INTRO
GIT & GITHUB

Max Humber, DSI

LEARNING OBJECTIVES

▸ Answer the question 'what is Git?'

▸ Use/explain git commands like init, add, commit, push, pull, clone

▸ Distinguish between local and remote repositories

▸ Create, copy, and delete repositories locally or on Github

▸ Fork and Clone remote repositories

AGENDA

‣ Intro to Git

‣ Demo and Guided Practice: Individual Git Usage

‣ Collaboration with Git and Github

‣ Forking and Cloning

WHAT IS GIT?

‣ **Git** is a version control system for tracking changes to computer files

‣ Version control?

‣ Allows programmer to track changes at every version (called **commit**) of a file

‣ Also allows collaboration on projects

Name
- 120525_document_updated.txt
- 120604_document.txt
- 120605_document_amended.txt
- 120605_document_John.txt
- 120605_document_latest.txt
- 120605_document_latestcopy.txt
- 120605_document.txt
- 1200602_document.txt
- document_meeting.txt

This is only going to end in tears.
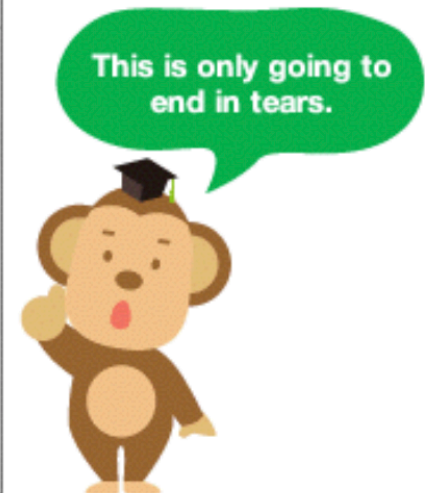
WHAT IS GIT?

▸ **More about version control:**

▸ A system that records changes to files over time

▸ Allows users to recall specific versions later
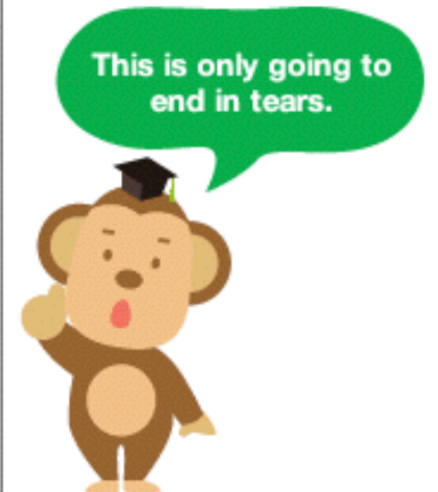
Version Control in Git: Commits

▸ A commit records changes within a file/directory

▸ Commit at milestones to be able to observe changes chronologically

▸ Each commit has a 40-character checksum hash as its identifier (IOW: a long number)

▸ When committing your changes, you must enter a commit message

  ▸ Provides descriptive comments regarding the changes you have made
  ▸ Separating different types of change (bug fixes, new feature, improvements...) into different sets of commits helps understand why and how those changes were made

▸ **Check:** Why might we want to only record changes at specific points instead of continuously? (Think Google Drive)

REPOSITORIES AKA REPOS

‣ A GIT codebase, holding all versions of a file and the tracked changes

‣ Like a directory with a history

‣ Can either turn an existing directory into a repository (command: `git init`) or clone an existing repo (command: `git clone`) onto local machine

‣ Do NOT make repositories within other repositories

‣ Do NOT turn directories **containing** other repositories into repositories
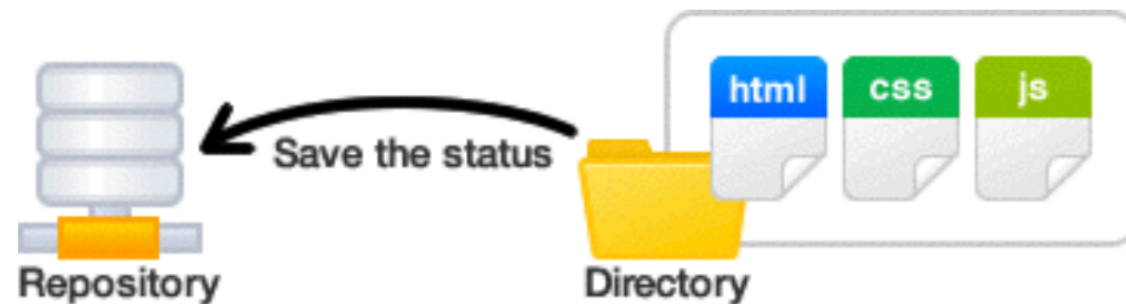
GIT WORKFLOW

