

# Objectives

- Design Wrap Up
- Introduction to Version Control

# Embedded Tutor: Vincent Ziccardi

- **Drop-in Hours**

- **Times:** Tuesday and Thursday 2:15pm to 4:15pm

- **Location:** Advanced Lab

- **Appointment Availability - Penji**

- **Location:** Anywhere on campus or Zoom

- **Times:** Tuesday 8-10pm, Sunday 12pm to 3pm

Hours may change

Schedule to be posted in advanced lab

# Review: 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 if you always want red/black tokens for all objects (just need one shared variable)
  - Alternatively, make it an instance variable if you want to be able to change the tokens' colors (which is a feature that could be added).
5. `Player` could be in `game.py` as an abstract parent *or* as a class in `war.py`.
  - Either answer can be justified.

## What kind of variable should this be?


- Should this variable persist (beyond this function/method)? Is this part of the class's/object's state?
  - If not → *local*
- Does each object of the class need its own variable/state? → *Instance*
- Or, should all objects of the class share one variable? → *Class*

# 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

# Readability of Given Code

# 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?

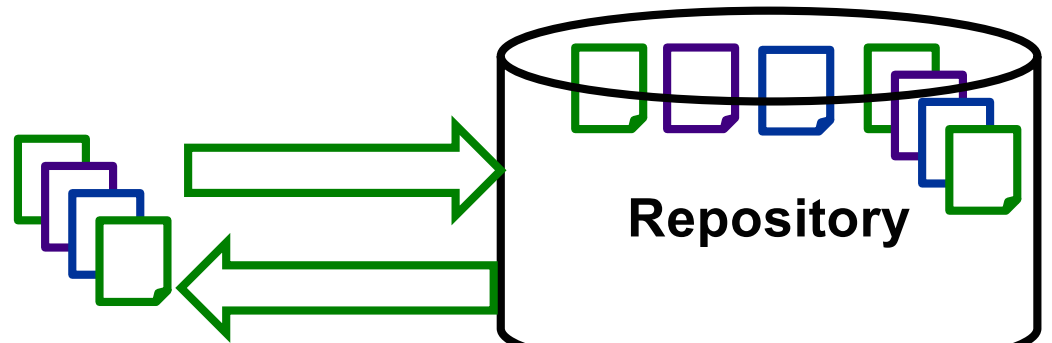
# Motivating Version Control: Collaboration

- 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 (VCS) 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

