**Git Tag**

To recap, the git tag command is used to add a marker on a specific commit. The tag does not move around as new commits are added.

$ git tag -a beta

This command will:

* add a tag to the most recent commit
* add a tag to a specific commit *if a SHA is passed*

**Further Research**

* [Git Basics - Tagging](https://git-scm.com/book/en/v2/Git-Basics-Tagging) from the Git Book
* [Git Tag](https://git-scm.com/docs/git-tag) from the Git Docs

**Git Branch**

To recap, the git branch command is used to manage branches in Git:

*# to list all branches*

$ git branch

*# to create a new "footer-fix" branch*

$ git branch footer-fix

*# to delete the "footer-fix" branch*

$ git branch -d footer-fix

This command is used to:

* list out local branches
* create new branches
* remove branches

**Further Research**

* [Git Branching - Basic Branching and Merging](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging) from the Git Docs
* [Learn Git Branching](http://learngitbranching.js.org/)
* [Git Branching Tutorial](https://www.atlassian.com/git/tutorials/using-branches) from the Atlassian Blog

**The git checkout Command**

Remember that when a commit is made that it will be added to the current branch. So even though we created the new sidebar, no new commits will be added to it since we haven't *switched to it*, yet. If we made a commit right now, that commit would be added to the master branch, *not* the sidebar branch. We've already seen this in the demo, but to switch between branches, we need to use Git's checkout command.

$ git checkout sidebar

It's important to understand how this command works. Running this command will:

* remove all files and directories from the Working Directory that Git is tracking
  + (files that Git tracks are stored in the repository, so nothing is lost)
* go into the repository and pull out all of the files and directories of the commit that the branch points to

So this will remove all of the files that are referenced by commits in the master branch. It will replace them with the files that are referenced by the commits in the sidebar branch. This is very important to understand, so go back and read these last two sentences.

## *⚠️ Know The Branch ⚠️*

*It's very important to know which branch you're on when you're about to merge branches together. Remember that making a merge makes a commit.*

*As of right now, we do not know how to*undo*changes. We'll go over it in the next lesson, but if you make a merge on the wrong branch, use this command to undo the merge:*

*git reset --hard HEAD^*

*(Make sure to include the ^ character! It's a known as a "Relative Commit Reference" and indicates "the parent commit". We'll look at Relative Commit References in the next lesson.)*

**Merge Recap**

To recap, the git merge command is used to combine branches in Git:

$ git merge <other-branch>

There are two types of merges:

* Fast-forward merge – the branch being merged in must be *ahead* of the checked out branch. The checked out branch's pointer will just be moved forward to point to the same commit as the other branch.
* the regular type of merge
  + two divergent branches are combined
  + a merge commit is created

**Further Research**

* [Basic Merging](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging#Basic-Merging) from Git Book
* [git-merge](https://git-scm.com/docs/git-merge) from Git Docs
* [git merge](https://www.atlassian.com/git/tutorials/git-merge) from Atlassian blog

**Merge Conflict Recap**

A merge conflict happens when the same line or lines have been changed on different branches that are being merged. Git will pause mid-merge telling you that there is a conflict and will tell you in what file or files the conflict occurred. To resolve the conflict in a file:

* locate and remove all lines with merge conflict indicators
* determine what to keep
* save the file(s)
* stage the file(s)
* make a commit

Be careful that a file might have merge conflicts in multiple parts of the file, so make sure you check the entire file for merge conflict indicators - a quick search for <<< should help you locate all of them.

**Further Research**

* [Basic Merge Conflicts](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging#Basic-Merge-Conflicts) from the Git book
* [How Conflicts Are Presented](https://git-scm.com/docs/git-merge#_how_conflicts_are_presented) from the Git docs