How to Find Awesome Open Source Projects

Featuring “Version Control --How to Use Git and GitHub”

Credits go Udacity, I have merely repackaged their content for presentation.

Introduction: Part 1

- We’ll cover the concept of version control
  - Sorta like having a giant undo button for your projects
  - Save different versions of your files at different points in time
    - Restore previous versions
    - Compare to previous versions
  - Makes it easier to collaborate
    - Better than sending big zip files back and forth

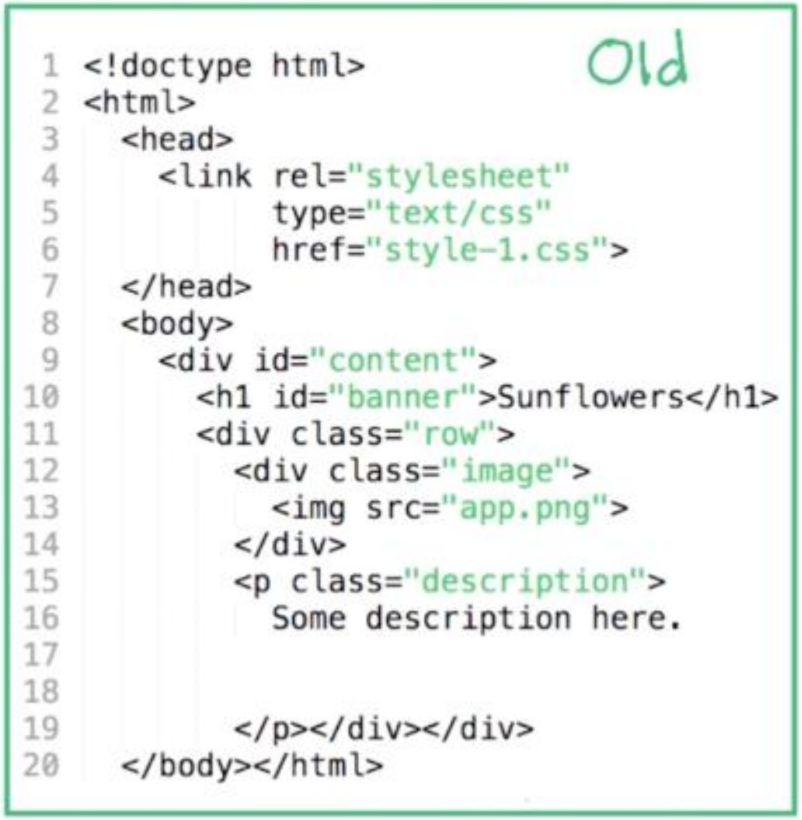
Overview

- Learn to use Git
  - A version control system
- Learn to use GitHub
  - A code sharing and collaborating platform

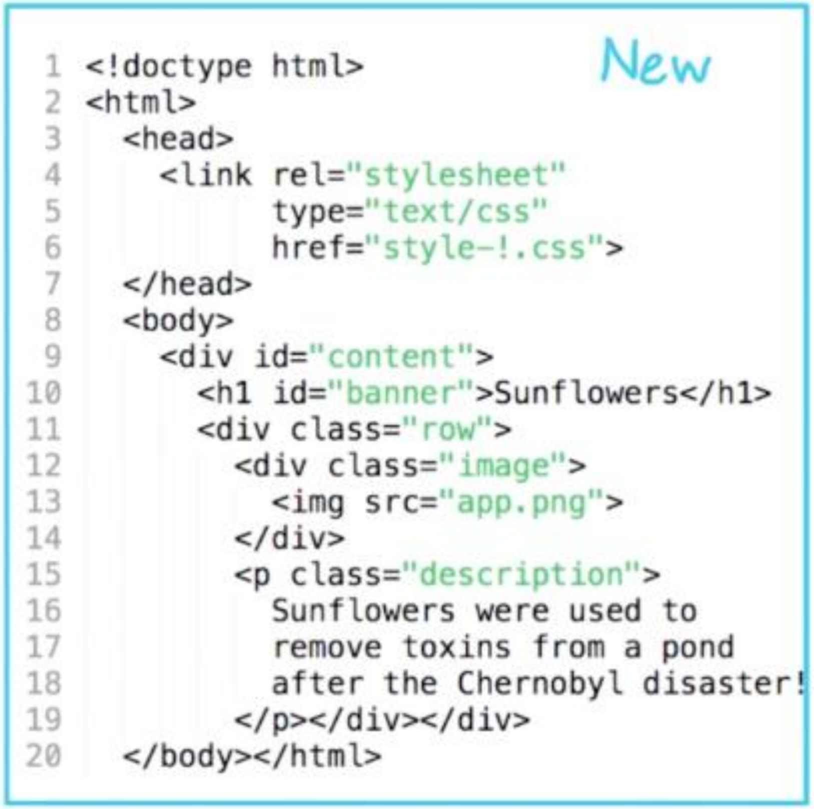
- Need to use a unix style command line
- This will also make a little more sense to those with previous coding experience

- Three Parts:
  - 1: How version control is useful, and what different forms it can take on.
  - 2: Learn more about Git and start using it
  - 3: Practice sharing code and collaborating on GitHub

Differences Between Two Files



```
 1 <!doctype html>
 2 <html>
 3   <head>
 4     <link rel="stylesheet"
 5           type="text/css"
 6           href="style-1.css">
 7   </head>
 8   <body>
 9     <div id="content">
10       <h1 id="banner">Sunflowers</h1>
11       <div class="row">
12         <div class="image">
13           <img src="app.png">
14         </div>
15         <p class="description">
16           Some description here.
17
18
19         </p></div></div>
20   </body></html>
```



```
 1 <!doctype html>
 2 <html>
 3   <head>
 4     <link rel="stylesheet"
 5           type="text/css"
 6           href="style-!.css">
 7   </head>
 8   <body>
 9     <div id="content">
10       <h1 id="banner">Sunflowers</h1>
11       <div class="row">
12         <div class="image">
13           <img src="app.png">
14         </div>
15         <p class="description">
16           Sunflowers were used to
17           remove toxins from a pond
18           after the Chernobyl disaster!
19         </p></div></div>
20   </body></html>
```

- Lines 6, 16, 17 and 18 have changed

Finding Diffs Between Larger Files

- What if there are 100’s or 1000’s of lines?
  - E.g, Asteroids Game
    - Try it here: http://dougmcinnes.com/html-5-asteroids/
  - There is a bug in this game!
    - Someone changed the key mappings
      - Up arrow -> Space
      - Space -> Enter
    - And now the engines don’t work! Typo? Where is it?
- Download the two versions of the file
  - https://www.udacity.com/api/nodes/2960778928/supplemental_media/game-oldjs/download?_ga=1.32442489.672083044.1467344711
  - https://www.udacity.com/api/nodes/2960778928/supplemental_media/game-newjs/download?_ga=1.34079078.672083044.1467344711
- Too long to compare the old version to the new version by hand. They are over 1000 lines!
  - Automatically compare files
    - Windows – FC (file compare)
      - cd directory/with/files
      - FC file1 file2
    - Mac or Linux – Diff
      - cd directory/with/files
      - diff -u file1 file2
        - » -u stands for unified diff format (makes output easier to read)

Reflections

- How did viewing a diff between two versions help you spot the bug?
  - Why should I?
    - It kinda helps cement the material
    - We’ll use this content for VC Practice
  - How?
    - One file per part
    - Plain-text, NOT rich-text

Reflect: Using diff to Find Bugs

- Make sure you can access the command line
- Choose a text editor
  - Notepad++, Sublime, Atom, Emacs, Vim, etc…
- Make sure you can launch your editor from the command line (not required)
- Set up your workspace
  - cd ~ # change directories to your home directory
  - mkdir version-control # make version-control directory
  - cd version-control # go to version-control directory
  - mkdir reflections # create reflections directory
  - cd reflections # go to reflections directory
  - subl lesson_1_reflections.txt # launch sublime (or any other editor) with a file called lesson_1_relfections.txt
  - pwd # print working directory – shows what directory your are in
  - ls # list the files in this directory
- Use short lines
  - Git is less useful if your files contain very long lines. Remember to press enter to make a newline.
- Do the first reflection exercise
  - lesson_1_reflections.txt
  - **How did viewing a diff between two versions of a file help you see the bug that was introduced?**
  - Create your document, write down your thoughts, and save the file.

Where Do Versions Come From?

- Versions
  - Saving manual copies
  - Dropbox
  - Google Docs
  - Wikipedia

Reflect: Using History for Efficiency

- **How could having easy access to the entire history of a file make you a more efficient programmer in the long term?**
- Update your document.

