Objectives

- Design Wrap Up
- Introduction to Version Control

Open Poll Everywhere

Fill in the blank: I found ~80% of the terms to be _____ to define/identify

1

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Design Questions

1. `turn` is an *instance* variable of the `Game` class.
   - Is it better design for `turn` to be a local, instance, or class variable? Justify your answer.
2. `user_input` is a *local* variable in the `getInput` method of the `ConnectFour` class.
   - Is it better design for `user_input` to be a local, instance, or class variable? Justify your answer.
3. `RANKS` is a *class* variable of the `Card` class.
   - Is it better design for `RANKS` to be a local, instance, or class variable? Justify your answer.
4. `tokens` is an *instance* variable of the `ConnectFour` class.
   - Is it better design for `tokens` to be a local, instance, or class variable? Justify your answer.
5. `Player` is a class in `war.py`.
   - Is it better design for the `Player` class to be defined in `war.py` or in `game.py`? Justify your answer.
6. `War`'s `step` method takes as a parameter `dummyInput`. What purpose does it serve?

What questions should you ask to determine if some state should be local, instance, or class variable?

Design Answers, in Brief

1. `turn` should be an *instance* variable of the `Game` class.
   - Each object of the Game class should have its own `turn` variable.
2. `user_input` should be a *local* variable in the `getInput` method of the `ConnectFour` class.
   - It is not useful to any other method of the class; it is just for that method.
3. `RANKS` should be a *class* variable of the `Card` class.
   - There should only be one RANKS object for *all* Card objects.
4. `tokens` should be a *class* variable of the `ConnectFour` class.
   - There only needs to be one copy of `tokens` for *all* ConnectFour objects
5. `Player` could be in game.py as an abstract parent *or* as a class in `war.py`.
   - Either answer can be justified.

Design Answers, in Brief

`War`'s `step` method takes as a parameter `dummyInput`.
What purpose does `dummyInput` serve?

From Game class:

```
def main(self):
    while not self.isGameOver():
        print(self)

        self.step(self.getInput())

    print("\nGame Over!\n")
    print(self)
```

`pass` in War

Conclusion:

- Use of `dummyInput` is an indication that something should be designed better

What kind of variable should this be?

- Does this variable need to persist?
  - If not → *local*
- Does each object of the class need its own variable/state? → *Instance*
- Or, can all objects of the class share one variable? → *Class*

VERSION CONTROL: GIT

Motivating Version Control

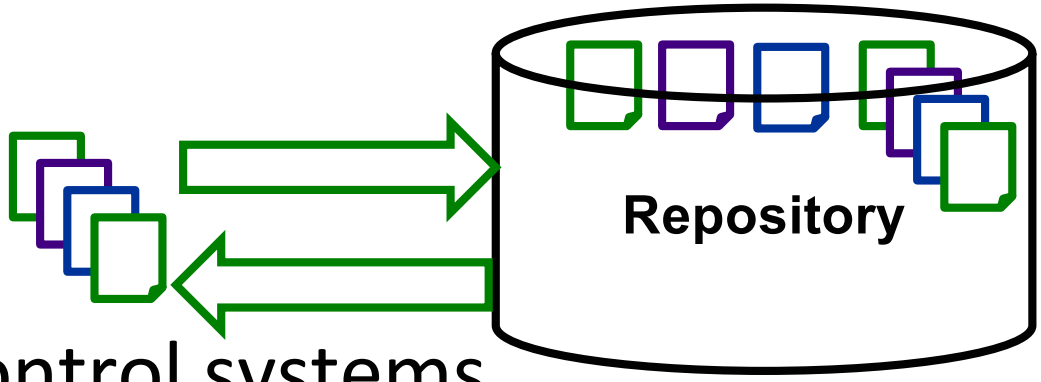
- Do you name your documents like the following?
  - Paper-final.pdf
  - Paper-final2.pdf
  - Paper-final_real.pdf
  - Paper-FINAL-final.pdf
- Do you sometimes break your code really badly and want to go back to a previous state?
- Do you forget why you made a code change?
- Do you want to just try something out and, if it doesn't work, revert back?