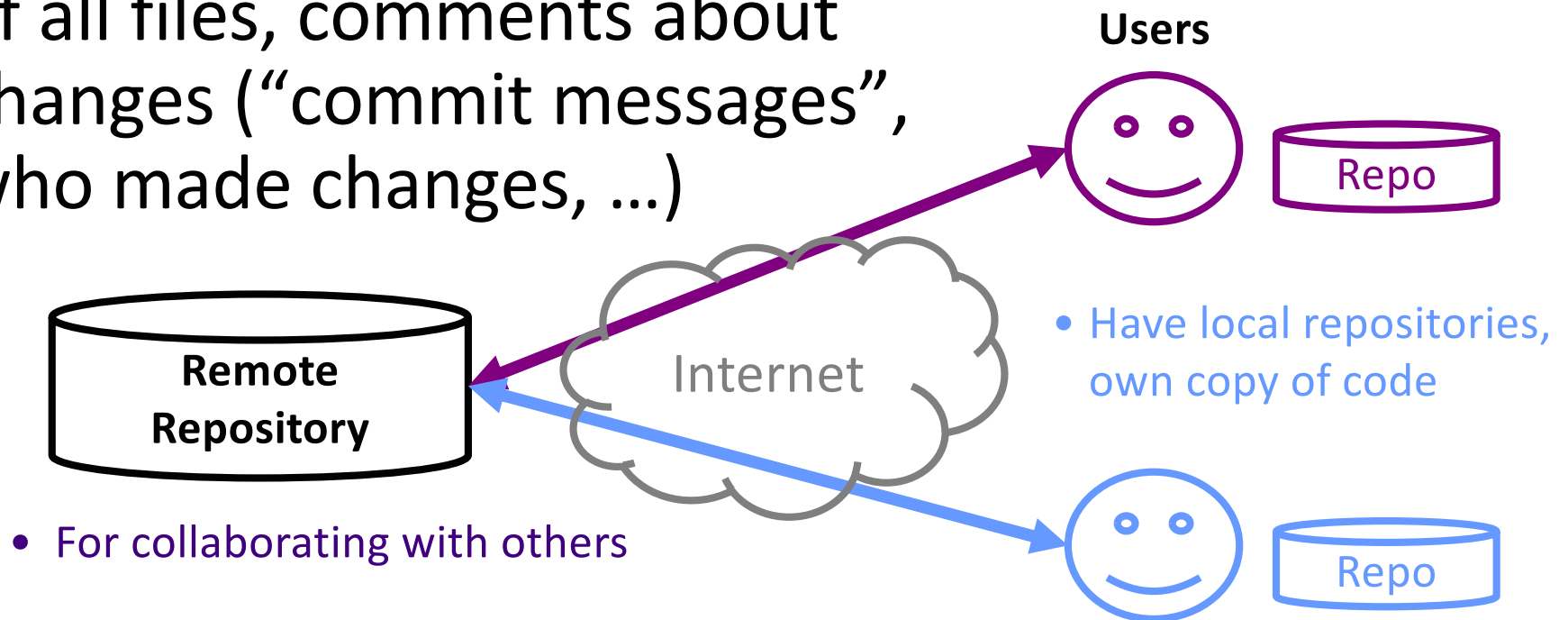
Command	What it does
clone	Clones a repository – sets up your repository so that you can coordinate
add <file>	Adds the <i>file</i> to the staging area
commit	Commits all the staged files (locally)
push	Push all your changes to the remote → You need your code to be pushed so that I can see it.
branch	List all local branches
branch <name>	Creates a new branch named <i>name</i>
switch <name>	Switches to the branch named <i>name</i>



