Collaboration

Hamish Gibbs

Collaboration

- Most programming isn’t solo.
- In companies / research, you are often working on a small part of a larger project.
- To collaborate - you need to share code!

Version control

- To collaborate, we need:
  - A place to store a shared version of our code.
  - A way to track changes to different parts of the code.
- Solution:
  - A “repository” to store our code
  - A version control system to track changes to the code

Local vs. Remote code

- A repository stores the code for a specific project.
- There are two different types of repository:
  - **Local:**
    - *Think: a file folder on your computer.*
  - **Remote:**
    - *Think: a GitHub repository*

Local vs. Remote code

- With version control, I want my **local** changes to be reflected in the **remote** repository.

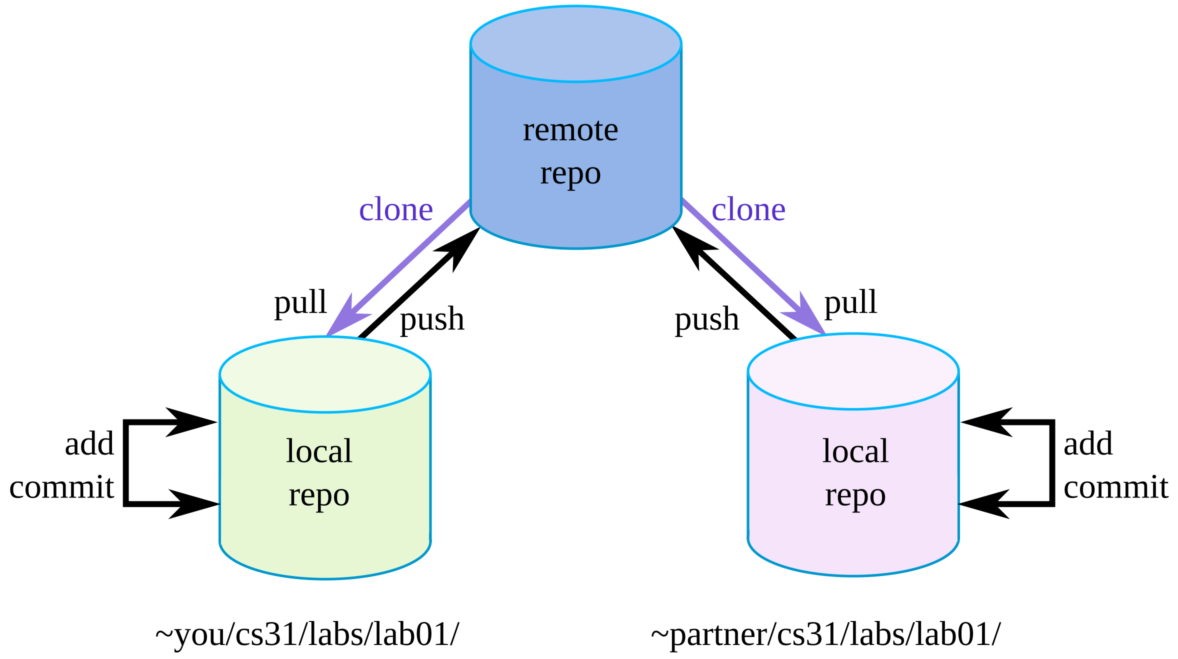

Image credit: Swarthmore Computer Science

git and GitHub

- **git** is an open source version control system.
  - Purpose: recording and reconciling changes to code.
- **GitHub** is a place to store **remote** repositories.

Why use git?

- The best example:
  - Here is the repository for this course
- You can:
  - Look at the history of changes: here
  - Go back to an earlier version of the course materials: here

Why use git?

- `git` lets you:
  - Save a version of your code online.
  - Delete / modify code without losing anything.
- I recommend:
  - Build a portfolio by saving any programming you do for your courses in GitHub.
  - This can show off your programming experience for jobs / graduate school.

Google docs

- `git` is kind of like Google Docs.
  - I make a change to a document.
  - You make changes to the same document.
  - Our changes are combined together.
- **Except:** `git` is *very manual*.

Version control

- With `git` you need to be explicit about:
  - Saving changes (called ‘committing’).
  - ‘pushing’ **local** changes to the **remote** repository.
  - ‘pulling’ changes from the **remote** repository.
  - ‘merging’ changes together.
- ***Good question:*** *Why does this have to be so explicit?*

Aims: today

- This is a high-level introduction to `git` but it is sufficient for today.
- We want to:
  - Create a place where we can compare everyone’s solutions to the Challenge.
  - Let everyone contribute their **local** code to this **remote** repository.

Aims: today

1. Clone the shared repository:

   ```
   git clone [repo-url].git
   ```

2. Copy your code to the cloned repository.

Aims: today

3. Create your own **branch**.

```
git checkout -b [my-branch]
```

4. Add & commit your changes.

```
git add [myfile].py
git commit -m "Adding my file!"
```

Aims: today

5. Push your code to the **remote** repository:

```
git push
```

6. Pull other changes from the **remote** repository:

```
git push
```

Other useful commands

- After `git add` but before `git commit`:
  - Inspect which files have been created / modified / deleted:

    ```
    git status
    ```

  - Inspect changes to the code since the last commit:

    ```
    git diff --cached
    ```

Diving deeper into `git`

- There is more to `git`:
  - **Branching**: creating different versions of the same code base.
  - **Merging**: combining different branches back into the `main` branch.
- And more to GitHub:
  - Issue / project tracking
  - Automated actions

Tip

- Github’s Education Benefits give you access to a lot of free stuff!
    - GitHub Copilot
    - GitHub Copilot Chat
    - Free web hosting