‣ Developing a project revolves around the basic edit/stage/commit pattern:

1. First, you edit your files in the working directory.

2. When you're ready to save a copy of the current state of the project, you stage changes with `git add`.

3. After you're happy with the staged snapshot, you commit it to the project history with `git commit`.
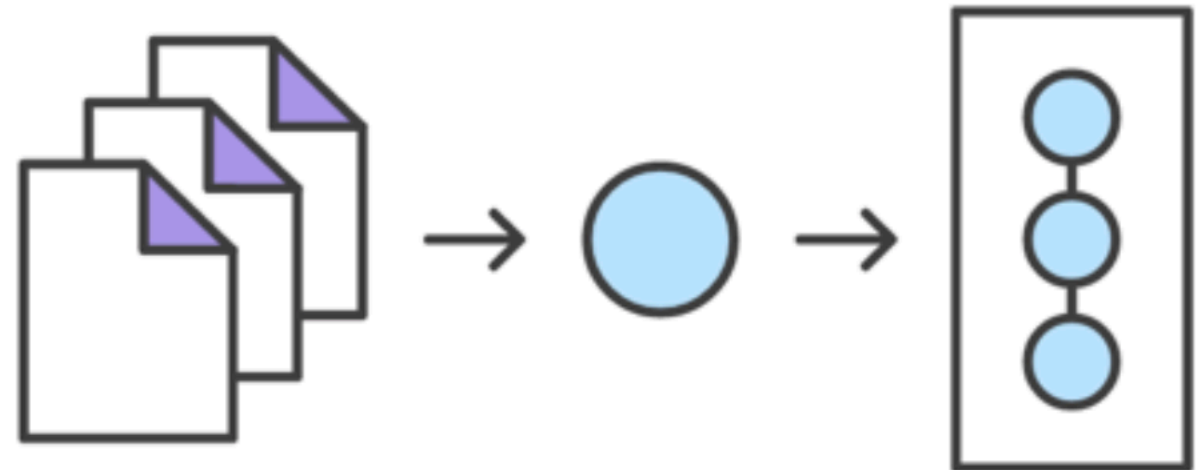
STAGES OF GIT

‣ Modified – changes have been made to a file but the file has not been staged or committed to the git database yet

‣ Staged – marks a modified file to go into your next commit snapshot

‣ Committed – files have been committed to the Git database
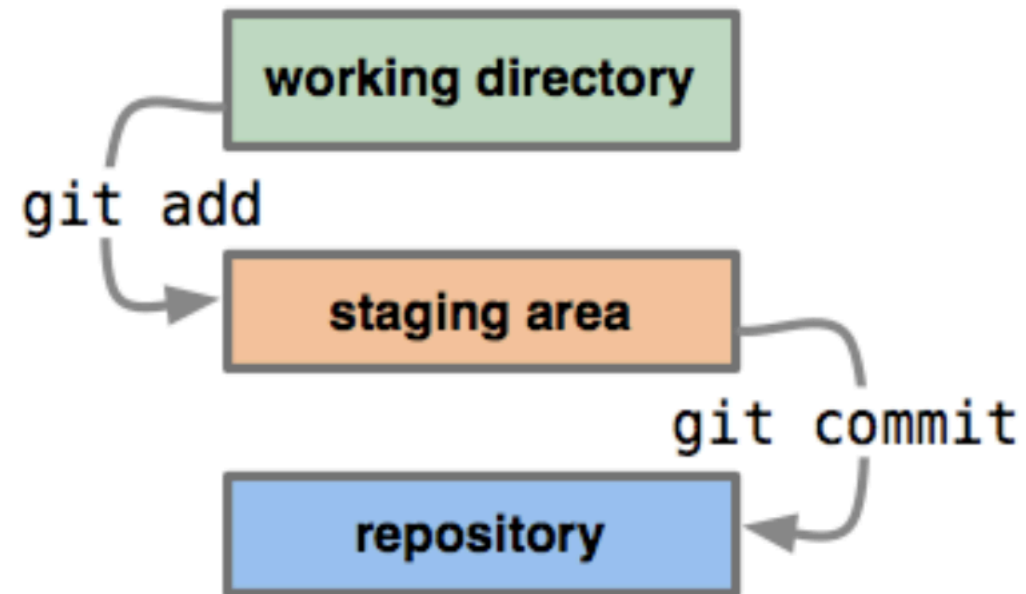
GIT COMMANDS