<https://xkcd.com/1597/>

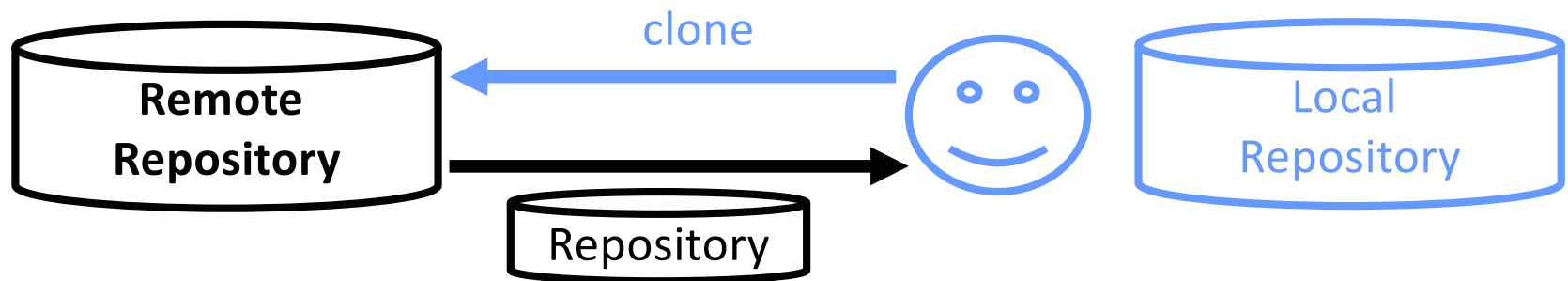
# Git Architecture

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



# Using Version Control: **clone**

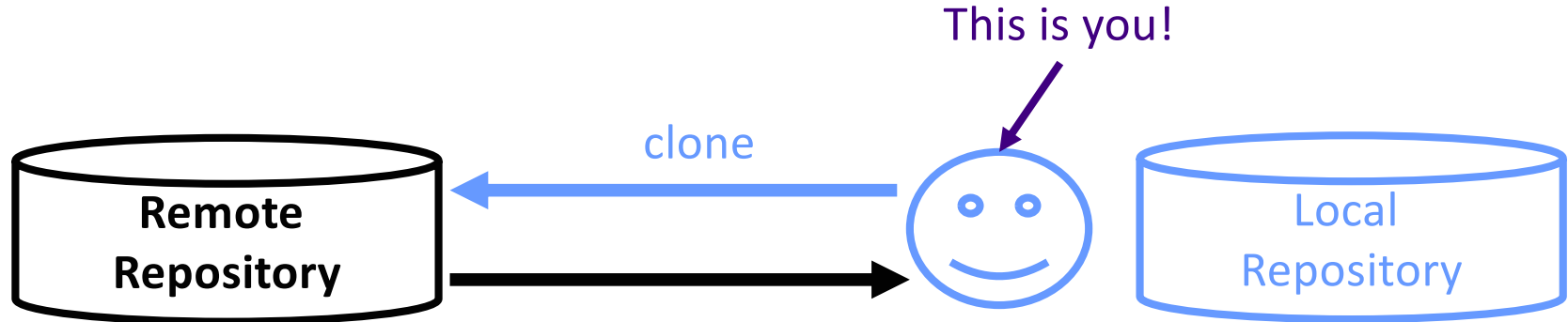
- Our typical workflow: first, **clone** the repository



```
git clone url_of_repository
```

# Using Version Control: **clone**

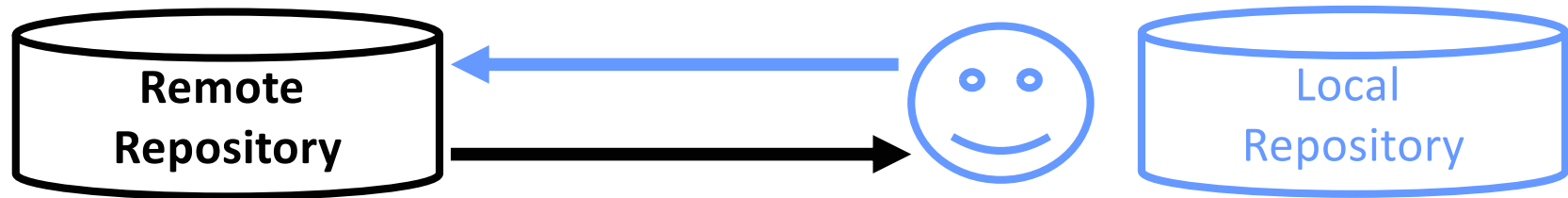
- Our typical workflow: first, **clone** the repository



For us, remote repository is hosted on GitHub

# Benefit of Distributed Version Control

- Every repository is a complete copy\* of the repository
- If a repository is deleted (from GitHub or a local repository), the existing repository can be pushed or cloned



# Using Version Control: **commit**

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

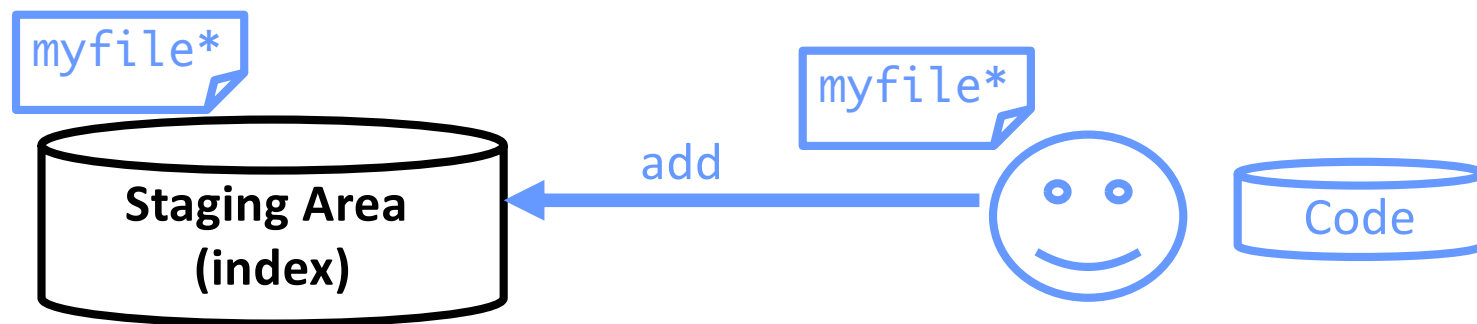


- Updates each modified file
- Records comments with updated files

```
git commit
```