Properties of a VCS For Code

Feature Comparison Chart						
	Any Editor	Use offline				
Manual Saving	**YES**	**YES**				
Dropbox	**YES**	**NO**				
Google Docs	**NO**	**NO**				
Wikipedia	**NO**	**NO**				

<u>When to Save:</u>

As a programmer, when would you want to have a version of your code saved?

- At regular intervals (e.g. every hour)
- Whenever a large enough change is made (e.g. 50 lines)
- Whenever there is a long pause in editing
- When you choose to save a version

Manual Commits In Git

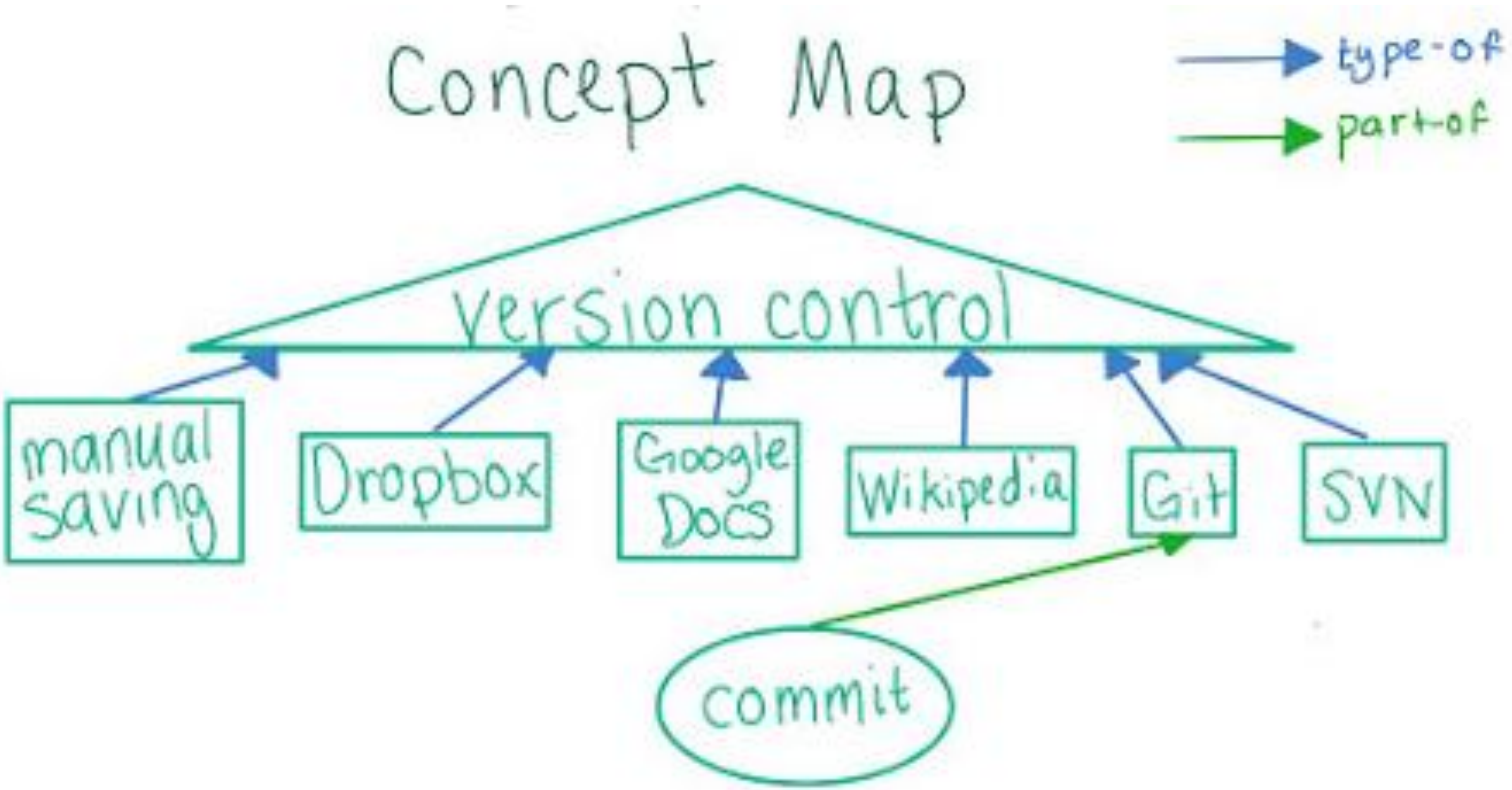
Feature Comparison Chart

	Any Editor	Use offline	Manual Save			
Manual Saving	**YES**	**YES**	**YES**			
Dropbox	**YES**	**NO**	**NO**			
Google Docs	**NO**	**NO**	**NO**			
Wikipedia	**NO**	**NO**	**YES**			
Git	**YES**	**YES**	**YES**			
SVN	**YES**	**NO**	**YES**			

Commits are user created checkpoints. Examples of commit messages are:

- fix off-by-one bug
- add cool new feature
- Improve user docs

Creating a Concept Map



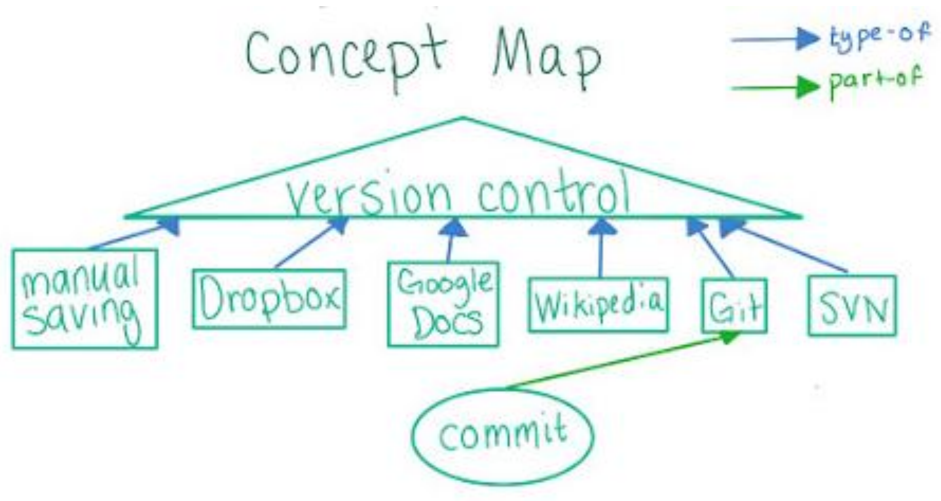
Using Git to View History

- Viewing the history of a file on Git can be done completely offline (ex. game.js)
  - cd version-control/asteroids/
  - git log
    - Every commit that’s every been made, starting with the most recent

      - ID: Sorta like a serial number (unique identifier)
      - Author
      - Date
      - Message: Explains what has changed since the last commit
  - git diff commitID#1 commitID#2
    - Similar to FC or diff program, but compares different versions of a file within git
    - Output is similar to the output from diff
    - Colorized!!!

Concept Map: diff

- If we were to put "git diff" on the map, which of the following existing concept(s) would we connect with it?
  - commit
  - Google Docs
  - git
  - Version Control



One Commit Per Logical Change

- How Often to Commit
  - Since you can choose when to make a commit, you might be wondering how often to commit your changes.
  - A good rule of thumb is to make *one commit per logical change.*
- Commit Size
  - You commit all the changes required to add a new feature, which you've been working on for a week. You haven't committed since you started working on it. (Too Big)
  - You find three typos in your README. You fix and commit the first. (Too Small)
  - You commit all the changes required to add a new feature, which you've been working on for an hour. (Just Right)
  - You fix two small bugs in different functions and commit them both at once. (Too Big)
- It's always a judgment call!

Reflect: Manual Commits

- **What do you think are the pros and cons of manually choosing when to create a commit, like you do in Git, vs having versions automatically saved, like Google Docs does?**
- Update your document.