Command	Purpose
init	Creates new, empty Git repo
clone	Copies an existing Git repo
config	Allows you to configure your Git installation or an individual repo from the command line
add	Adds a change in the working directory to the staging area. Tells Git that you want to include updates to a particular file in the next commit. Does NOT actually record changes.
status	View the state of the working directory and the staging area. Lets you see which changes have been staged, which haven't, and which files aren't being tracked.
commit	Commits staged snapshot to project history. Will never be changed unless you explicitly tell it to.
log	Displays committed snapshots. Lets you list, filter, or search project history. ONLY operates on committed history.
checkout	Switch from one branch to another
revert	Undoes a committed snapshot. Does not remove the commit from project history, instead figures out how to undo the changes introduced by the commit.
remote -v	List all currently configured remote repositories
remote add	If you haven't connected your local repository to a remote server, add the server to be able to push to it
fetch	Fetch and merge changes on the remote server to your working directory
merge	To merge a different branch into your active branch
diff	Show all merge conflicts
branch	List all the branches in your repo, and also tell you what branch you're currently in
push <a> <b>	Send changes to a (destination, remote repo) from b (current branch)
pull	Fetch and merge changes on the remote server to your working directory
reset	Like revert, but DOES remove the commit from project history. Basically a permanent undo. Be careful - this is one of the only Git commands that could allow loss of work.
clean	Removes untracked files from your working directory. Like an ordinary rm command, git clean is not undoable, so make sure you really want to delete the untracked files before you run it.

INDIVIDUAL GIT USAGE

REPO PROCESSES

‣ Establish existing folder as repository or make a new folder (one-off):

- ‣ Navigate into that folder
- ‣ `git status` (check if folder is already a repo)
- ‣ `git init`
- ‣ `git status` (untracked files)

‣ Add files to the staging area

- ‣ `git add .`
- ‣ `git status`

‣ Make first commit

- ‣ `git commit —m "first commit"`
- ‣ `git status`

‣ Make changes to repo and commit them

- ‣ `touch mouse.txt`
- ‣ move through workflow (status, add, status, commit, status)
- ‣ `git log` (shows commit history

‣ Revert (undo) commit:

- ‣ `git revert` <last commit id>
- ‣ if stuck in bash screen, type :wq
- ‣ `ls` (`cat.txt, dog.txt, bird.txt,` and `fish.txt` should be gone)
- ‣ `git log` (new commit created, could still go back to older version)
- ‣ `git reset` would NOT do this

‣ Reset (go back to) commit:

- ‣ `git reset --hard <some commit id>`
- ‣ Unlike `git revert`, this actually deletes commits

WHAT IS GITHUB?

‣ DIFFERENT FROM GIT

‣ Hosting service for Git repositories

‣ Web interface to explore Git repositories

‣ Social network for programmers

‣ Allows collaboration on git repositories

‣ Github **uses** Git
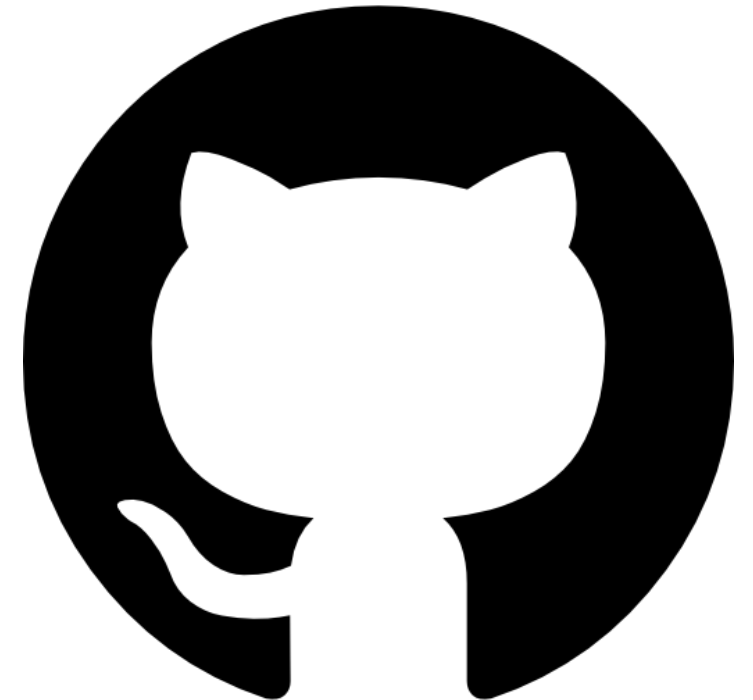
COLLABORATING WITH GIT

