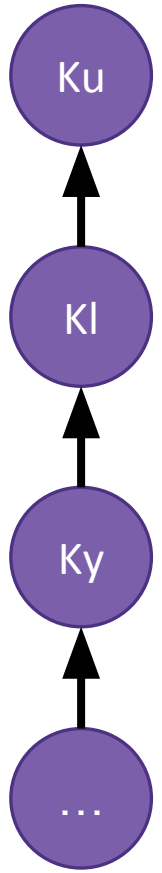
# Checking the changes on GitHub



# Branching

# Collaboration – the ideal case

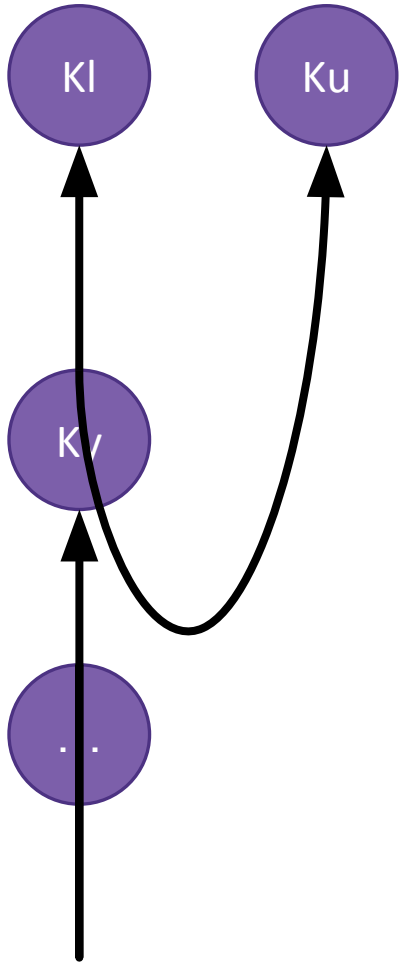
- 3 Collaborators
- Kasey makes a change and updates the repo (git push)
- Kalyani gets Kasey's changes (git pull), edits some files and updates the repo (git push)
- Kushal gets the latest changes that include both Kasey and Kalyani's changes (git pull)
- This goes on…



# But …

# Collaboration – the reality

- 3 Collaborators
- Kasey makes a change and updates the repo
- Kalyani and Kushal get Kasey's changes, edit some files and Kalyani updates the repo while Kushal is still working on some edits
- And here you see different people having different version histories



# Error!!

```
kushaljh@LAPTOP-UA25NDJJ:/mnt/d/kusha/OneDrive - UW/Documents/GitHub/git_practice$ git push
Username for 'https://github.com': kushaljh
Password for 'https://kushaljh@github.com':
To https://github.com/kushaljh/git_practice.git
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/kushaljh/git_practice.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
kushaljh@LAPTOP-UA25NDJJ:/mnt/d/kusha/OneDrive - UW/Documents/GitHub/git_practice$
```

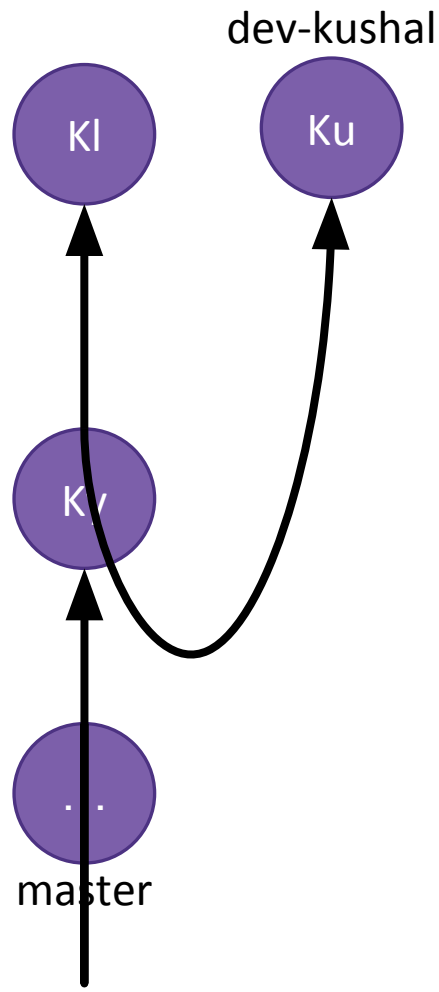
# Error!!

```
kushaljh@LAPTOP-UA25NDJJ:/mnt/d/kusha/OneDrive - UW/Documents/GitHub/git_practice$ git push
Username for 'https://github.com': kushaljh
Password for 'https://kushaljh@github.com':
To https://github.com/kushaljh/git_practice.git
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/kushaljh/git_practice.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
kushaljh@LAPTOP-UA25NDJJ:/mnt/d/kusha/OneDrive - UW/Documents/GitHub/git_practice$
```