Tracking Across Multiple Files

- Tracking across files
    - E.g., Function in one file, call to that function in another file. If you add an argument to the function, you'll need to change code in both files
  - Where does it make sense to track multiple files together rather than separately?
    - Competition-style coding
    - html and css files that make up a webpage (yes)
    - Photos you have photoshopped
    - A novel, split up by chapter (could go either way)

Git Commits Across Multiple Files

- Commits with Multiples Files
  - Git calls this collection of files is called a repository
  - When you make a commit, you save a version of every file in your repository
    - Tracks the state of game.js, index.html, index.css; even if you don't change all of them.
- git log --stat
  - Gives some statistics about which files changed.
  - Green plus signs = additions
  - Red minus signs = deletions
  - Some commits affect more than one file
- git diff commitID#1 commitID#2
  - File changed followed by a summary of the changes

Reflect: Multi-File Commits

- **Why do you think some version control systems, like Git, allow saving multiple files in one commit, while others, like Google Docs, treat each file separately?**
- Update your document

Installing Git

- If you already have Git installed, check it's version using the command git --version (1.8)
- Otherwise install/upgrade Git here https://git-scm.com/downloads
- Let me know if you have any problems

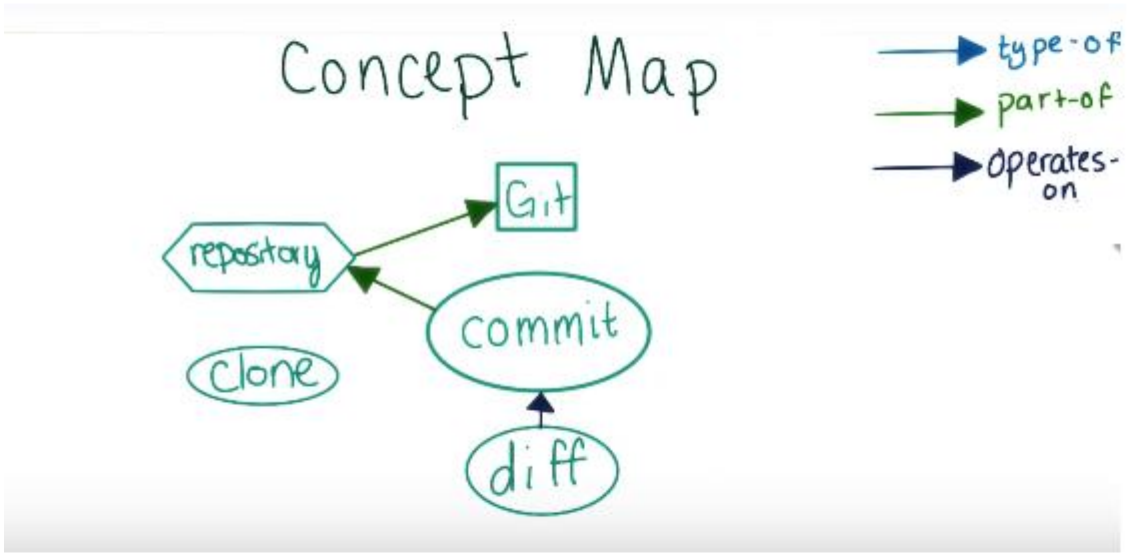
Cloning and Exploring the Repo

- Cloning a Repository
  - git clone RepositoryURL
- Asteroids URL
  - https://github.com/udacity/asteroids.git
- Exiting git log
  - To stop viewing git log, press q (stands for quit)
- Getting colored Output
  - Run git config --global color.ui auto
- Copying and Pasting from the Command Line
  - Windows: Cntl + Shift + Insert (enable QuickEdit)
  - Mac: Cmd + C, Cmd + V
  - Ubuntu: Cntl + Shift + C, Cntl + Shift + V
- Using git log and git diff
  - Running git log shows a list of the recent commits with info about them
  - Running git diff followed by the two commit ID's compares the two versions
- Entering commit IDs
  - It's easier to enter just the first four or more characters of the commitID rather than the entire commit ID.

Reflect: Using Git to View History

- **How can you use the commands git log and git diff to view the history of files?**
- Update your document

Concept Map: Repository, Clone, Log



Which concepts should log be connected to?

Git Errors And Warnings

- Strange Warnings and Error Messages
  - Sometimes Git gives cryptic, confusing, or downright bizarre errors and warnings.
- Which of the following is NOT a Git error or warning?
  - Should not be doing octopus
  - You are in ‘detached HEAD’ state
  - Panic! (the ‘impossible’ happened)

Checking Out Old Versions of Code

- Checking Out Prior Commits
  - Temporarily changing our files back to the way they were at the time of that prior commit. This is called git checkout
    - Not the same as an svn checkout
  - Good for finding out which commit introduced a bug
    - Checkout the commit and run the code.
- Running the Asteroids Code
  - Open the index.html file in a web browser
    - Find the file in Windows Explorer, Finder, etc
    - Open with Google Chrome or Firefox (not IE)
    - Endless stream of bullets is a bug!!!
  - Checkout the 'revert controls' commit, to see if it has the bug
    - git checkout commitID
    - Will get warning 'You are in 'detached HEAD' state.'
    - git log, when run from an earlier commit will not show future commits.
    - Most recent commit ID is 3884eab839af1e82c44267484cf2945a766081f3
- What is the ID of the first commit with the bug?

Reflect: Confidence from Version Control

- **How might using version control make you more confident to make changes that could break something?**
- Update your document

Git Workspace

- Purple user name
- Blue directory
- Green commit ID
- Star when changes have been made
- Tab completion
- Open a text editor when you want to write a commit message

Setting Up Your Workspace on Windows

- Use a white background (you can skip this step)
  - Icon in upper left -> Defaults -> Colors
  - Set screen color to white, screen text to black
  - Close and reopen GitBash
- Download this file for text completion
  - https://raw.githubusercontent.com/git/git/master/contrib/completion/git-completion.bash
- Download this file for git features in the prompt
  - https://raw.githubusercontent.com/git/git/master/contrib/completion/git-prompt.sh
- Download this file for configuring git bash
  - https://www.udacity.com/api/nodes/3341718587/supplemental_media/bash-profile-course/download
- After downloading move the files
  - cd ~ (go to home directory)
  - mv Downloads/git-completion.bash.txt   git-completion.bash
  - mv Downloads/git-prompt.sh.txt   git-prompt.sh
  - mv Downloads/bash_profile_course   .bash_profile
    - If you already have a bash profile, just copy and paste the content from this file into that file.
    - Feel free to modify the file if you don't want to keep all of the things in it.
- Configure Sublime (or any other text editor) (C:\ProgramFiles\Sublime Text 2)
  - git config --global core.editor "'path\to\sublime\sublime_text.exe' –n –w"
    - N = new window
    - W = wait for you to close before trying to continue
  - Add this to your bash_profile (it could be a hidden file) – alias subl="path\to\sublime\sublime_text.exe"
- Run these config commands
  - Git config --global push.default upstream
  - Git config --global merge.conflictstyle diff3
- Close and reopen for your changes to take effect.

Setting Up Your Workspace On Mac

- Download this file for text completion
  - https://raw.githubusercontent.com/git/git/master/contrib/completion/git-completion.bash
- Download this file for git features in the prompt
  - https://raw.githubusercontent.com/git/git/master/contrib/completion/git-prompt.sh
- Download this file for configuring git bash
  - https://www.udacity.com/api/nodes/3341718587/supplemental_media/bash-profile-course/download
- Right click > save as.
  - Save to Downloads directory
  - Uncheck "hide extension"
  - Rename to extension to be .bash or .sh
- After downloading move the files
  - cd ~ (go to home directory)
  - mv Downloads/git-completion.bash   git-completion.bash
  - mv Downloads/git-prompt.sh git-prompt.sh
  - mv Downloads/bash_profile_course   .bash_profile
    - If you already have a bash profile, just copy and paste the content from this file into that file.
    - Feel free to modify the file if you don't want to keep all of the things in it.
    - Linux users might need to rename this file .bashrc instead of .bash_profile
- Configure Sublime (or any other text editor) (/Applications/Sublime \Text \2)
  - Add this to your bash_profile (it could be a hidden file) – alias subl="/Applications/Sublime\ Text\ 2.app/Contents/SharedSupport/bin/subl"
  - git config --global core.editor "subl –n –w"
    - N = new window
    - W = wait for you to close before trying to continue
  - Git config --global push.default upstream
  - Git config --global merge.conflictstyle diff3
- Close and reopen for your changes to take effect.

Reflect: How Do You Want to Use Git?

- **Now that you have your workspace set up, what do you want to try using Git for?**
- Update your Document

Part 1 Summary

- Saving different versions of your files can be useful
- Practice using Git to review the history of existing projects
- Now lets create a Git repository!

Introduction: Part 2

- Learn how to use Git on your own machine, for solo projects
- Create a new repository and make commits
- Merging and Branching
- Part 3: Using Git collaboratively

What Makes A Repository A Repository?