‣ Each developer gets their own copy of the repo

‣ Usually want to share a series of commits (rather than every one)

‣ Publish local history by "**pushing**" commits to other repositories

‣ Acquire changes from other contributors by "**pulling**" commits into your local repository
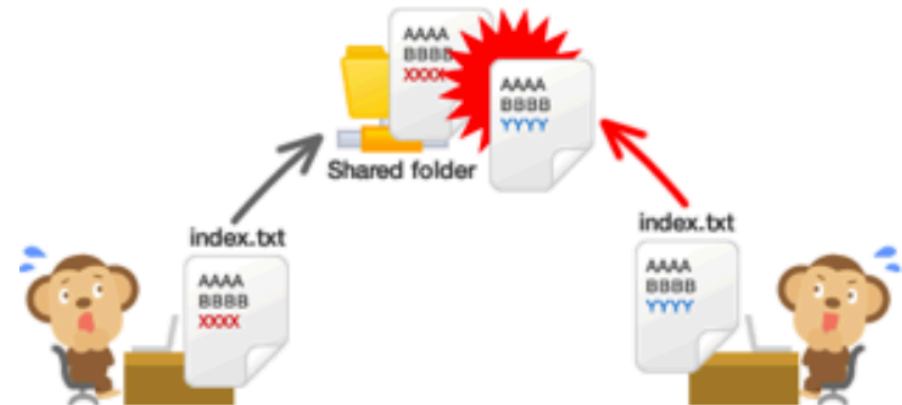
REMOTE VS LOCAL REPOSITORIES

‣ Local repository: on local machine of individual user
  ‣ Can use all of Git's version control features (reverting changes, tracking changes, etc.)
  ‣ Can be new (init) or a copy (clone)

‣ Remote repository: on a remote server, often shared by team members
  ‣ Used for sharing your changes or pulling changes from your team
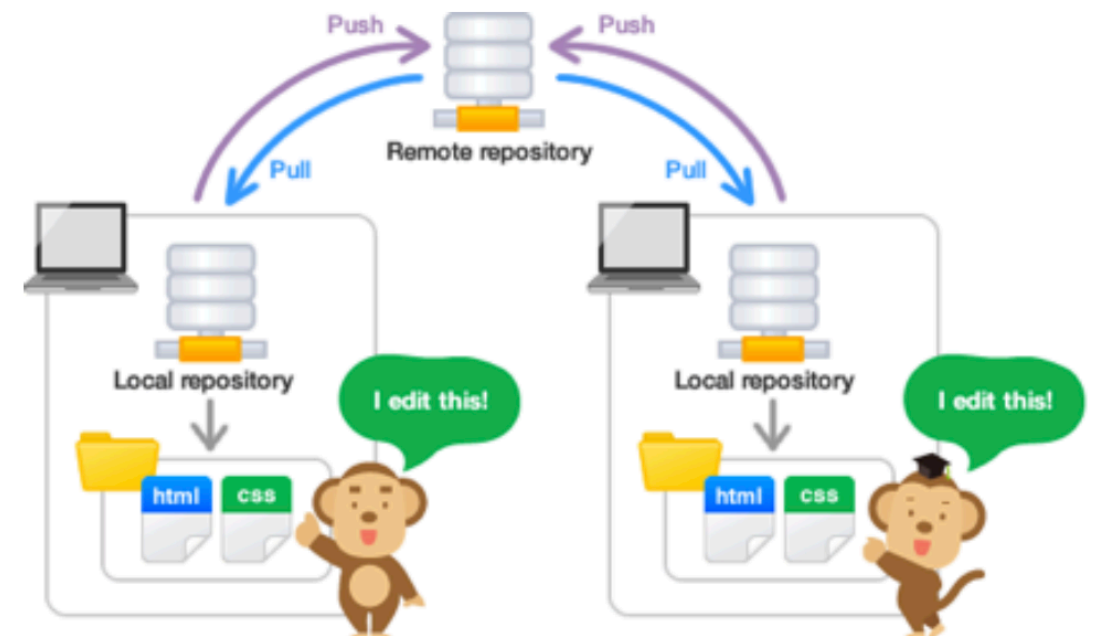
SETTING UP CONNECTION

