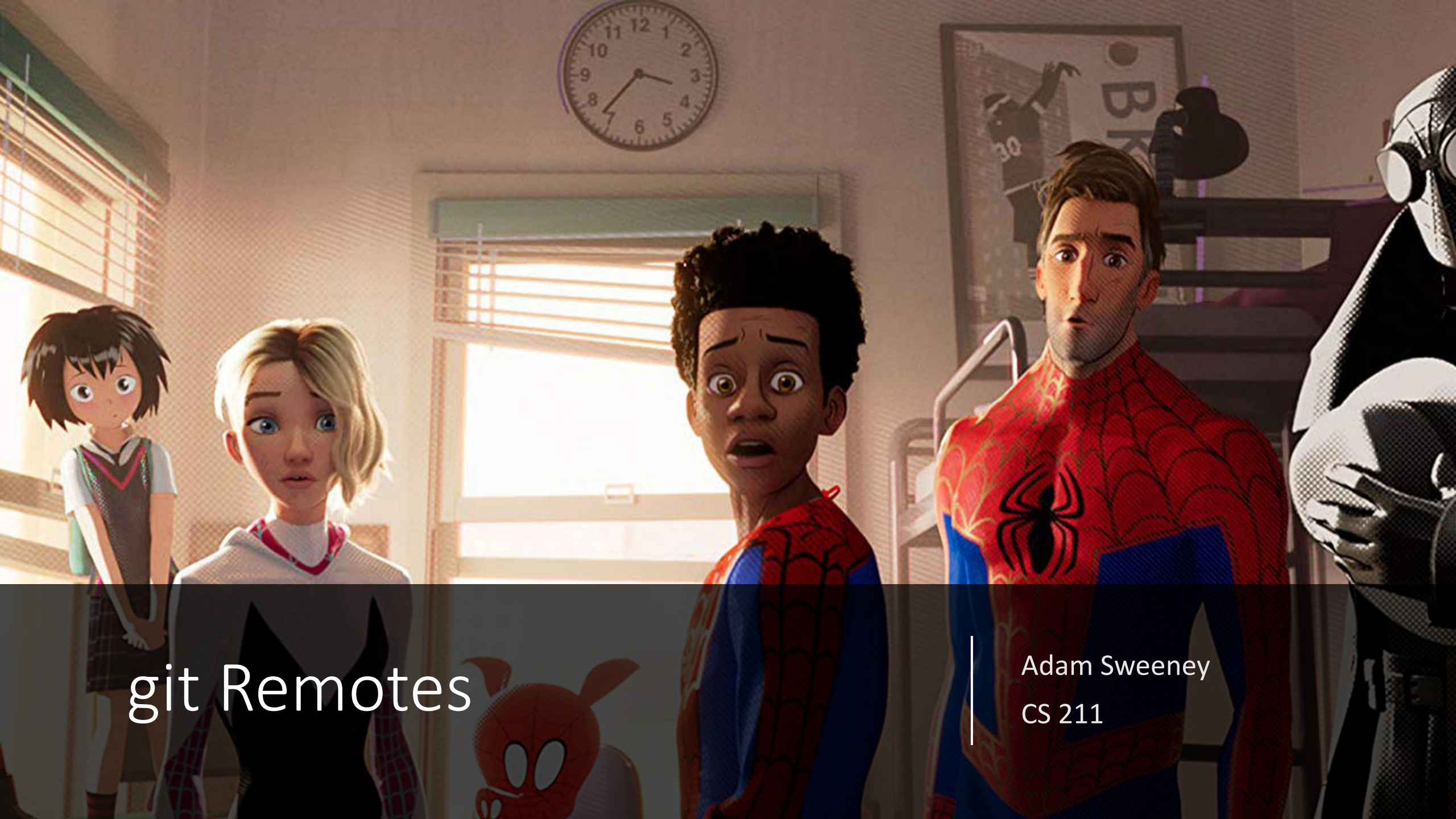

git Remotes

Adam Sweeney
CS 211

Introduction

- git is a tool that controls our software versioning
- When collaborating, the code usually needs to live on a network location
  - Called remotes
- Let’s learn about using git with remotes

Agenda

- Adding a remote to an existing project
- Starting a project with a remote
- Collaborating

Adding a Remote

What is a Remote?