- Looks like just another normal directory
- The difference is that Git stores a bunch of metadata about the history of the repository
- Tucked away in a hidden .git directory
- Command ls –a can help you see these hidden files

Initializing a Repository

- All this can be done offline
- Creating a Git repository
  - ls –a (you won't see a .git directory)
  - Run git init in either an existing directory
  - ls –a (now you will!)
- How many commits do you think the new repository contains?
  - 0! (Git could create an initial commit, but what if you don't want to commit everything in the directory?)
  - Running git log can prove this

Examining the New Repository

- Running git log produces:
  - fatal: bad default revision ‘HEAD’
- Running git status will prove that this is still a Git repository
  - You will be on the master branch
  - The commit will be ‘initial commit’
  - All of your files should be untracked
- Creating your reflections repository
  - Run the commands git init and git status in the directory with your reflections

Reflect: Initializing a Repository

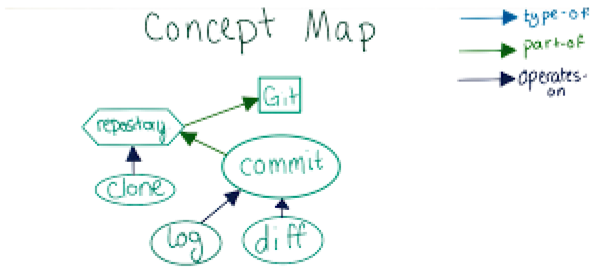
- Create a new file called lesson_2_reflections.txt
- **What happens when you initialize a repository? Why do you need to do it?**
- You might want to run git status after you create the file. You should see that now both files are listed as untracked files.

Staging Area

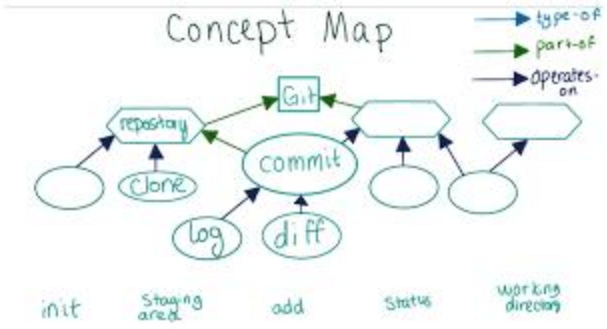
- Choosing what changes to commit
  - Working directory (your hard drive)
  - git commit doesn’t just send stuff from your working directory to the repository
  - The staging area makes it easier to choose what changes you want to commit
  - Git bundles the entire contents of the staging area into a single commit and adds that to the repository
- git add fileToAdd
  - For example:
    - git add cake-recipe.txt
    - git add frosting-recipe.txt
  - git status shows that these files are now changes to be committed
- Adding reflections to the staging area
  - Use git add to add whatever reflections files you would like to the staging area
  - Could be separate because they are for different lessons
  - Could be together, whatever you want to do.

Concept Map: init, add, staging area

- Previous Version



- Where should the new concepts go?



Reflect: Staging Area

- **How is the staging area different from the working directory and the repository? What value do you think it offers?**
- Update your document. We’ll learn how to commit it next.

Writing Good Commit Messages

- How to write a commit message
  - You’re about to make your first commit to your reflections repository.
  - You can also specify a commit message via the command line by running git commit –m “Commit message” instead of just git commit.
- Commit message style
  - Here is a link to some best practices.
  - http://udacity.github.io/git-styleguide/

Committing Changes

- To commit the changes you added, run the command git commit.
  - Git should open the editor you configured so that you can write a commit message.
    - Standard practice to write your commit message as a command. E.g. “Add a recipe” rather than “Added a recipe”
  - Save the file and quit the editor. Use git log to ensure that the change has been committed.
  - Running git status should no longer show the files you committed as untracked.
- Commit your changes in the staging area
  - git commit
- Commit all changes
  - git add
  - git status
  - git commit
- What is the output of git status after all your changes are committed?
- Git cheat sheet
  - https://github.com/github/training-kit/blob/master/downloads/github-git-cheat-sheet.pdf

Git Diff Revisited

- Fixing the bug from the asteroids repository
  - Use git diff to compare the commit with the bug to the previous one
  - We need to re-add the line this.delayBeforeBullet = 10;
  - Use git diff (without any arguments) to compare the working directory to the staging area.
    - This probably won't be something you need to do now, but with huge changes it can be a helpful way to remember what changes you made.
  - Use git diff --staged to compare the staging area to the most recent commit
    - This is a helpful way to make sure the files you've added are really what you want to commit.
- Add game.js to the staging area and commit. Also try out git diff and git diff --staged
  - If you want to discard changes to the working directory or staging area, run git reset --hard (but be careful, this action is unreversible)
    - Basically only run it after running git diff and git diff –staged to make sure there are no changes in the working directory or staging area that you want to keep.

What two versions does each form of git diff compare?

- Choices:
  - Working directory
  - Staging area
  - Commit1
  - Commit2
- git diff
  - Working directory, staging area
- git diff --staged
  - Staging area, Commit1
- git diff commit1 commit2
  - Commit1, Commit2

Commit the Bug Fix

- Leave ‘detached HEAD’ state
  - Run git checkout master
- Fix the delay bug
  - In game.js find the statement if (this.delayBeforeBullet <= 0) { (should be on line 423). On the next line, insert this.delayBeforeBullet = 10;
- Stage your change
  - Add game.js to the staging area
- Commit your change
  - Make sure to use a meaningful commit message!
- Check out this link if you’re interested in learning more about Git internals and how commit ids are generated
  - http://git-scm.com/book/en/Git-Internals-Git-Objects

Reflect: Commit Size

- **How can you use the staging area to make sure you have one commit per logical change?**
- You may want to commit your changes to the file.

Branches

- Sometimes you might want to make a commit history that branches out into multiple versions.
  - Linear makes sense for
    - Fixing bugs
    - Adding new features
    - Updating documentation
  - Branching makes sense for
    - Experimental features
    - Different versions of the same program (Spanish version)
- Master is the name given to the main branch in most repositories
  - Other branches could be called
    - Spanish
    - Exp
    - Etc…
- If you check out a branch and then make a commit, the branch label automatically updates to the new commit. It also stays checked out.
- Current last commit on a branch is known as the tip of that branch.

Making a Branch

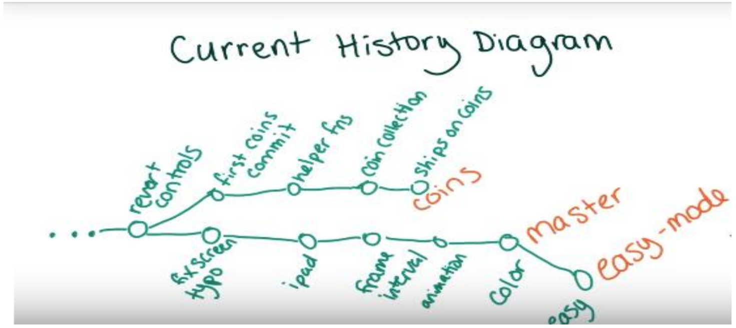
- Make an experimental change in the asteroids repository (easy version)
  - We'll make the asteroid break into only two smaller fragments instead of three.
- Running git branch (with no arguments) shows you your current branches
- Running git branch easy-mode will create a new branch with the name easy-mode
- Running git branch again will show you that easy-mode has been created
- Running git checkout easy-mode followed by git branch should change the star, which indicates the active branch.
- Change 3 to 2 in the beneath the //break into fragments comment in game.js (line 678)
- Add and commit changes making the game easier
- The output of git status, once we're done should be
  - On branch easy-mode
  - nothing to commit, working tree clean

Reflect: When to Use Branches

- **What are some situations when branches would be helpful in keeping your history organized? How would branches help?**
- You may want to commit your changes.

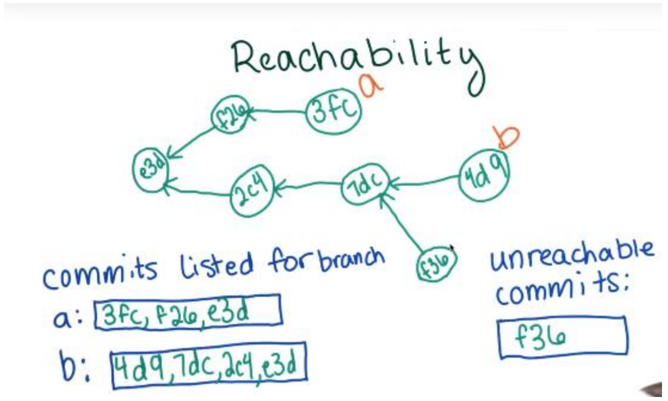
Branches for Collaboration

- If you and your collaborators all make changes on the same branch, you can't work on separate changes simultaneously.
- Commonly we will make a separate branch for every feature or bug-fix
  - Then once the changes are complete, you can either update master to point to the tip of the new branch, or merge it with the current master.
- For example, adding a coins feature to the asteroids game.
  - git checkout coins
    - Remote branch means that you didn't create the branch
  - git log, for the reason why there is not color in this version
  - git log on master will show many newer commits
- Viewing the commit history
  - Run git log --graph --online master coins



Reachability

- git log doesn't always show you every commit
  - Each commit knows about its parent
  - If you were on a branch when you made a commit, git only stores the commit ID of what was then the tip of the branch
  - Commit doesn't care about branch names
  - git log will trace through the commit history until it finds a commit without a parent (initial commit)
- Sometimes there are commits that aren't reachable from certain branches
  - So if you check out a commit, and make a new commit from there, that would move HEAD to that commit. If you then checkout an existing branch, the commit you just made would be lost unless you remember the commit ID, because it's not reachable by any of the other branches.