‣ Local repository: on local machine, every team member has their own

‣ Remote repository: on remote server, can have individual branches/forks but usually members share a central repo
   ‣ List current remote connections with a repository using "git remote –v"

Repo Setup	Creation	Connection
Fresh/New	git init (from within directory)	git remote add <name> <url>
Copy	git clone	automatic from original repo

FORKING

The project maintainer initializes the official repository

Developers fork the official repository

Developers clone their forked repositories

Developers work on their features

Developers publish their features

Mary creates the pull request

The project maintainer integrates their features

Developers synchronize with the official repository

The Forking Workflow is fundamentally different than other workflows.

Instead of using a single server-side repository to act as the "central" codebase, it gives every developer a server-side repository.

This means that each contributor has not one, but two Git repositories: a private local one and a public server-side one.

PULL REQUESTS

ACTIONS

- Clone
- Create branch
- Create pull request
- Compare
- Fork
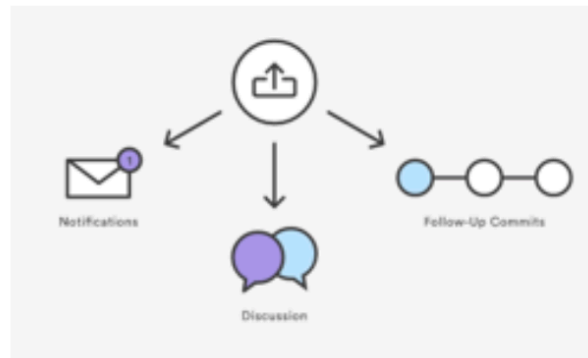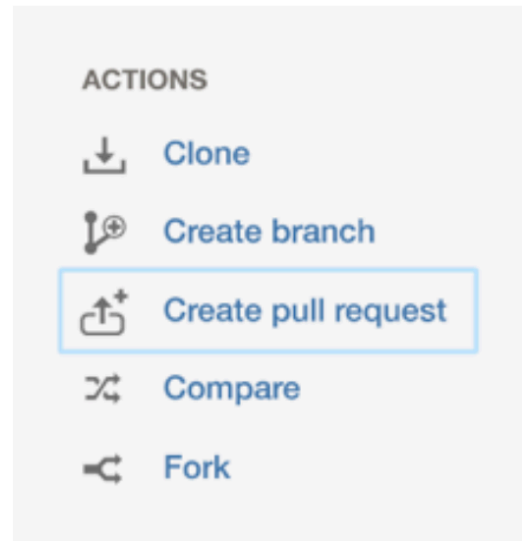
‣ Alerts team to view proposed code and merge it into main project
‣ Forum to discuss proposed changes/features before integrating them ‣ Teammates can post feedback in the pull request
‣ Teammates can tweak the feature by pushing follow-up commits
‣ All of this activity is tracked directly inside of the pull request
In the pull request process, the developer:
‣ *(Only if branching) Creates the feature in a dedicated branch in local repo*
‣ Pushes the branch to a public repository
‣ Files a pull request
‣ The rest of the team reviews the code, discusses it, and alters it.
‣ The project maintainer merges the feature into the official repository and closes the pull request