ONE DOES NOT SIMPLY
PUSH TO MASTER

# Git Branches

- The scenario mentioned in the previous slides would be tough to solve if we used a single history across all collaborators.
- Instead you can start your own history at any point by “branching” out of the main history for a repo
- The master / main branch which is central history and hence the source of truth for the whole project
- We create new version histories based on the main branch and give each of these a branch name



# Git Branches

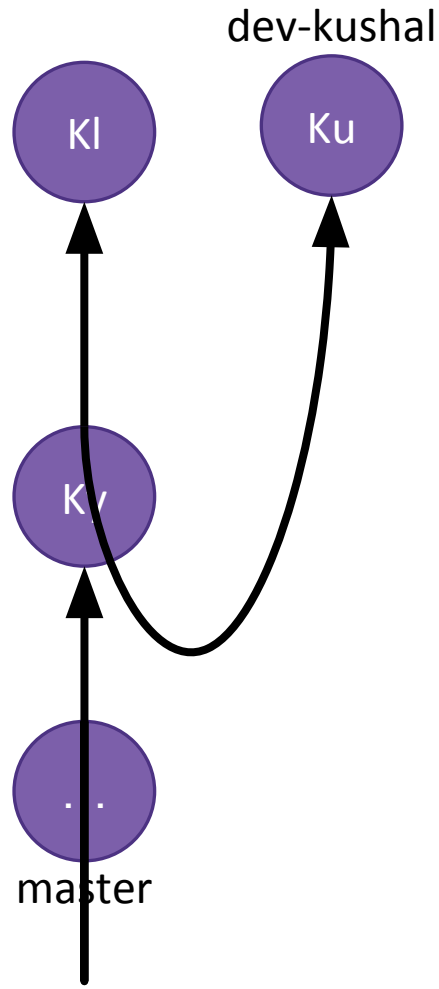
- Kushal's improved workflow using branches!

- Kushal creates a new branch called `dev-kushal` when she starts working

  - `git branch dev-kushal`
  - `git checkout dev-kushal`

    **OR**
  - `git checkout –b dev-kushal`

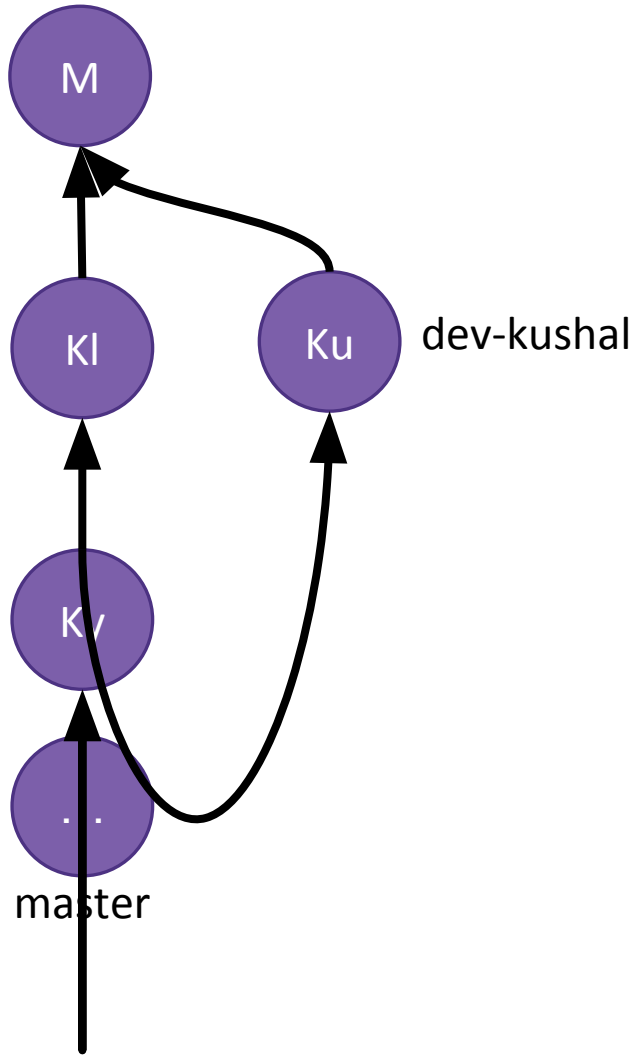
- Now she makes commits on this branch until she is ready to update master