Motivating Version Control: Collaboration

- How do you share documents with teammates?
  - Emailing with conflicting updates?
  - Google Docs/Box Notes → not meant for code
  - Merging contributions
  - Who has the *real* version of the documents?

Version Control Systems Can Help!

- Main idea: ***repository*** holds the code and all changes to it
  - Need to push and pull code to and from the repository



- Centralized version control systems
  - E.g., CVS, Subversion, …
- Distributed version control systems
  - E.g., Git, Mercurial, …

Git & GitHub

- We're going to use Git
  - Distributed version control system
- Our repositories will be hosted by GitHub
  - How you'll get code from me
  - How you'll submit assignments

GitHub's Octocat

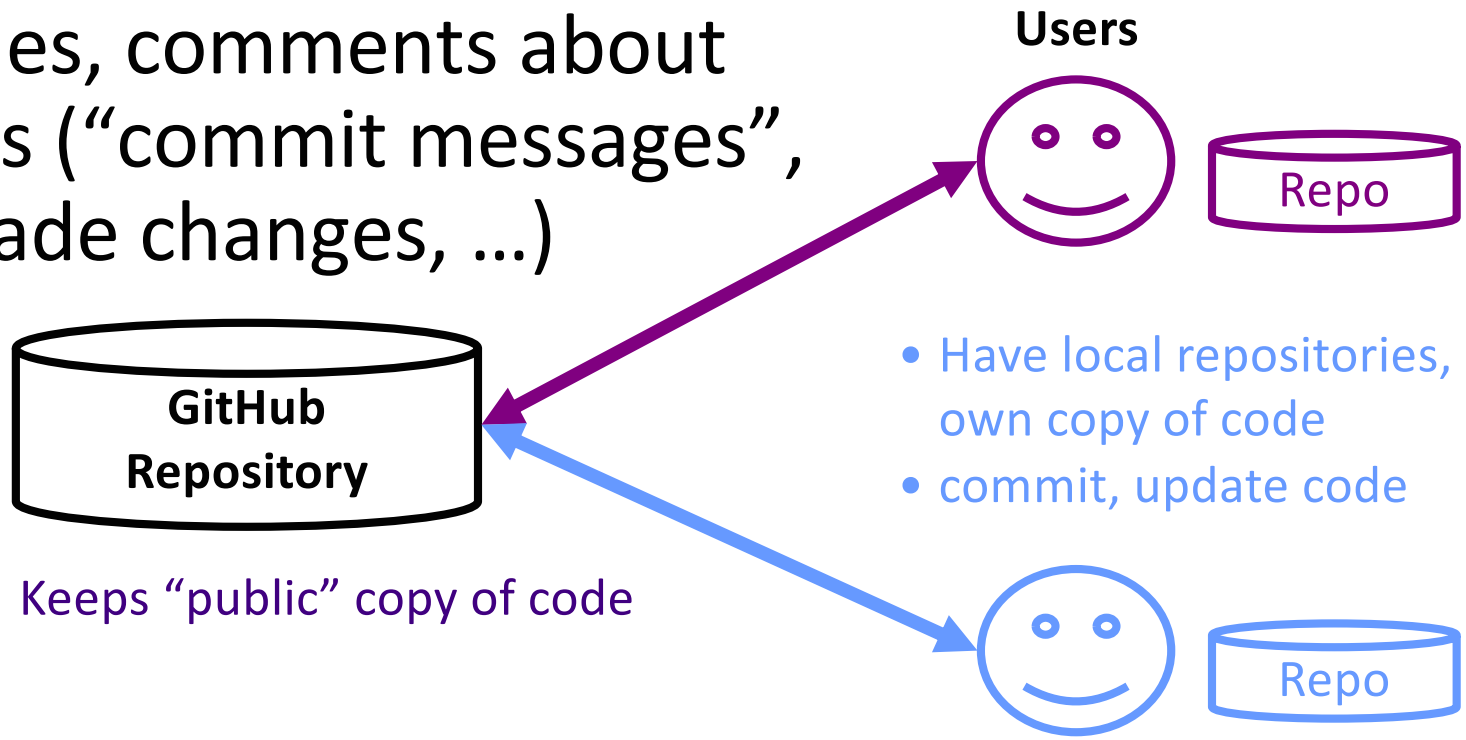
Common Git Commands

Command	What it does
clone	Clones a repository – sets up your repository so that you can coordinate
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 named *name*
checkout <*name*>	Switches to the branch named *name*