PULL REQUESTS
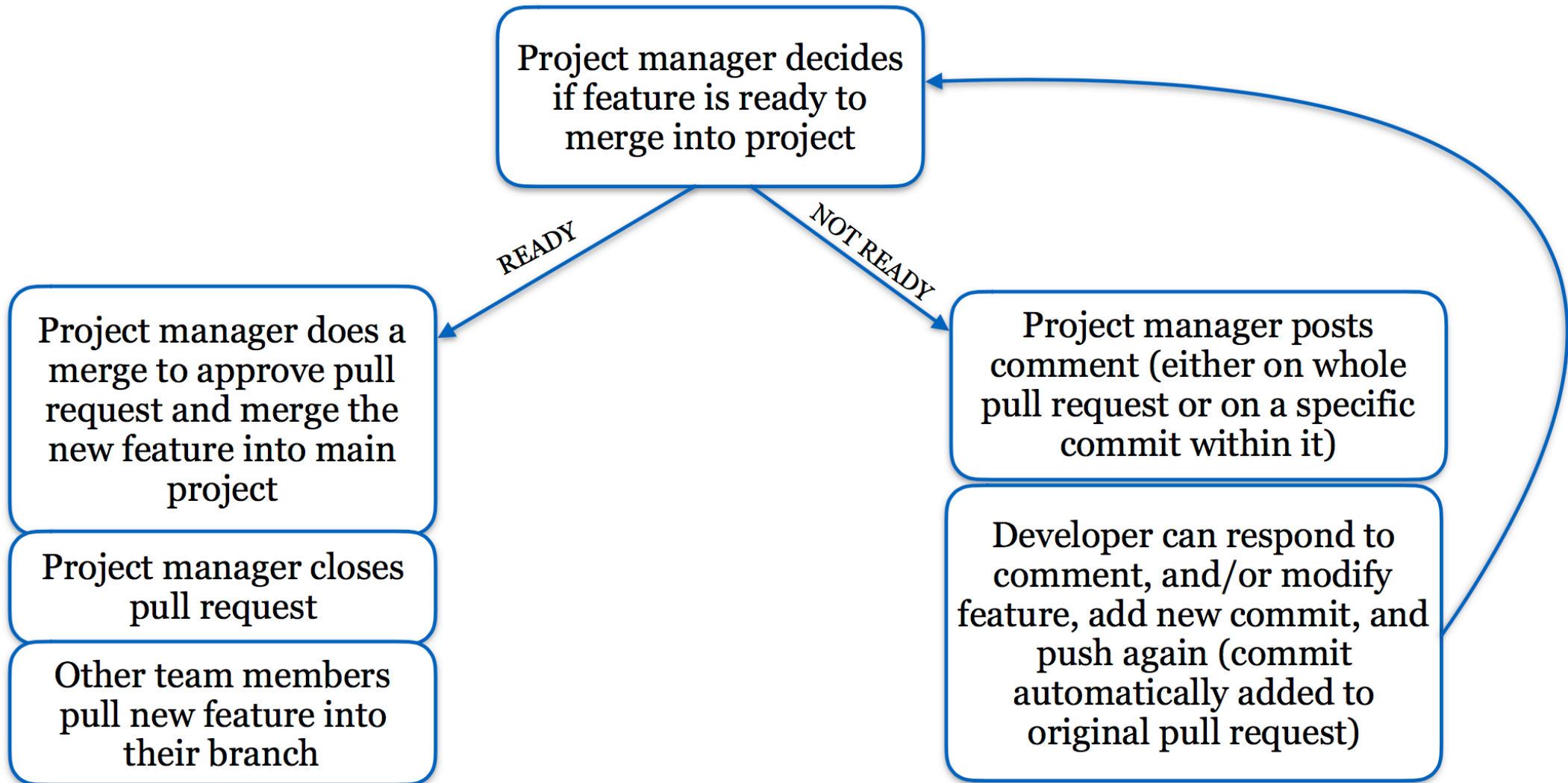
MERGE CONFLICTS

If the two branches you're trying to merge contain conflicts in the same part of the same file, Git won't know which to use and will stop so that you can manually resolve the conflict.

In a merge conflict, running `git status` shows you which files need to be resolved, like this:

```
# On branch master
# Unmerged paths:
# (use "git add/rm ..." as appropriate to mark resolution)
#
# both modified: hello.py
#
```

The developer needs to:
▸ Fix the conflict by editing the files
▸ Run `git add` on the file(s) to signal that the issue is resolved
▸ Run `git commit` to generate the merge commit
Look familiar?

FINAL NOTES

- **Four commands you will use the most**:
  ```
  git add
  git commit -m
  git push
  git pull
  ```


- **Additional important commands**:
  ```
  git status
  git log
  git reset
  git revert
  ```

.gitignore:

A file where you can write names of files that you do not want git to track