Git Branching and Merging

Goals

- Create, list, and delete branches
- Explain use cases for branching
- Merge one branch with another

Branching

So far in our Git workflow we've only been working on a single branch.

But when you're working with a team, this isn't usually desirable.

What if you want to go off on your own and work on some experimental new feature?

You want to be able to work without worrying about breaking the current code or conflicting with other coders

- Typically everyone does not code on a single branch
- A branch holds a collection of commits related to a specific purpose (bug fixes, new features, deployment)
- A code base typically has multiple branches being worked on at once
- Once the purpose of the branch is complete we merge the commits of the branch to the main codebase

- After making your initial commit in a new git repo, a branch named ***main*** is generated automatically
- To see your branches, type ***git branch*** in the terminal.
- At this point, you should only see ***main***, we will see how to create a branch next
- 'An Important Note': GitHub used to call the default 'main' branch of a repository the 'master' branch. As of late 2020, GitHub has decided to rename the default branch from 'master' to 'main'. ***All new repositories have a 'main' branch, but many older repositories will still call this branch 'master'. If you are using an existing GitHub account, or are using an older version of Git, your 'main' branch might be called 'master'. We are using the 'main' branch in this set of instructions to keep up with the latest industry trends. If you have an older version of git or older repositories, simply swap out 'main' with 'master' when you are entering these commands into your terminal.***
- 'Read More About Naming Conventions': If you are curious about GitHub's decision to rename the default branch to 'main', read their documentation here - https://github.com/github/renaming <https://github.com/github/renaming>

Creating a branch

To create a new branch we use the ***git checkout*** command with the ***-b*** flag and then pass in a name of a branch.

```
git checkout -b NAME_OF_BRANCH
```

This creates the branch and moves you to the newly created branch

Also note that where you create a new branch matters

All code on the branch you were on when you created the new branch will automatically be included in the new branch code

Moving between branches

- To move back to main (or to any other branch) we use the command ***git checkout NAME_OF_BRANCH***
- If we have unsaved work on a branch we will get an error message
- The easiest way to get around this is to finish and commit the work on this branch
- We will explore different approaches later such as
  - How to ***git reset*** our code without saving
  - How to ***git stash*** the current state of our code and later retrieve it using ***git stash pop***

Deleting a branch

- Typically we will delete a branch after
  - The branch has been merged to the primary codebase
  - We decide we don't want to merge the code

In order to delete a branch we have to be currenlty be on a different branch

Then we can run the command ***git branch -d NAME_OF_BRANCH***

- We may run into error message when trying to delete if we try to delete a branch that has unsaved work
- This is git's way of warning us "are you sure you want to delete this branch, you didn't commit your work"
- To force the delete use a capital ***D*** flag in our command ***git branch -D NAME_OF_BRANCH***
- To see all of the branches we have locally, we use the command ***git branch***.
- Sometimes we also want to see the branches others are working on remotley
- To list all branches on GitHub or remote branch we pass the ***-a*** flag
- The flag does not matter right now, but it's good to get in the habit of using ***git branch -a***.

Merging

- With a branch workflow, we usually create a new branch for something we are working on (a new feature, a redesign, etc.).
- Traditionally, the ***main*** branch is reserved for production code and immediate bug fixes.
- When we are done modifying our branch, we need to ***merge*** the code into the ***main*** branch.

Here's what we're going to do:

- Create a folder called ***learn_branching*** and ***cd*** into it => ***mkdir learn_branching && cd learn_branching***.
- Initialize a ***git*** repository => ***git init***.
- Create a file called ***first.txt*** => ***touch first.txt***.
- Add that file ***git add .***.
- Commit that file ***git commit -m "initial commit"***.
- Create a new branch called ***feature*** => ***git checkout -b feature***.

Now that you are on the ***feature*** branch, create a file called ***new.txt*** => ***touch new.txt***.

- Add that file => ***git add .***.
- Commit that file => ***git commit -m "adding new.txt"***.
- Create another file called ***another.txt*** => ***touch another.txt***.
- Add that file => ***git add .***.
- Commit that file => ***git commit -m "adding another.txt"***.
- Now lets move back to the main branch using ***git checkout main***
- Note that the main branch has no awareness of ***new.txt*** or ***another.txt***!
- Merge our changes from the feature branch into the ***main*** branch => ***git merge feature***
- Delete our branch called ***feature*** => ***git branch -d feature***
- Now if you take a look at ***git log --oneline --decorate*** you'll see that the commit history on ***feature*** has ben merged into ***main***!

Git merge excercise

Your Turn

Practice makes perfect. Walk through the following steps to get more experience with the branching and merging workflow.

1. Create a folder called ***branch_time***.
2. ***cd*** into that folder.

3. Initialize an empty git repository.
4. Create a file called ***first.txt***, then add and commit the file.
5. Create a new branch called ***amazing_feature***.
6. Create a file called ***best.txt***.
7. Add the file.
8. Commit the file with the message -m “added best.txt”.
9. Switch back to the ***main*** branch.
10. Merge your changes from the ***feature*** branch into ***main***.
11. Delete the feature branch.