https://xkcd.com/1597/

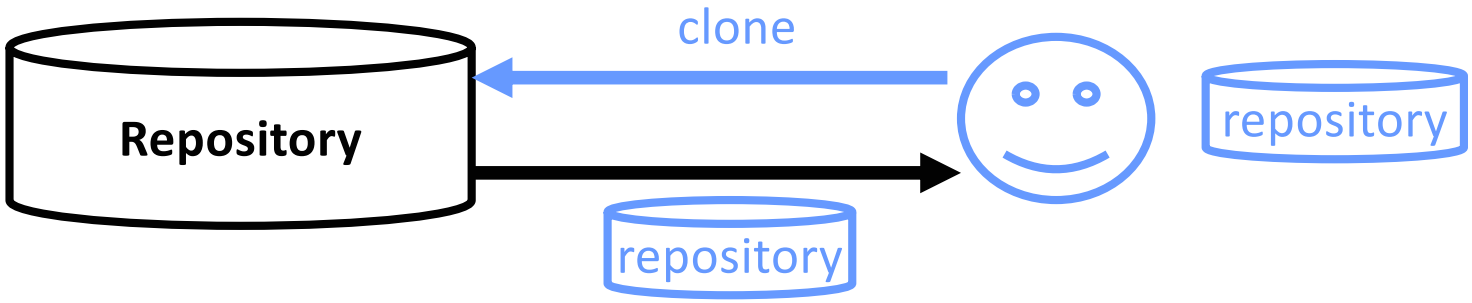
Using Git

- Git is a *distributed* VCS
- ***Repositories*** store all versions of all files, comments about changes ("commit messages", who made changes, …)



Using Version Control: `clone`

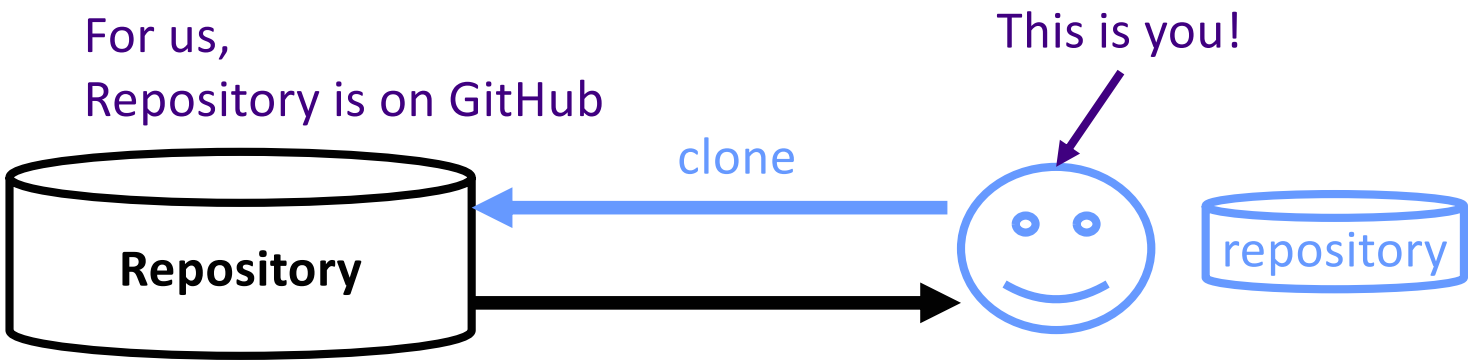
- Our typical workflow: first, **clone** the repository




```
git clone url_of_repository
```