# Git Merge

- After Kushal is done with her work, she would like the changes from her branch `dev-kushal` be reflected in the `master` branch
- Steps:
  - `git checkout master`
  - `git merge dev-Kushal`
- If there are no changes that are made in the same lines by Kalyani and Kushal there are no conflicting differences, and the merge is good to go
- If not, **WE HAVE A MERGE CONFLICT!**

- Note: this is not the only workflow used for merging



# PULL before you start working!!


GIT REPO
GIT PULL
GIT
MERGE CONFLICT

# Merge conflicts

- Merge conflicts happen when you are merging two branches that have a diff on the same line(s) of some file(s) in a repo
- When running merge we run into an error in such a scenario

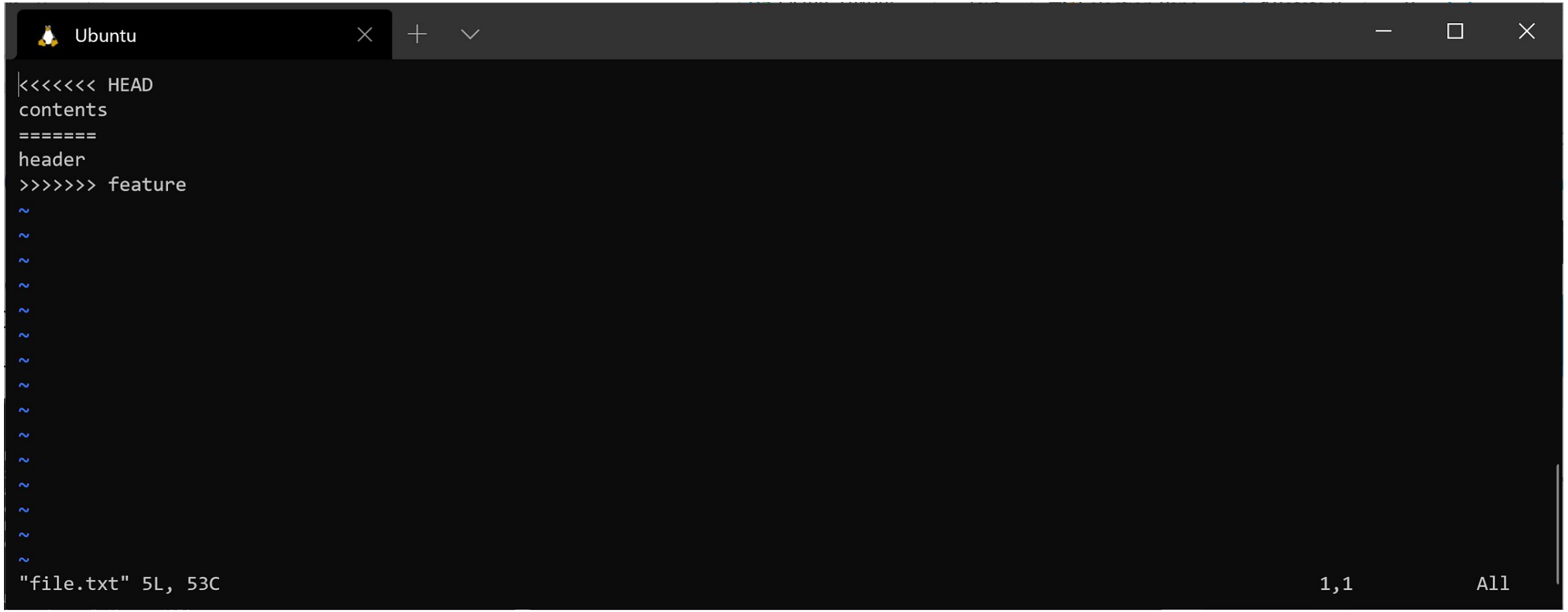
Ubuntu

```
kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$ git merge feature
Auto-merging file.txt
CONFLICT (add/add): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$ git status
On branch master
Your branch is up to date with 'origin/master'.

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both added:      file.txt

no changes added to commit (use "git add" and/or "git commit -a")
kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$
```