Detached HEAD Revisited

- Get this message when checkout out a commit, rather than a branch.
- HEAD just means current commit

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and <u>you can discard any commits you make in this state without impacting any branches </u>by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:

```
git checkout –b new_branch_name
```

Reflect: Visualizing with Diagrams

- **How to the diagrams help you visualize the branch structure?**
- You may want to commit your changes.

Combining Simple Files

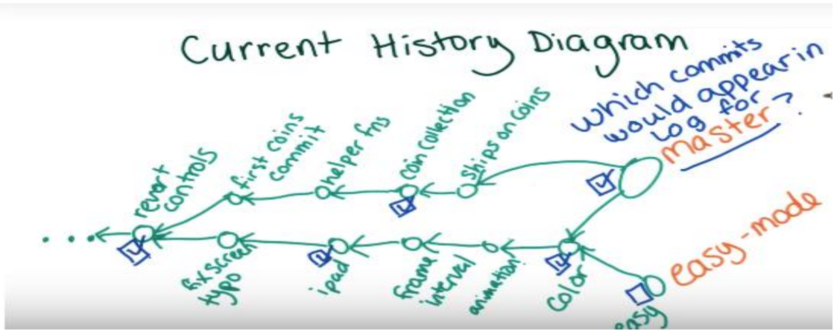
- How do we combine changes from multiple branches into a single version?
  - Could just update the label of master if no changes have been made
  - But if changes have been made to master we will have to combine both branches into one version
- Merging Files
  - Jake
    - B, D, E
  - Rachel
    - A, B, C, D
  - What should be in the final file?
    - A?, B, C?, D, E?

Combining Simple Files Using Original

- How do we solve the problem of knowing which lines to include?
  - It would help to know what the original looked like?
- Merging files
  - Original: A, B, D
  - Jake: B, D, E
  - Rachel: A, B, C, D
  - Final: B, C, D, E

Merging Coins Into Master

- Git can create a new commit that will merge two different branches together.
  - Both former branches become the parent of this commit.
  - The new commit should become the tip of the master branch “merge coins into master”
  - All of the commits from both branches get merges into the master branch as well.
- You can delete old branches after they have been merged into another branch, since all of their commits are also accessible from the branch they were merged into. (this just deletes the label)
  - But if you deleted the coins branch before merging it into master, all of those commits would become unreachable.
    - Still accessible by commit ID until Git’s garbage collection runs. It happens automatically but you can also run it manually with the command git gc.



Merging on the Command Line

- Verify you are on the master branch
  - Run git branch
- Create a merged version of the two branches
  - Run git merge master coins
  - Commit comes with a default commit message
- View the merge commit
  - Run git log
- Verify the merge worked
  - Run index.html in chrome
- Viewing the diff between a commit that is not the parent of another commit can be misleading
  - Makes it look like there were many more changes than normal
- View the diff between a commit and its parent
  - Run git show commitID
  - Will probably show a much smaller diff
- Delete the old coins branch
  - Run git branch –d coins (d stands for delete)
  - Won’t commit the commits, just the label
- Windows users might experience a merge conflict from Newline characters)
  - To resolve this run git config --global core.autocrlf True
  - More info here https://help.github.com/articles/dealing-with-line-endings/#platform-all

Reflect: Merging Two Branches

- **What is the result of merging two branches together? Why do we represent it in the diagram the way we do?**
- You may also want to commit your changes

Merge Conflicts

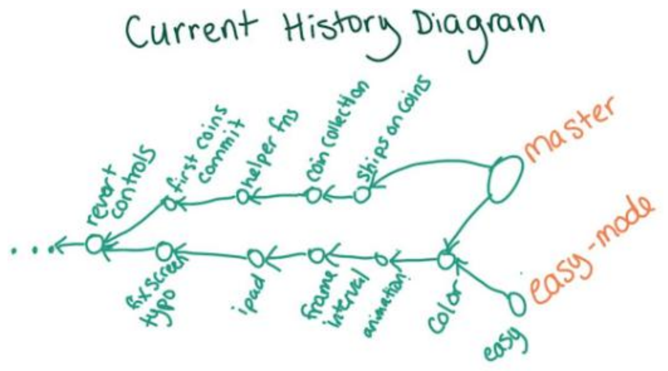
- Merging files
  - Can we always merge automatically?
    - Example:
      - Original: A, B, D
      - Jake: A, B’, D
      - Rachel: A, B’’, D
      - Final: A, B?, B’?, B’’?, D
  - We’re not sure which version should be included in the final file.

Conflict Detection

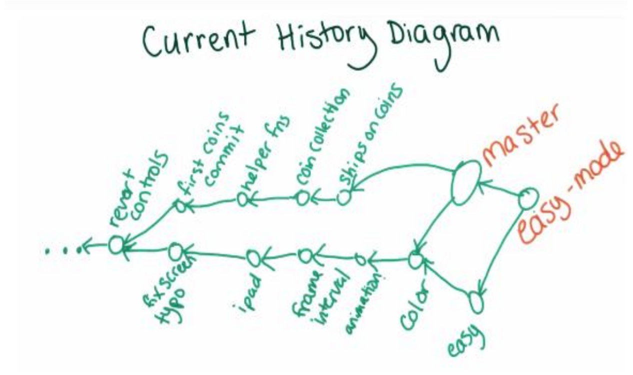
- Git just notes that these lines are different.
- Git will note a merge conflict when we start with two identical files, and there are different changes made to both of those files in the same general area.
- It would be near impossible to make Git know about all of the cases in which you might want to automerge.
- Git chooses to leave the conflicts to be resolved by the users

Update Easy Mode

- Motivation
  - Master has updated since the easy-mode branch was created.
  - If you merge master into the easy-mode branch, what will the history look like afterward?
- Previous version of the diagram



- How could we update the diagram?



- Diagramming tools
  - git log --graph
  - Gliffy https://www.gliffy.com/
  - yUML http://yuml.me/diagram/activity/draw

Resolving Merge Conflicts

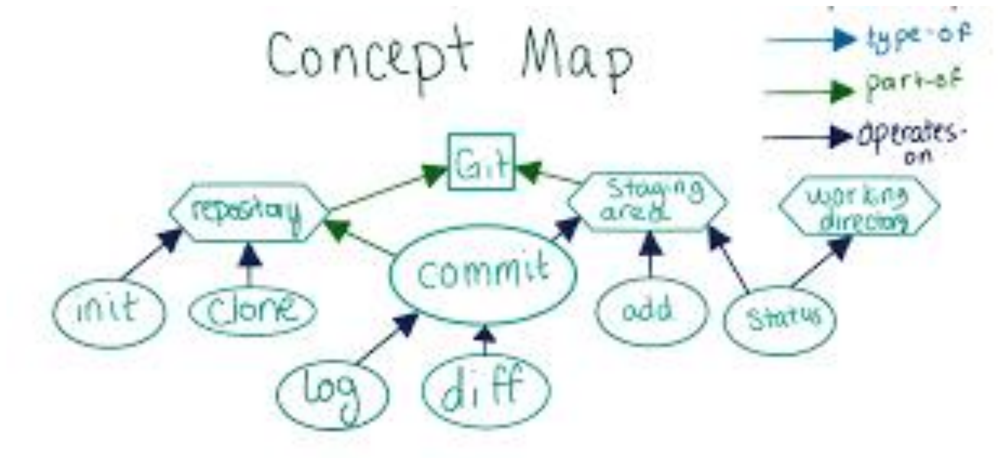
- Don’t want to include easy-mode in master, since we want to maintain two separate difficulty’s. Here’s how to do this.
  - Run git checkout easy-mode (this updates easy-mode, rather than master)
  - Run git merge master easy-mode (merge conflict)
  - Open file where the conflict is and look for:
    - <<<<<<<< HEAD
      - Your code
    - ||||||||| merged common ancestors
      - Original version that both branches modified
    - =========
      - Code in master
    - >>>>>>>> master
- How should we resolve this conflict and still keep both changes
  - Function call plus number of asteroid fragments.