Using Version Control: `clone`

- Our typical workflow: first, **clone** the repository



Using Version Control: `commit`

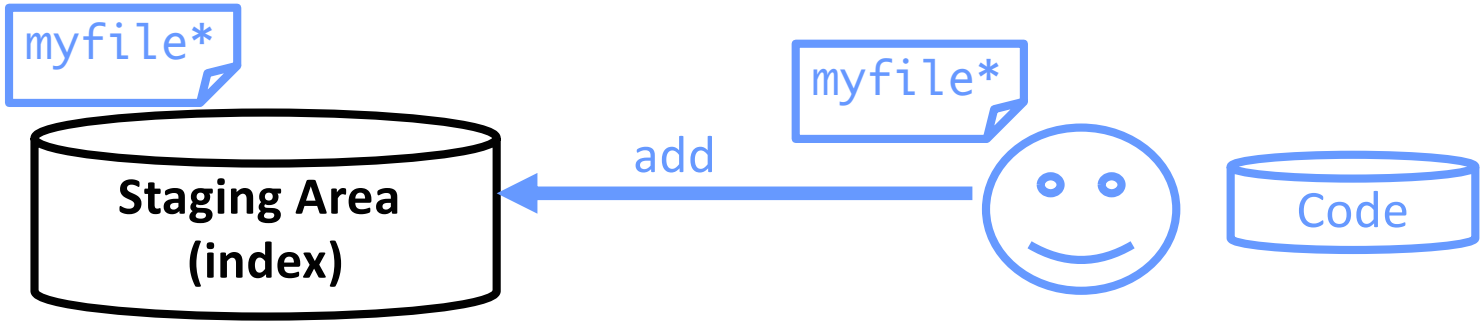
- After you make changes that you want to document, **commit** your version
  - Include comments about changes you made and ***why***

- Updates each modified file
- Records comments with updated files

```
git commit
```

Using Version Control: `add, delete`

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



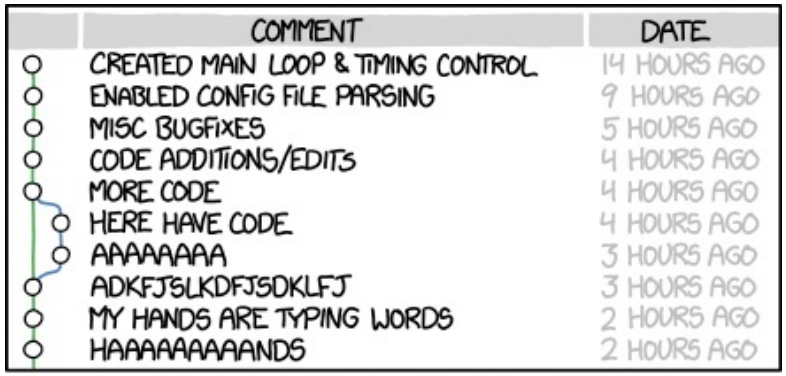
- Marks the files that will be part of the next commit
- When you commit, these files are added to your local repository

- Add, delete files and directories

```
git add myfile
```

Using Version Control: Commit Messages

- Many different conventions
- Make your messages meaningful and descriptive
  - Emphasis on the *why*
  - Your future self and contributors will thank you
  - Especially as you move to bigger projects with more collaborators