# Using Version Control: **add**

- You need to **add** files to the *staging area* (or *index*), then **commit** those files



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

- Add files

```
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

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJ&LKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

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

# Commit Strategies

- Why commit?
  - You want to document changes to your code
    - Maybe you fixed a bug or implemented a new feature
  - You want to be able to undo the changes, potentially
- When should you 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

# Workflow (So Far) Recap

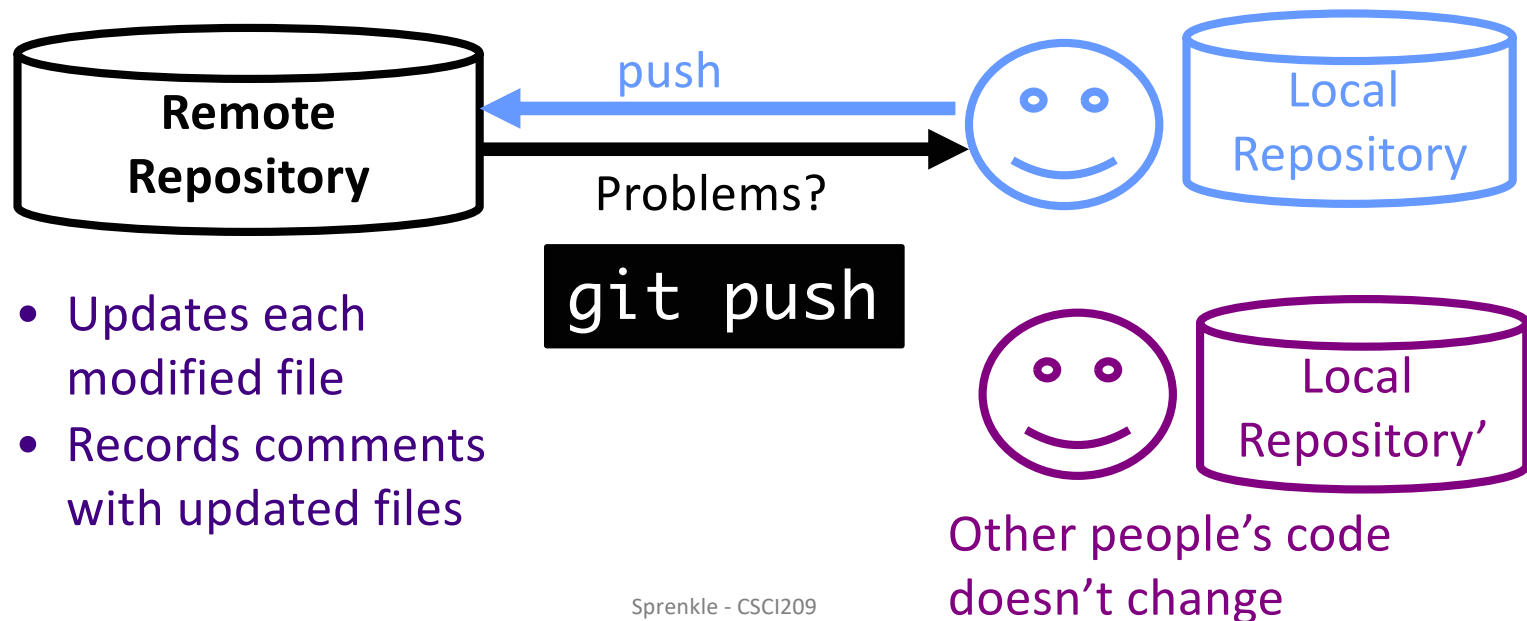
1. Clone repository (once)
2. Update files
3. Decide which files you want to commit
4. Add files to staging area
5. Commit, with descriptive message

## Using Version Control: **status**

- To see where you're "at" in the working tree
- Shows which branch you are on
- Shows which files are in the staging area
- Shows what files have been changed since the last commit

# 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

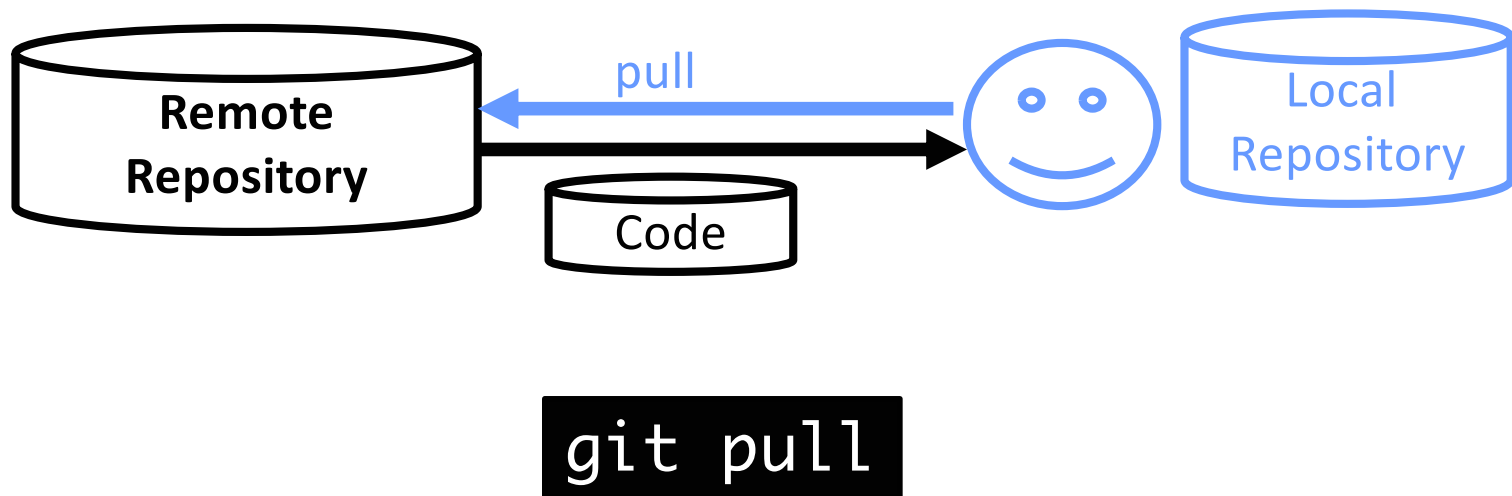


# When Should You Push?

- Depends!
- Periodically, after good check points
  - Note: helps to have a backup of your repository in the cloud
- Don't wait until you are completely done an assignment