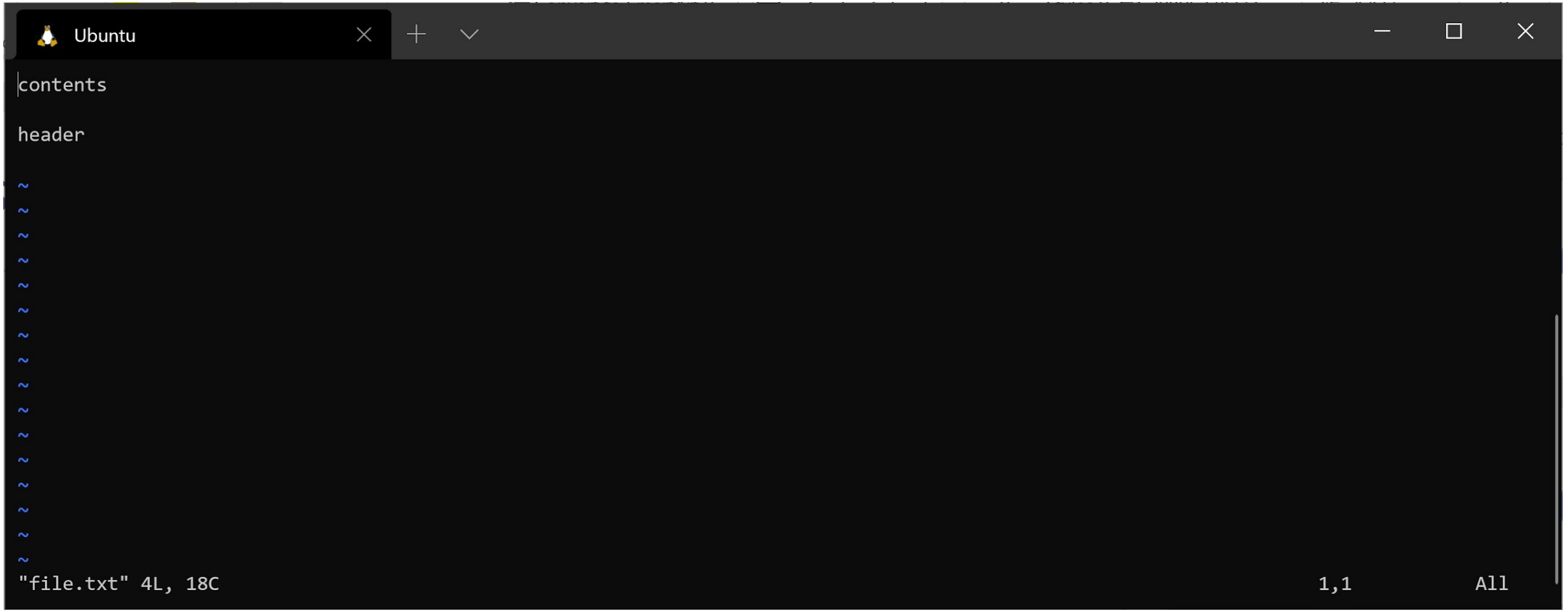
Ubuntu

```
<<<<<<< HEAD
contents
=======
header
>>>>>>> feature
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"file.txt" 5L, 53C
```

