Git Add Recap

The git add command is used to move files from the Working Directory to the Staging Index.

```
$ git add <file1> <file2> … <fileN>
```

This command:

- takes a space-separated list of file names
- alternatively, the period . can be used in place of a list of files to tell Git to add the current directory (and all nested files)

Git Commit Recap

The **git commit** command takes files from the Staging Index and saves them in the repository.

```
$ git commit
```

This command:

- will open the code editor that is specified in your configuration
    - (check out the Git configuration step from the first lesson to configure your editor)

Inside the code editor:

- a commit message must be supplied
- lines that start with a # are comments and will not be recorded
- save the file after adding a commit message
- close the editor to make the commit

Then, use git log to review the commit you just made!

What To Include In A Commit

The goal is that each commit has a single focus. Each commit should record a single-unit change. Now this can be a bit subjective (which is totally fine), but each commit should make a change to just one aspect of the project.

Let's say you want to change your sidebar to add a new image. You'll probably:

- add a new image to the project files
- alter the HTML
- add/modify CSS to incorporate the new image

Conversely, a commit shouldn't include unrelated changes - changes to the sidebar *and* rewording content in the footer. These two aren't related to each other and shouldn't be included in the same commit. Work on one change first, commit that, and then change the second one. That way, if it turns out that one change had a bug and you have to undo it, you don't have to undo the other change too.

The best way that I've found to think about what should be in a commit is to think, "What if all changes introduced in this commit were erased?". If a commit were erased, it should only remove one thing.

Further Research

- Associating text editors with Git from GitHub Help Docs
- Getting Started - First-Time Git Setup from Git book

Good Commit Messages

Here are some important things to think about when crafting a good commit message:

Do

- do keep the message short (less than 60-ish characters)
- do explain *what* the commit does (not *how* or *why*!)

Do not

- do not explain *why* the changes are made (more on this below)
- do not explain *how* the changes are made (that's what git log -p is for!)
- do not use the word "and"
    - if you have to use "and", your commit message is probably doing too many changes - break the changes into separate commits
    - e.g. "make the background color pink *and* increase the size of the sidebar"

The best way that I've found to come up with a commit message is to finish this phrase, "This commit will...". However, you finish that phrase, use *that* as your commit message.

Above all, ***be consistent*** in how you write your commit messages!

Udacity's Commit Style Requirements

You can check it out on our Git Commit Message Style Guide.

Git Diff Recap

To recap, the **git diff** command is used to see changes that have been made but haven't been committed, yet:

`$ git diff`

This command displays:

- the files that have been modified
- the location of the lines that have been added/removed
- the actual changes that have been made

Further Research

- git diff from the Git Docs

Git Ignore Recap

To recap, the `.gitignore` file is used to tell Git about the files that Git should not track. This file should be placed in the same directory that the .git directory is in.

Further Research

- Ignoring files from the Git Book
- gitignore from the Git Docs
- Ignoring files from the GitHub Docs
- gitignore.io

Globbing Crash Course

Let's say that you add 50 images to your project, but want Git to ignore all of them. Does this mean you have to list each and every filename in the .gitignore file? Oh gosh no, that would be crazy! Instead, you can use a concept called globbing.

Globbing lets you use special characters to match patterns/characters. In the .gitignore file, you can use the following:

- blank lines can be used for spacing
- # - marks line as a comment
- * - matches 0 or more characters
- ? - matches 1 character
- [abc] - matches a, b, _or_ c
- ** - matches nested directories - a/**/z matches
    - a/z

- a/b/z
- a/b/c/z

So if all of the 50 images are JPEG images in the "samples" folder, we could add the following line to .gitignore to have Git ignore all 50 images.

```
samples/*.jpg
```