Committing the Conflict Resolution

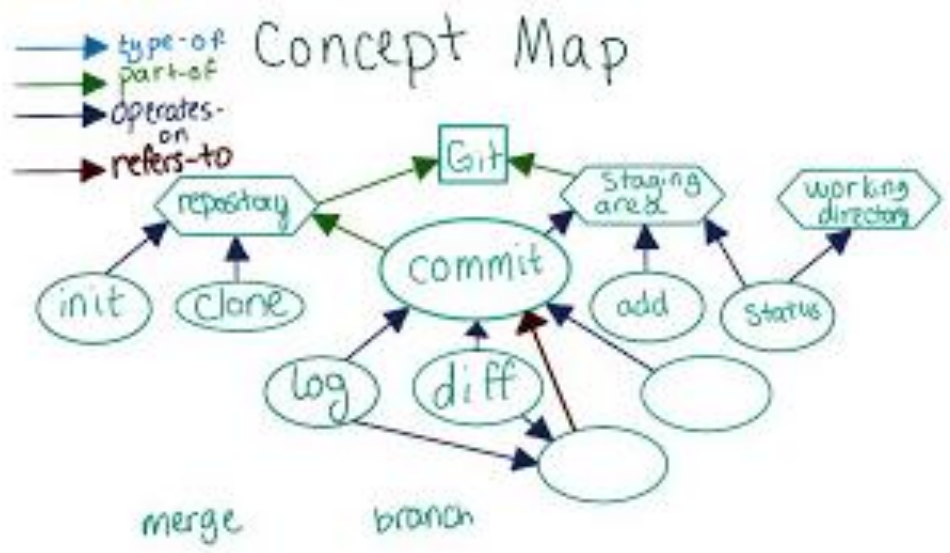
- To let Git know the conflict is resolved, you should commit the resolution
  - Save the file
  - Run git status (to ensure the file is both modified)
  - Run git add game.js
  - Run git status (conflicts are fixed, still merging)
  - Run git commit (autofilled commit message)
  - Run git log (only one new commit created)
- Resolving a merge conflict
  - Merge master into easy-mode on your computer
  - What is the output of git log –n 1?
    - (n mean that git only shows that number (1) of commits)

Concept Map: Branch, Merge

- Old Version



- New Version



Reflect: Automatic vs. Manual Merging

- **What are the pros and cons of Git’s automatic merging vs. always doing merges manually?**
- You may also want to commit your changes.

Part 2: Summary

- Create a Git repository and added commits to it
- Use the staging area to control exactly which version make it into your commit
- Maintain parallel versions of your code using Git branches and merge branches together into one combined version
- Next Up GitHub!

Part 3: Intro

- Share your changes with other people
- Bring other peoples changes into your own projects
- GitHub is one of many projects designed for this

Intro to GitHub

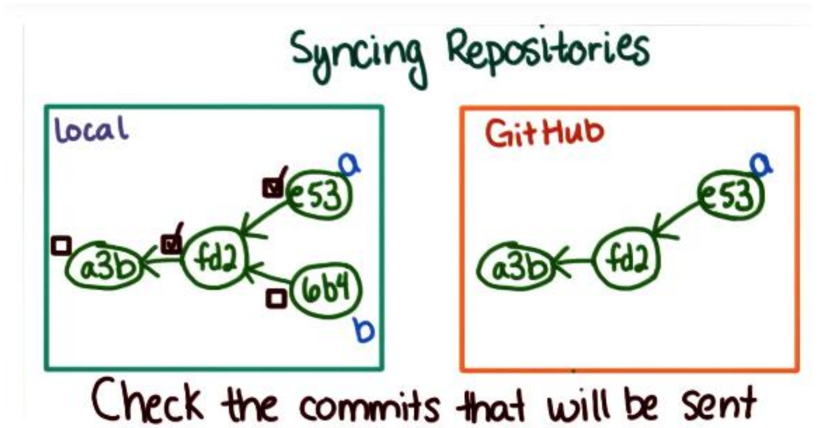
- A website that makes it easy to share an entire Git repository with other people
  - Ex: udacity/create-your-own-adventure
  - Can click around in the file structure
  - Can view the commit history
  - README displays on the main screen
  - We'll practice contributing to this project
- GitHub is home to many public projects
  - Most of which are open source
  - GitHub allows you to suggest changes that can be accepted by the owner of the project
  - Some are very small https://github.com/mathquill/mathquill (36 contributors)
  - Some are really big https://github.com/ipython/ipython (485 contributors) or https://github.com/twbs/bootstrap (853 contributors) or https://github.com/jquery/jquery (261 contributors) or https://github.com/atom/atom (357 contributors)
  - Anyone could theoretically contribute to these, but sometimes it can be hard to get your changes accepted by the person maintaining the project.
  - You can still keep your own version, even if they aren't accepted into the project.
- Private repositories cost money, but public repositories are free. This is part of GitHubs effort to promote open source.

Creating a GitHub Account

- Create a GitHub account
  - You’ll be sharing changes on GitHub, so you’ll need a GitHub account. Sign up by visiting https://github.com/ (choose the free plan)
- Set up password caching
  - Every time you send changes to GitHub via the command line, you’ll need to type your password to prove that you have permission to modify the repository .
  - If you find this annoying follow the directions here https://help.github.com/articles/caching-your-github-password-in-git to have your password autofilled.
  - Windows users should follow instructions for mysysgit.

Keeping Repositories In Sync

- Syncing Repositories
  - GitHub only hosts commits, so you have to stage and commit your changes before they can live on GitHub
  - GitHub lets you choose when to sync, so that your changes don't go public before they're ready
- An empty repository on GitHub is a prerequisite to moving your local repository to GitHub
  - Git has a concept of a remote repository
  - This remote is used for pushing and pulling data to and from GitHub
  - You could push/pull a commit, but mostly you'll want to push/pull branches
  - E.g., push master pushes all the commits in master to GitHub
  - Git doesn't send commits that are already on GitHub



Adding a Remote

- Log into GitHub
  - Select + > New Repository
  - Name the repository “reflections”
  - Leave it as public
  - Don’t initialize the repository with a README
- Add as a remote
  - Use git remote to see current remotes (empty)
  - Run git remote add origin <url> (origin is the standard name)
  - Running git remote should show that origin was created
  - Run git remote –v (v is for verbose) to see the url
  - Use git push origin master to send your changes to the remote
  - This copies all of the commits from master to origin (GitHub)
- Refresh GitHub and you should see all of your files there.
- Note:
  - Copy the HTTPS URL, not the SSH URL!
  - Go ahead and share your reflections, if you want to!

Editing Files On GitHub

- View files in the repository
- View the commit history by clicking on commits.
  - You'll only see changes that have been pushed to GitHub
- Adding content to GitHub
  - Click + to create a new file (lesson_3_reflections.txt)
  - Enter a commit message
    - Create file for lesson 3
  - Click "Commit new file"

Reflect: When to Use a Remote Repository

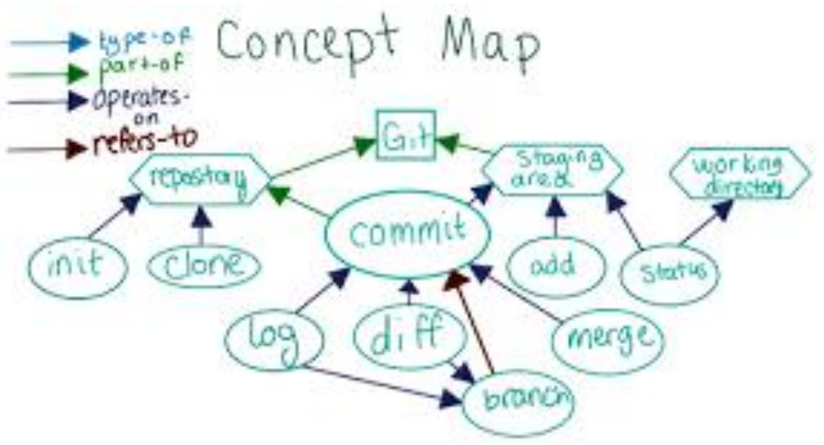
- Make changes on GitHub
  - Use the GitHub website to create a new file for your lesson 3 reflections
- **When would you want to use a remote repository rather than keeping all your work local?**
- Commit your changes.

