VERSION CONTROL

Review: Version Control

- Why do we need version control?
  - What is it good for?
- What are examples of version control systems?
- Using git
  - How do we "get" code from the repository initially?
  - How do we put our code into the public version of the code?
  - How do we get code from the public version of the code?

Backups and Coordination Issues

- Maybe you wrote a paper and had several versions
  - Good practice to iterate over it!
  - Keep track of versions using names, e.g., `paper1.pdf, paper.draft.pdf, paper.outline.pdf, paper.mar7.pdf`
- Coordinate a group
  - One person's account has *the* version
  - Make conflicting changes to files
    - Figure out fix, Merge files

Version Control

- Backup and Restore
  - Files are saved as they are edited
  - Revert to a specific version/checkpoint
- Synchronization
  - Lets people share files
  - Stay up-to-date with the latest version
- Track changes to code
  - Save comments explaining why change happened
  - Stored in the VCS, not the file
  - Track how, why a file evolves over time
- Track Ownership
  - Tags every change with the name of the person who made it

Version Control

- Short-term undo
  - Messed up a file? Go back to the last **good** version
- Long-term undo
  - Created a bug a year ago? Jump back to see change you made.
- Sandboxing
  - Making a big change? Make temporary changes in isolated area, test, work out kinks before "checking in" your changes
- Branching and merging
  - Branch a copy of your code into a separate area, modify it in isolation (tracking changes separately)
  - Later, merge work into common area.

Common Git Commands

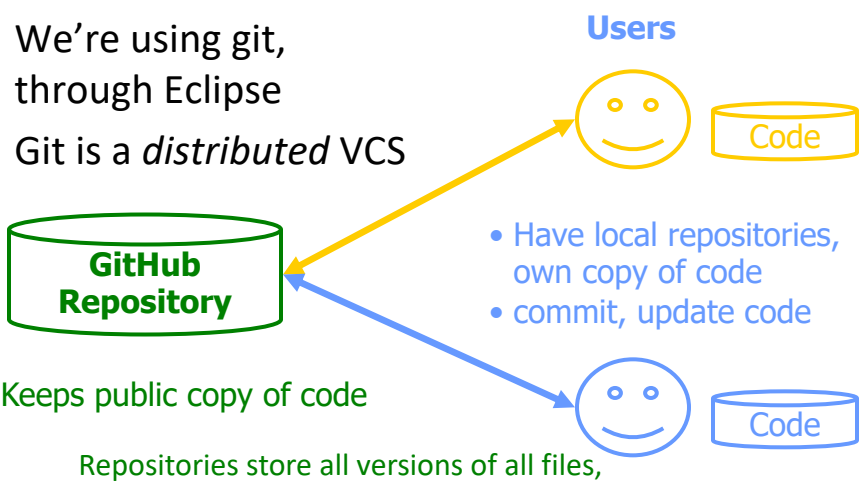
Command	What it does
add [file]	Adds the file to the staging area
commit	Commits all the staged files (locally)
push	Push all your changes to the remote → You need your code to be pushed so that I can see it.
branch	List all local branches
branch [name]	Creates a new branch with that name

Typical, Simple Workflow

- Clone the project
- Create a new branch, named by what you're working on
  - Switch to that branch
- Update files
- When you've hit a good checkpoint, add the changed files to the "staging area" and then commit those files
  - Add a descriptive comment about what you've done.
- Switch back to the "main" branch and merge in the branch you were working on
- If you are ready to put your code on GitHub, push

Using Version Control

- We're using git, through Eclipse
- Git is a *distributed* VCS

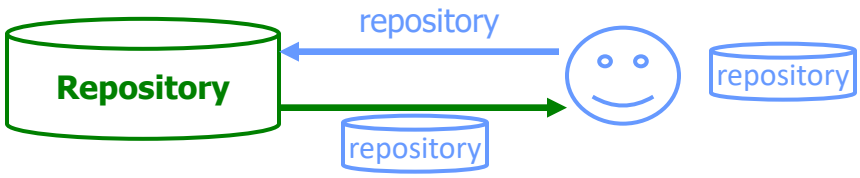


- Keeps public copy of code

Repositories store all versions of all files, comments about changes ("commit messages", who made changes

Using Version Control: `clone`

- To start, need to **clone** the repository



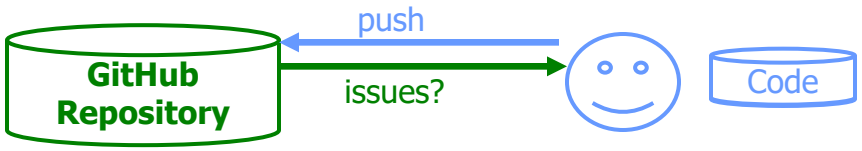
Using Version Control: `commit`

- After you make changes that you want to document, **commit** your version
  - Include comments about what changes you made and why

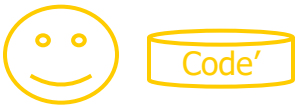
- Updates each modified file
- Records comments with updated files

Using Version Control: **push**

- After you make changes that you want others to see, **push** your version
  - Comments → from your previous commits



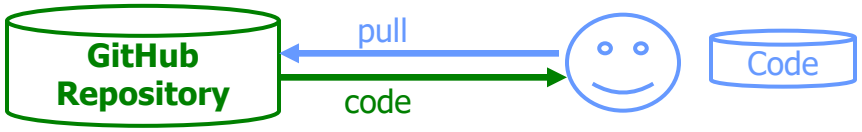
- Updates each modified file
- Records comments with updated files



Other people's code doesn't change

Using Version Control: `pull`

- To see the *current* version of the code in the remote repository, **pull**
  - Resolve conflicts



Using Version Control: `add, delete`

- You need to **add** and **delete** files and directories to the staging area, then **commit**

- Create new records for added files
- Close records for deleted files
  - Files not deleted from repository

- Add, delete files and directories
- Commit

Version Control Advice

- Does not eliminate need for communication
  - Process becomes much more difficult if developers do not communicate
- Write descriptive comments when you commit so your team members (and you!) know what you did and why
- **Push** only after you've tested code and you're fairly sure it works