# 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)



# Using Git: Branches

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



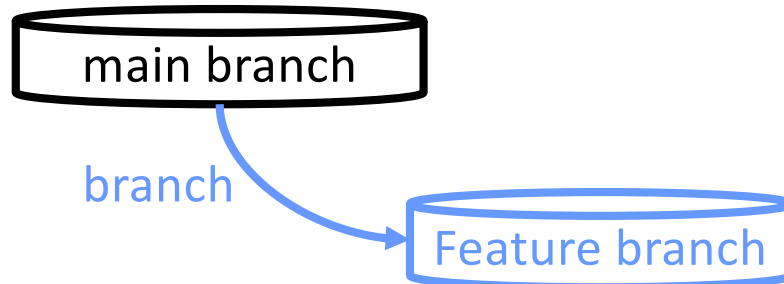
# Using Git: Branches

- Create a branch: `git branch branch_name`
- Switch to the branch:

```
git checkout branch_name
```

OR

```
git switch branch_name
```



# Using Git: Merge Branches

- Switch back to the main branch `git switch main`
- Merge changes from one branch into the current branch `git merge branch_name`



# Workflow Recap

1. Clone repository (once)
2. Create a new branch
3. Switch to new branch
4. Make changes to code
5. Add files to staging area
6. Commit with a descriptive message
7. Switch to main branch
8. Merge new branch
9. Push to remote

# Common Git Commands

Command	What it does
clone	Clones a repository – sets up your repository so that you can coordinate
add <file>	Adds the <i>file</i> 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 <i>name</i>
switch <name>	Switches to the branch named <i>name</i>

Got it?  
Let's practice!

<https://xkcd.com/1597/>



# 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

# Looking Ahead

- Rest of today: working on Git Lab
  - Due Wednesday before class
    - Recommended that you complete through Part 4 by Monday
  - Reload your assignments in the browser when you return to them
    - I may have updated based on student questions
- Set up Java
  - Part recommended by Monday
  - All by Wednesday