Pulling Changes

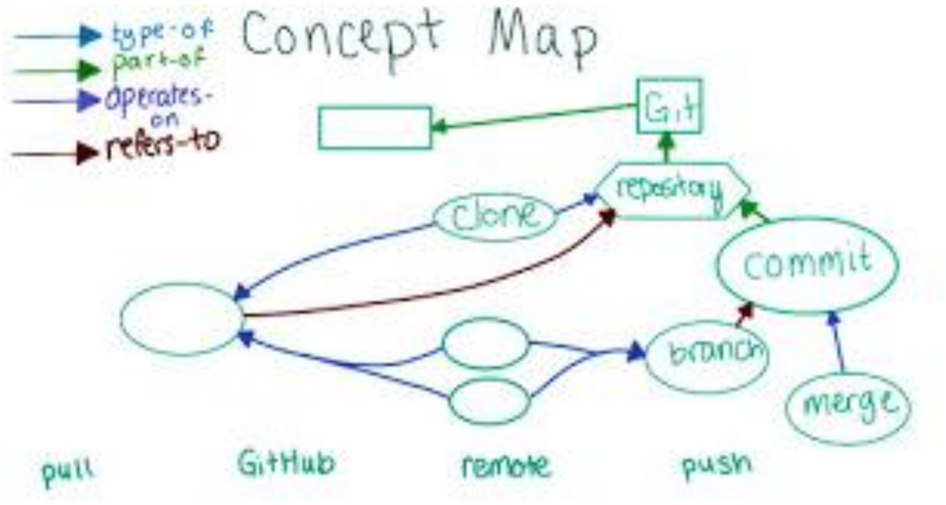
- Now your local version is out of date with the remote (GitHub) version
  - Pull master back into the local version
- Pull changes
  - Running git log shows that the most recent commit isn’t available in the local repository
  - Run git pull origin master. This will pull all commits reachable from the master branch into your local repository
  - Running git log shows that the new commit is present in your local repository

Concept Map: GitHub, Push, Pull, Remote

- Previous Version



- New Version



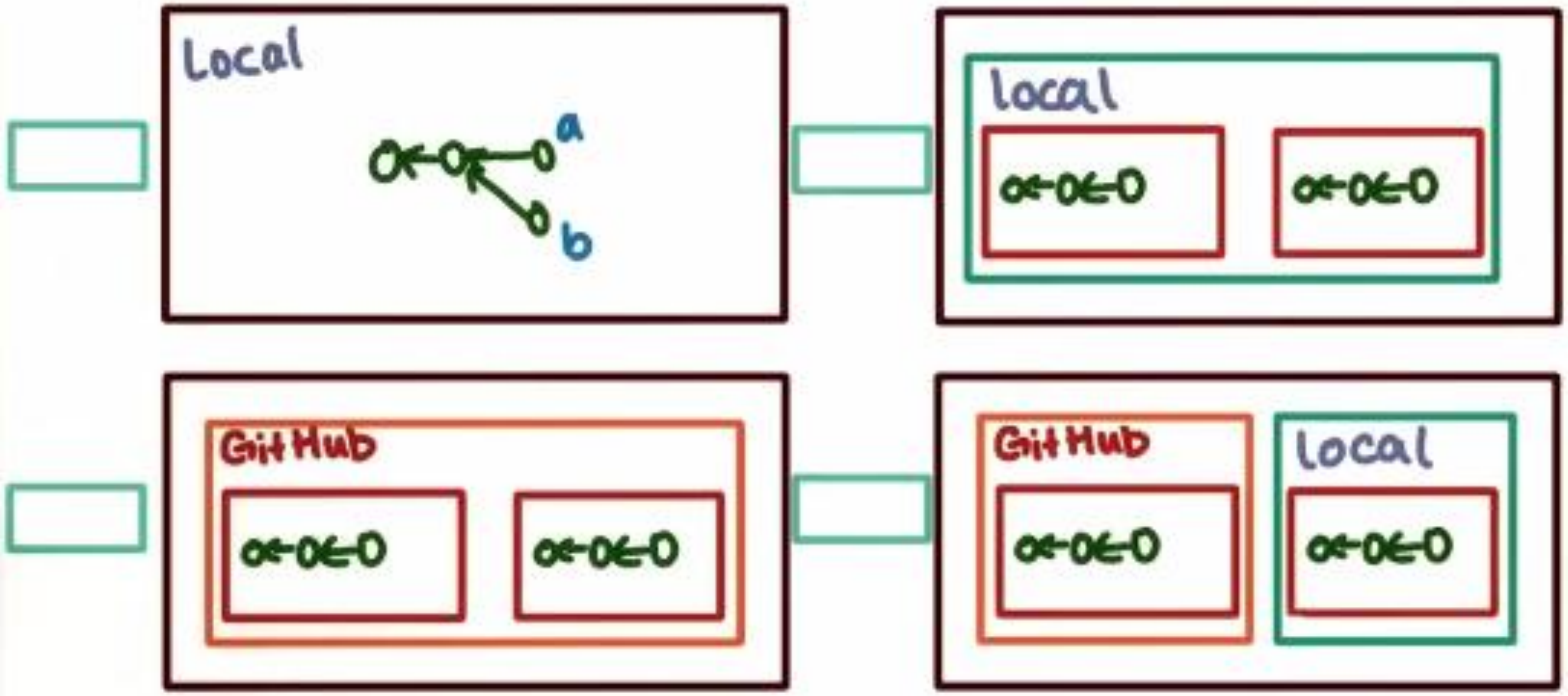
Reflect: Manual vs. Automatic Pull

- **Why might you want to always pull changes manually rather than having Git automatically stay up-to-date with your remote repository?**
- You may want to commit and push changes.

Forking a Repository

- Recipes repository
  - We should make these better
  - We could clone this repository and set it up as a remote, then create a new repository on GitHub and set up that repository as a remote and then push changes to the new remote
  - Anyone else who wants to could do this, but this is too involved and doesn’t give credit to the original author of the repository, since no one who clones a modified version would know about the original owner
- Forking a Repository
  - Copy someone else’s repository on GitHub without having to pull it down to your local machine first.
  - A fork is just a clone that GitHub makes for you, but:
    - GitHub keeps track of the number of forks
    - GitHub makes it easier to push changes back to the original repository

Clone, Branch, or Fork?



Fork the Recipes Repository

- Here is a link!
  - https://github.com/LarryMad/recipes
- Click the Fork Button
  - Now there should be a copy of the repository on your GitHub profile
- Clone the Fork to your local machine
  - Run git clone <url>
  - cd recipes
  - Git automatically sets up a remote when you clone a repository (verify with git remote –v)
- Add others as collaborators on GitHub
  - Feel free to add each other

Push Changes to the Recipes Repository

- Add a new recipe to the repository
  - On your own computer, add a new recipe for a food you like and commit it on the master branch.
- Push your changes
  - Push the master branch to your fork
- Where was your commit present?
- Right before running git push
  - Commit was present in your local repository
- Right after running git push
  - Commit was present in your local repository and on your fork (on GitHub)

Reflect: Forks, Clones and Branches

- **Describe the differences between forks, clones, and branches. When would you use one instead of another?**
- You may also want to commit and push your changes.

Collaborations Cause Conflict

- What if both repositories have commits reachable from the same branch?
  - When you pull, you want a combination of both commits from both versions
  - Just like merging, but you are merging the local and remote versions
  - This comes up when collaborating with others

Change the chili recipe

- To get conflicting changes, you’ll want changes to be made by two people – you and someone named Sarah. First, you should make your change locally. Modify the chili recipe by adding a new spice that you like to the ingredients list, and modify step 2 of the directions to mention that spice. Commit your change, but **don’t push the change to your fork yet.**
- Your branch should be ahead by one commit of origin.

Sarah changes the chili recipe

- Sarah doesn’t like cumin, so She’s going to push a change to the master branch that removes cumin from the chili recipe.
- You will see Sarah’s change in your commit history.
- GitHub is going to see this as a conflict, however, since you both changed the same line.