- A common repository that allows team members to share their changes with each other
- Common remotes that we think of include GitHub, Gitlab, BitBucket
- Remotes can also be hosted internally
  - GitHub, Gitlab, BitBucket, Gitea, Gogs, etc.

Connecting Our Project to a Remote

- Log in to your remote and create an empty repository
  - Fairly simple, just follow steps
- Remember the address
  - https://gitlab.com/username/repo_name
- Two scenarios to consider
  - An existing folder with code
  - An existing local repository

An Existing Folder with Code

- `cd existing_folder`
- `git init`
- `git remote add origin https://gitlab.com/username/repo_name.git`
  - “origin” can be replaced with a different name, but it is standard practice that the main remote is called origin
- `git add .`
- `git commit -m "Initial commit”`
- `git push -u origin master`
  - ”-u” option or “--set-upstream”, tells git that this is going to be our main remote

An Existing Local Repository

- `cd existing_repo`
- `git remote rename origin old-origin`
  - Only needed if switching remotes
- `git remote add origin https://gitlab.com/username/repo_name.git`
- `git push -u origin --all`
- `git push -u origin --tags`
  - Only needed if you've been using tags

Starting a Project with a Remote

The Simpler Path

- Log in to your remote and create a new repository
  - Do initialize it with a README now
- On your system navigate to where you want the repository to be
  - Don’t create a folder for the repo
- `git clone https://gitlab.com/username/repo_name [folder_name]`
  - folder_name is optional, by default git will name it the same as your repo
- The remote has been set up for you and called origin

Collaboration

Teamwork Makes the Dream Work

- Working together on a project requires more than correct tooling
- Communication is key
- Software development patterns like AGILE become very important
  - Intro to Software Engineering covers this
- We are concerned with the technical challenges with respect to git
  - Steps considered here are still simplified
  - Practice makes perfect

Working with git Remotes

- It is important to stay in sync with the remote
- `git pull [branch]`
  - Combines two commands, git fetch (download updates) and git merge (transfer files)
  - Overwrites your working directory
- `git push [remote_name] [branch_name]`
  - Uploads your changes to the remote
  - If remote_name is not specified, defaults to origin
  - If branch_name is not specified, may not work
  - You should always specify remote_name and branch_name

Workflow

- Update your local repository
  - `git pull`
- Create a new branch for your work
  - `git branch <branch_name> && git checkout <branch_name>`
  - Or `git checkout –b <branch_name>`
- Do your work
  - You don’t need to be committing and pushing at this point
- When the work is done
  - `git commit -m “MESSAGE HERE”`
- `git pull` (if the fix took more than a day)
- `git push <remote_name> <branch_name>`

Getting Your Push Accepted

- Log in to the remote
- Find your branch
- Perform a merge request
  - At this point, code reviewers will look over your code, suggest changes, ask you to resolve conflicts, etc.
- Code is merged into the repository
- After your code has been merged, you can delete your local branch
  - `git checkout <branch_name> && git branch -d <submitted_branch>`

Merge Conflicts

Code Always Changes

- It’s likely that the master branch will be updated while you work on your branch
- This means that there is a possibility that the same line of code gets altered in two different branches
- When attempting to merge a branch where a line of code was altered in both branches, we get a merge conflict

How git Handles Conflict

- It doesn’t
  - It identifies the conflict and creates a set up for you to handle it

What git Does

- When it recognizes it can’t do a clean merge, git notifies you and then tells you how to fix it.
  - git generally does this when there are errors
- `CONFLICT (content): Merge conflict in FILENAME Automatic merge failed; fix conflicts and then commit the result.`
- It also changes the file in the area of the conflict

Resolving Merge Conflicts

```
<<<<<<< HEAD
```
(Receiving branch)

```
Hello you.
=======
Goodbye.
>>>>>>> bad_merge
```
(Merging branch)

- The highlighted text is how git changes a file
- The markers make identifying the conflicting area(s) easier to find
- To resolve, delete all but the line you wish to keep
- Save, add, and commit

Review

- Adding a remote to an existing project
- Starting a project with a remote
- Collaborating
- Resolving merge conflicts