# Branching

# Learning Objectives

- Understand why branching is used
- Make and checkout your first branch
- Be able to resolve merge conflicts when they occur

# Branching

- A branch is a way to make a copy of the code you're working on, or a fork that lives separately from the master branch of code
- A branch is typically used to work on a software feature, then merged back into the master branch eventually
- Branching is a core feature of git, NOT github
- However, GitHub uses git's branching feature in its software (website)

# Branching Basics

To find out what branch you're currently on:

```
$ git branch
```

To create a new branch:

```
$ git branch branch-name
```

To work on a specific branch:

```
$ git checkout branch-name
```

# Branching Basics

To push your branch up to GitHub:

```
$ git push origin branch-name
```

To pull down the latest from a specific branch into the branch you're working on:

```
$ git pull origin branch-name
```

# Exercise: Branching Basics

- Make 2 new branches for the git repo you're in called new-menu-bar and 215-fix-responsiveness
- Make commits on the branches and push both of these branches to GitHub
- Visually examine the GitHub page. Notice the "Branches" drop down and try looking at one of your branches - note the differences

# Pull Requests

Assuming that one of the branches you just made contains a change that you'd like to merge into the master branch right now, you must:

- Potentially squash all commits down to only a single commit so you remove any cruft from the git log (non-descriptive commits)
- Make sure your branch is up to date with the latest from master
- Open a pull request on GitHub so your peers can review your work
- Hope that one of them merges it into master, or leaves comments that enable you to do so later on

# Squashing Commits

- Squashing is the Git term for taking two or more commits and merging them into one.
- Typically, this is accomplished using a command like this:

```
$ git rebase -i HEAD~2
```

- This would tell Git that we'd like to interactively rebase the last two commits into one commit

# Squashing Commits

When `git rebase -i HEAD~2` a file will open in vim that you need to edit in order to continue with the rebase that looks like this:

```
pick 4acb091 Made changes to top menu. Closes #15.
pick 57f994e Whoops. Fixed bug.
```

You'll need to choose one of these commits to "squash" into the other. Using vim's insert mode, change the word `pick` to squash to choose the commit to be squashed, then quit and exit Vim using :wq

```
pick 4acb091 Made changes to top menu. Closes #15.
squash 57f994e Whoops. Fixed bug.
```

# Squashing Commits

Next up, you'll have to choose or write a commit message.

```
# This is a combination of 2 commits.
# The first commit's message is:

Made changes to top menu. Closes #15.

# This is the second commit message:

Whoops. Fixed bug.
```

You can comment out both messages with # to write a completely new one, or just comment out one message, or leave both. Either way, when you're done, quit vim using :wq.

# Squashing Commits

When all is said and done, you should get a success message from git:

```
Successfully rebased and updated refs/heads/master.
```

Also, if you check your git log, you should only have one commit where two once stood:

```
$ git log
commit c000320480932he2ibf2083r2r32hr3h2oihrioh3
Author: Zach Feldman <zach@nycda.com>
Date: Mon Apr 21 14:42:02 2014 -0400

Made changes to top menu. Closes #15.
```

# Tip: Squashing more than 2 commits

One final note: To squash more than 2 commits, just change the number in your original rebase command:

```
$ git rebase -i HEAD~4
$ git rebase -i HEAD~5
```

# Getting up to date with master

If you're about to open a pull request on a branch, you'll want to make sure that branch is up to date with master before anyone merges it

To do this, you'll want to rebase your branch with master

A rebase compares your branch with master, removes any new commits you've made, records any new commits on master, then records back the commits you've made as if you'd merged master from the latest.

# Getting up to date with master

Think of it almost like a cassette tape:

```
$ git rebase master
# rewind your branch to most recent commit you have from master
# record any new commits from master
# then, record any new commits from your branch
```

Now your branch is ready to merge with master and avoid (most) conflicts!

# Opening a pull request on GitHub

- Click on the pull request link on the right side of the repository, then click on "New pull request"
- On the left is the branch you'd like to merge into, on the right is the branch you're merging in.
- Add a descriptive title and description so the people in your repo know what you're trying to accomplish with your pull request

# Merging into master

- Typically, somebody else who is a member of your repo will review your pull request
- If it's up to snuff, they can click "Merge" to have the request automatically merged into the requested branch
- They can also make comments, after which you can add additional commits, rebase them, then push back up - these changes will then be reflected in your pull request automatically

# Merging into master

You could simply review someone's code on GitHub only before merging it, but it's best to actually check out their code and run it locally:

```
$ git clone <repo URL from GitHub>
```

or if you have a copy of the repo locally already:

```
$ git pull origin branch-name
$ git checkout branch-name
```

# Merge conflicts

- Occasionally, when rebasing a branch with master before it gets merged, we'll encounter merge conflicts
- A merge conflict is what happens when Git is unsure of how to merge together two separate versions of a project, typically because the same two files have been modified in a different way.
- The following line in the bash output after running the rebase or merge git command indicates a merge conflict:

```
CONFLICT (content): Merge conflict in <filename>
```

# Resolving merge conflicts

Never fear, merge conflicts are fairly easy to resolve!

Open up the affected project file in Sublime Text 2 or Vim and take a look:

```
<<<<<<< HEAD
<title>I like this title</title>
=======
<title>Lets do this one instead</title>
>>>>>>> Another commit.
```

# Resolving merge conflicts

```
<<<<<<< HEAD
<title>I like this title</title>
=======
<title>Lets do this one instead</title>
>>>>>>> Another commit.
```

The first line informs you that the code from that line to the ===== is in the branch you are trying to rebase with

Everything below that line (=====) to the >>> Commit message is from the code you have in the branch you're currently in.

# Resolving merge conflicts

```
<title>Lets do this one instead</title>
```

To resolve the conflict, simply remove the demarcating lines and leave the code that is meant to be in the file (based on your professional opinion)

Also test what you leave to be sure you didn't break anything!

Once the conflict is resolved, use git add to add the file to the rebase:

```
$ git add index.html
```

# Resources

## Codecademy

Learn Git - Git Branching

## Other Resources

Github for Developers

# Quiz

1. What is the purpose of branching?
2. What is the code to create a new branch and to start working on it?
3. Describe, at a high level, how to resolve a merge conflict.