Simulate Sarah's Changes

- Download sarah_changes.sh here
  - https://www.udacity.com/api/nodes/3108878699/supplemental_media/sarah-changessh/download?_ga=1.34385894.672083044.1467344711
- To run it
  - cd to the directory where you saved it
  - Type bash sarah_changes.sh <url to your fork>
- Where to these commits exist
  - Commit by Larry with the message "Add a chili recipe"
    - Exists locally, Exists on your fork
  - Commit by you adding a new spice to the chili recipe
    - Exists locally
  - Commit by Sarah with the message "Remove cumin from chili"
    - Exists on your fork

Updating Local Copies Of Remote Branches

- Merging Remote Changes
  - Git stores local copies of remote branches
  - For example this could be called origin/master
  - If you commit locally, the remote branch and the local copy of your remote branch don’t get updated until you push
- You can update just your local copy of the remote, without updating your local branch, by running git fetch
  - This is nice when there are potential conflicting changes, so you can use git log and git diff to see what changes were introduced on the remote
  - You can also merge these two
  - Wow! So git pull is basically a combination of git fetch and git merge

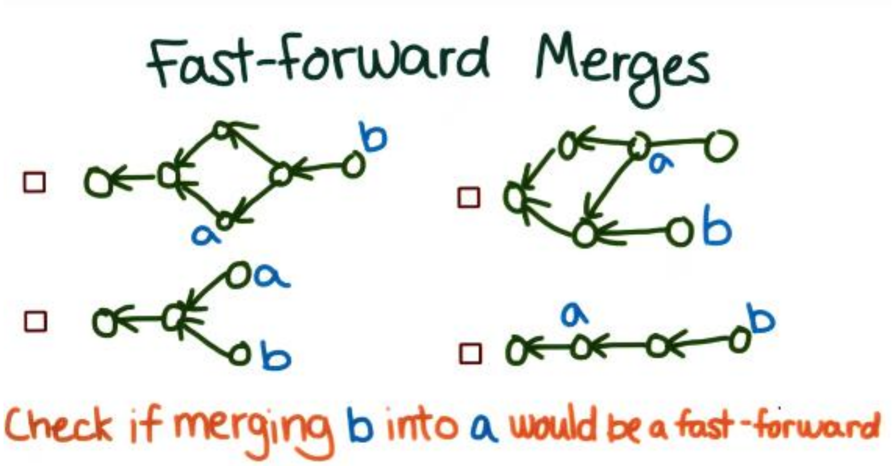
- On the command line with the recipes repo
  - Update local copies with git fetch origin
  - Inspect local copies with git log origin/master
  - Also git diff origin/master master works
- Output before and after git fetch
  - Before:
    - git log origin/master: Add chili recipe
    - git status: Ahead
  - After:
    - git log origin/master: Add chili recipe, Remove cumin
    - git status: Out-of-sync

Merging the Changes Together

- Verify origin/master contains Sarah’s changes with git status (branches have diverged)
- Ensure you have the master branch checked out and then run git merge master origin/master
  - Resolve the merge conflict
- Now add and commit the resolved chili recipe
  - Could have run git pull origin/master
- Now git status should indicate you are up to date

Fast-Forward Merges

- Question: If git pull is the same as git fetch and then merging, why haven't previous pull generated a merge commit?
- Answer: Fast-Forward Merges!
  - This happens when you merge two commits and one is an ancestor of the other.
  - So instead of creating a new commit (since commit b would already know everything about commit a), we just update the label so that they now point to the same place.
- Examples:



-

Reflect: Local Copies of Remote Branches

- **What is the benefit of having a copy of the last known state of the remote stored locally?**
- You may also want to commit and push your changes.

Making a Pull Request

- How to get feedback on your changes before you update the master branch
  - First make your change locally in a separate branch
    - E.g., git branch different-oil
    - Checkout the branch and make the change
  - Push the change to your fork (still not in the master branch)
    - E.g., add, commit and push the changes (with git push origin different-oil)
    - You can view the branch on GitHub (same list as git branch)
  - Click pull request on GitHub
    - You might need to change the base fork from the original repository to your own fork
    - Hit create pull request
    - Someone else can review your changes and merge your pull request, they can otherwise leave comments if they notice a mistake.

What gets Changed?

- edit and save README.md
  - Local working directory
- git add README.md
  - Local staging area
- git commit
  - Local master branch
- git pull origin master
  - Local master branch
- git push origin master
  - GitHub master branch
- merge alt pull request
  - GitHub master branch

Updating a Pull Request

- You’ll get an email if someone comments on your pull request
  - You can fix the issue and re commit it
  - git push origin different-oil
  - The commit message will show up under the comment
- Make a pull request
  - Create a different-oil branch
  - Make a commit different-oil changing the oil
  - Push different-oil to your fork
  - Create a pull request from different-oil into master
    - Optional: Make another commit and update the pull request

Reflect: Collaboration using Git and GitHub

- **How would you collaborate without using Git or GitHub? What would be easier, and what would be harder?**
- You may also want to commit and push your changes.

Conflicting Changes

- If someone else makes changes that conflict with your pull request
  - E.g., Sarah makes a pull request that increases the amount of oil (they are on the same line, so this is a merge conflict)
  - GitHub requires you to resolve merge conflicts on your own computer and then update the pull request with the merged version
  - Sometimes it's only acceptable to make changes with pull requests, since if you just merged into master no one else would know that master changed.
  - Update the branch, and then merge master into your branch, and push the branch to GitHub (which updates the pull request).

Updating Your Local Repository

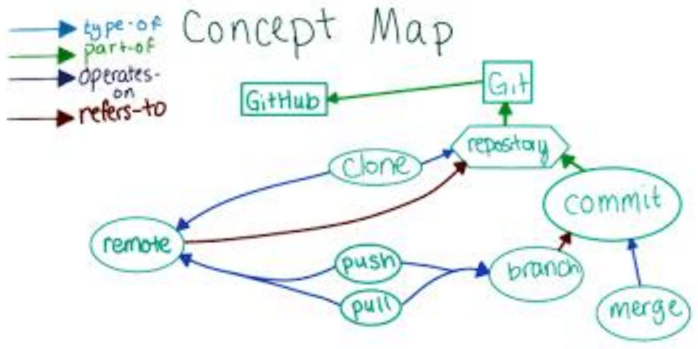
- Run git pull origin master
- Also could do
  - git fetch
  - git merge origin/master master
    - This is if you want to look at the changes before you merge them
- Incorporate the changes into the different-oil branch
  - git checkout different-oil
  - git merge master different-oil
  - Resolve merge conflict
  - Now git commit the merge
  - git log should show both changes
- https://www.udacity.com/api/nodes/3106648623/supplemental_media/sarah-changes-2sh/download?_ga=1.238512199.672083044.1467344711

Merging a pull request

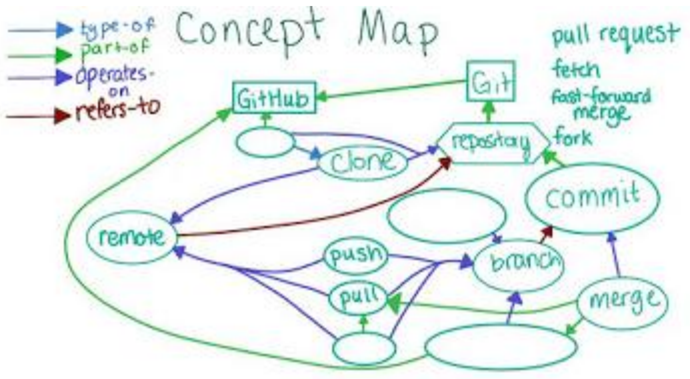
- If you finished everything from the last slide, then then “Merge” button on your pull request should be green. Now merge your pull request by pressing the button, then bring your local master up-to-date with the remote master using git pull
- One you’re done, run git log –n 1 on the master branch and check the output. Should be similar to
  - Merge: ab4b834 610bbc6
  - Author: woahboy <justintime94@gmail.com>
  - Date: Thu Apr 27 15:49:10 2017 -0500
  - Merge pull request #1 from woahboy/different-oil
  - Please take another look

Concept Map: Fork, Fetch, Pull Request

- Current version



- New Version



Reflect: When to use a separate branch

- Way more complicated
  - Rather than pulling and pushing you…
  - Pull changes into local master
  - Merge the local master into your branch (different-oil)
  - Then push your branch to the remote
  - Finally merge your branch into master (locally or not GitHub)
- Given this nightmare
  - **When would you want to make changes in a separate branch rather than directly in master? What benefits does each approach have?**
- You may also want to commit and push your changes.

Modifying the Adventure Repository

- Fork the repository and clone your fork
  - https://github.com/udacity/create-your-own-adventure
- Make a change to the story
  - Read the README in the repository for instructions
- Make a pull request
  - Click the “pull request” button from your branch like you did before, but this time, leave the original repository as the base
- Ask for your pull request to be merged
  - A bot might be able to merge your pull request automatically. If that doesn’t happen, feel free to modify your pull request so that it does. Otherwise you’ll have to wait for someone to merge the pull request. (Could take a while)
- If needed, update your pull request
  - If someone merges your pull request or leaves a comment, GitHub will email you and let you know.

Keeping a Fork Up-To-Date

- Merge Conflicts in Pull Request
  - After you fork, and then clone your fork, make your change, and push back to your fork, someone else changes the repository on GitHub
  - You need to get their conflicting changes by adding a remote. Origin points to your fork, but you need one that points to the original repository. This is usually called upstream. Then upstream/master will be added into your local repository, and you can merge it with your change.
  - Run git pull upstream master
  - Merge the master branch into your change branch
  - Push the master and change branch into the fork

Part 3 Summary

- Use remotes to push changes up to GitHub and pull changes down from other people
- Use pull request to collaborate with other people
- And now you can use Git!