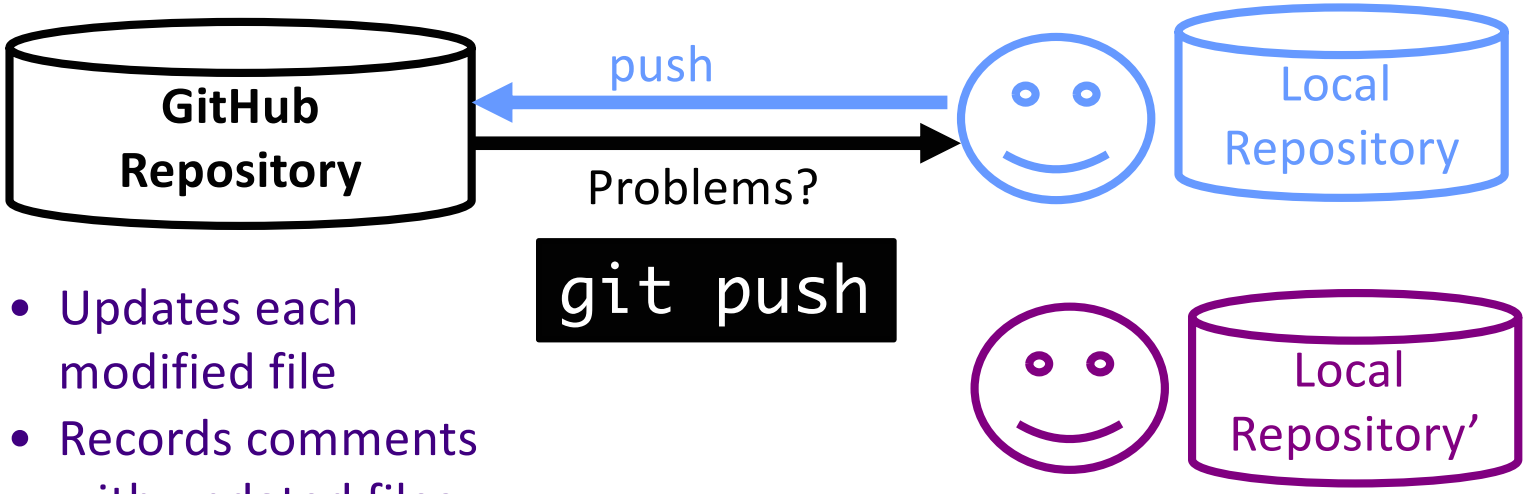
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

When Should I Commit?

- Depends – up to you
  - BUT: don't wait until you're completely done an assignment before committing
- Rules of thumb
  - Every time you get the next "thing" working
  - Every time you fix a bug

Using Version Control: **push**

- After you make changes that you want others (at first, that's just me) to see, **push** your version
  - Sends your previous commits and associated comments

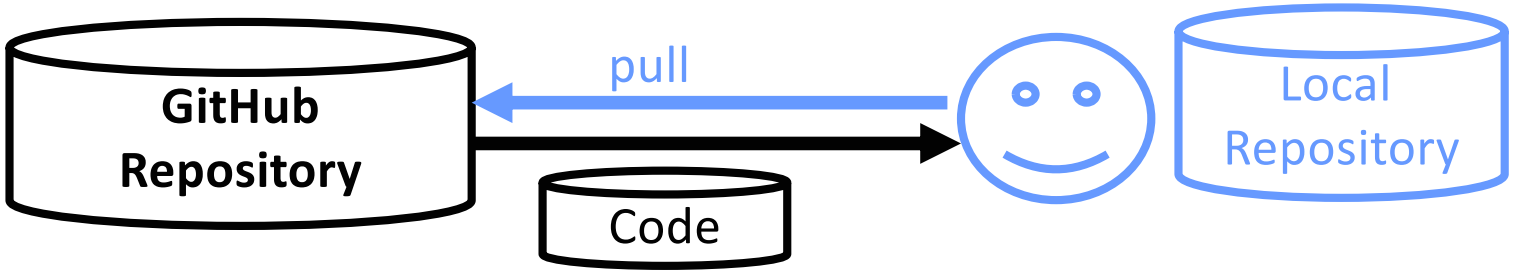


- Updates each modified file
- Records comments with updated files

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 (more on this later this term)




```
git pull
```

Using Git: Branches

- We create *branches* when we want to create a new "sandbox" to play in for …
  - New functionality
  - Bug fixes
  - Different approach

Using the Command Line

- What does a semicolon mean on the command line?

Using the Command Line

- What does a semicolon mean on the command line?
  - Semicolon ends/delimits a command
  - Ex: `command one; command two next; command3`

Why the Command Line?

- Because you *should* know it
  - Alumni feedback
- It can make your development process quicker
  - Caveat: After you get used to it
- Because you look so badass using it

Common Git Commands

Command	What it does
clone	Clones a repository – sets up your repository so that you can coordinate
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 named *name*
checkout <*name*>	Switches to the branch named *name*

Got it?
Let's practice!

https://xkcd.com/1597/

Looking Ahead

- Rest of today: working on Git Lab
  - Due Monday before class
  - Reminder: reload your assignments in the browser when you return to them
    - I may have updated based on student questions
- Set up Java for Monday