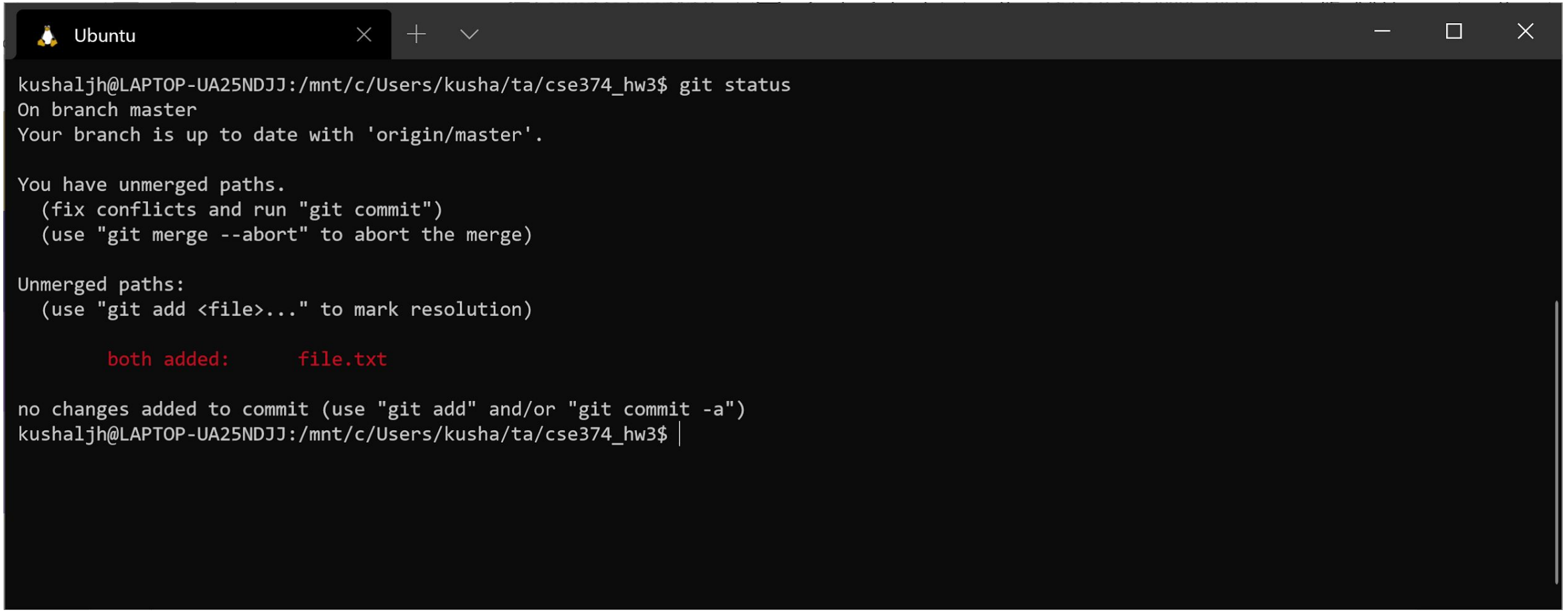
# Looking for the conflict



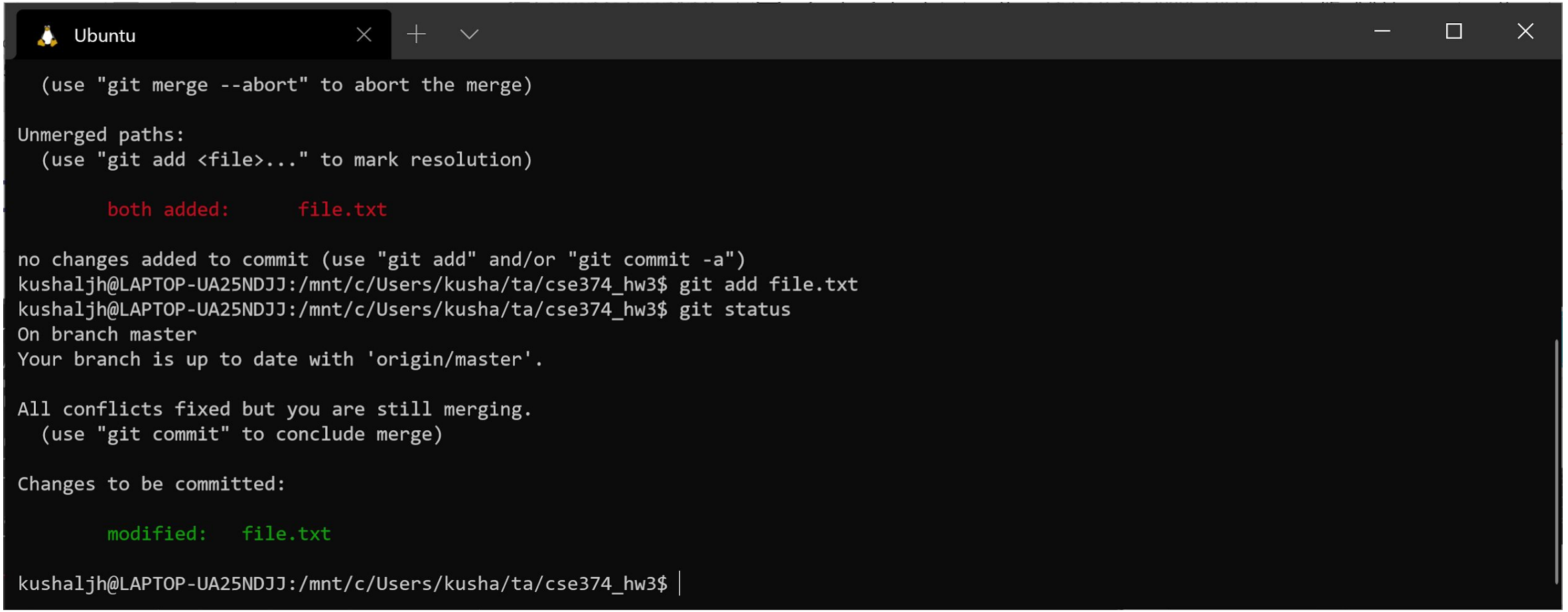
# Resolving the conflict



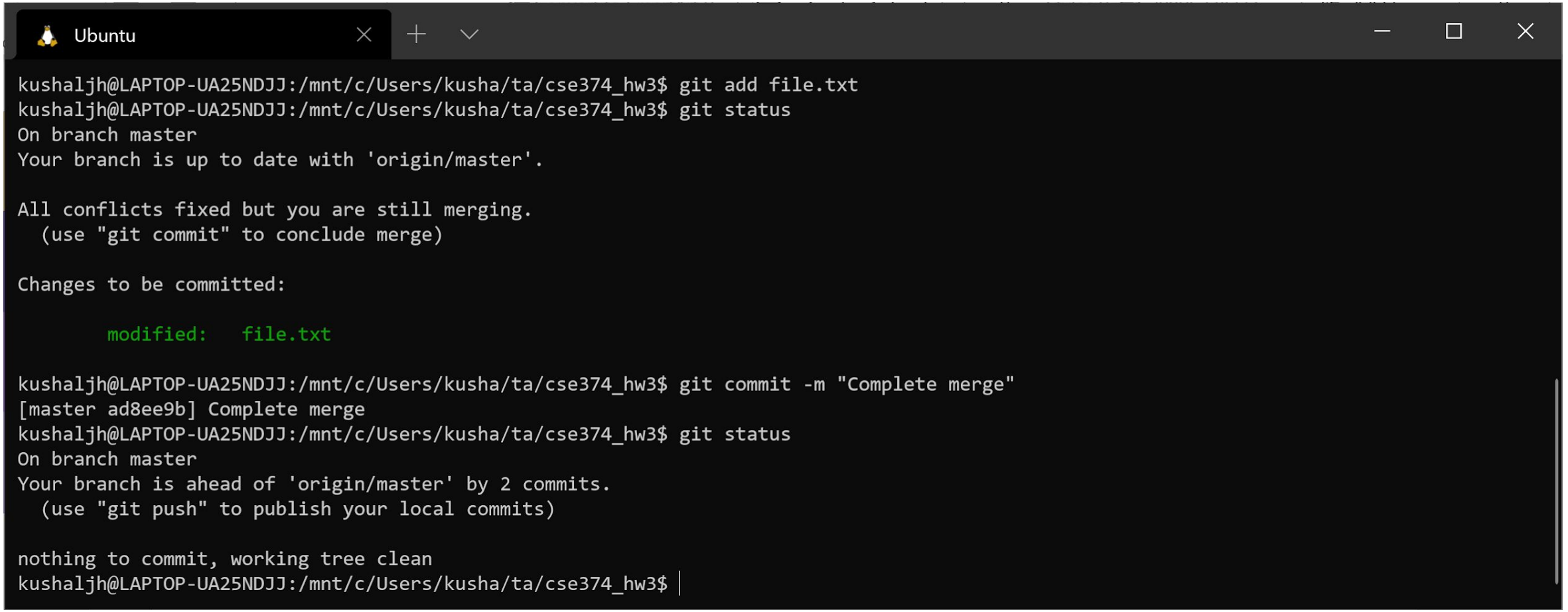
# Checking the status



# Adding resolved file and saving



# Conflict resolved!!

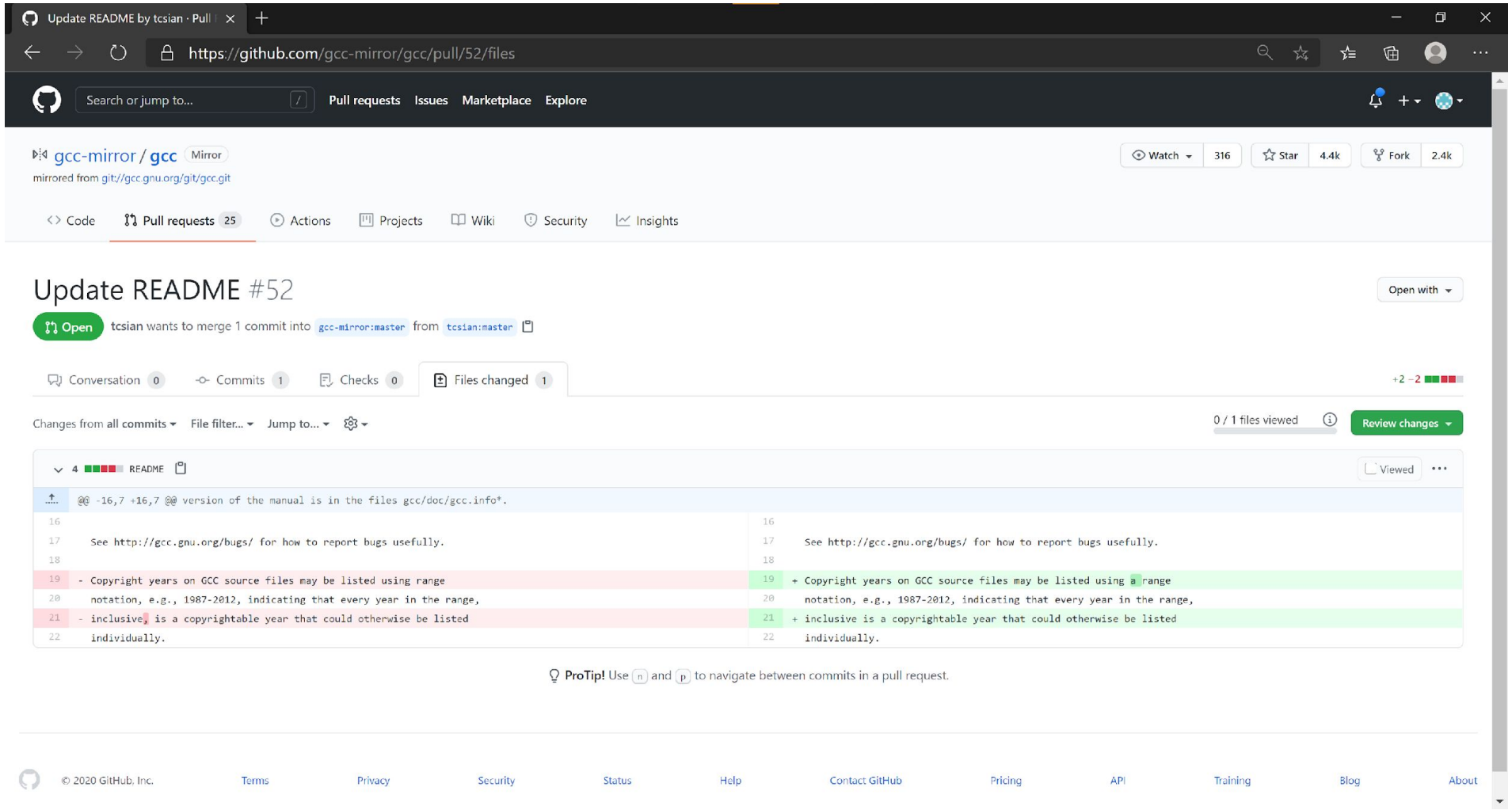


# Merge (Pull) Requests

- This was nice to know but we generally do not merge branches into `master` locally
- We use a Git server – GitHub / GitLab – to create a request to merge a feature branch into master
- **We must ensure that everyone on the project agrees to what is present in the `master` branch as this is our source of truth that everyone shares**

- The workflow is as follows:
  - Create a new branch
  - Add and commit changes to the branch and push it to GitHub
  - Create a Pull Request on GitHub
  - Other collaborators look at the PR and leave their feedback (this is generally called a **Code Review**)
  - Fix issues and then merge

# Sample Pull Request



# Working with branches & more | DEMO


1. git commit

2. git push

3. git out!

# Useful resources

- Try Git (resources and tutorial)
- The Git Cheat Sheet
- Stack Overflow's definitive guide for beginners
- When you are terribly stuck with git
  - DO NOT PANIC! Even experienced developers get stuck with git issues
  - https://ohshitgit.com/
  - https://stackoverflow.com/questions/tagged/git