**Exercise - Make changes and track them with Git**

* 15 minutes

Most development projects are iterative. You write some code, and then you test the code and make sure it works. Then, you write more code and invite other people to contribute code. Iteration means many changes: code additions, bug fixes, deletions, and replacements.

As you work on your project, Git helps keep track of the changes you make. It also lets you undo mistakes. In the next exercises, you continue building out the website you're working on and learn some important new commands, like git diff.

**Modify index.html**

The website's home page, *index.html*, currently contains just one line of HTML. Let's update it to do more, and then commit the change to Git.

1. Reopen the *index.html* file in the editor Shell (code index.html) and replace the file contents with the following HTML:

HTML

<!DOCTYPE html>

<html>

<head>

<meta charset='UTF-8'>

<title>Our Feline Friends</title>

</head>

<body>

<h1>Our Feline Friends</h1>

<p>Eventually we will put cat pictures here.</p>

<hr>

</body>

</html>

1. Save the file and close the online editor.
2. Use a git diff command to see what changed:

Bash

git diff

The output format is the same as the Unix diff command, and the Git command has many of the same options. A plus sign appears in front of lines that were added, and a minus sign indicates lines that were deleted.

The default is for git diff to compare the working tree to the index. In other words, it shows you all the changes that haven't been staged (added to Git's index) yet. To compare the working tree to the last commit, you can use git diff HEAD.

If the command doesn't return to the prompt after it executes, enter q to exit the diff view.

1. Next, commit the change. Instead of using the -a flag, you can explicitly name a file to be staged and committed if Git already has the file in the index (the commit command looks only for the existence of a file).

git commit -m "Add HTML boilerplate to index.html" index.html

1. Use git diff again to compare the working tree to the index:

git diff

This time, git diff produces no output because the working tree, index, and HEAD are all in agreement.

1. Let's say you decide "furry" sounds friendlier than "feline." Replace the two occurrences of "Feline" in *index.html* with "Furry." Then, save the file.

If you used the built-in code editor by using the code command, you won't see anything unusual. But, if you happened to use another editor, including an editor called *sed*, the editor probably created an *index.html.bak* file that you don't want to commit. Editors like Vim and Emacs create backup files with names like *index.html~* and *index.html.~1~*, depending on how they're configured.

1. Use the following command to create and open a file named *.gitignore* in the built-in code editor:

code .gitignore

1. Add the following lines to the file:

Bash

\*.bak

\*~

This line instructs Git to ignore files that have file names ending in *.bak* or *~*.

*.gitignore* is a very important file in the Git world because it prevents extraneous files from being submitted to version control. Boilerplate *.gitignore* files are available for popular programming environments and languages.

1. Save and close the editor, and then use these commands to commit the changes:

BashCopy

git add -A

git commit -m "Make small wording change; ignore editor backups"

This example uses the -A option with git add to add all untracked (and not ignored) files, and the files that have changed, to the files that are already under Git control.

If you do a git diff right now, the output will be empty because the changes have been committed. However, you can always use a git diff HEAD^ command to compare differences between the latest commit and previous commit. Try it and see. Don't forget to include the caret ^ character at the end of the command.

**Add a subdirectory**

Most websites use HTML *and* CSS style sheets, and the site you're building is no exception. Style sheets typically are stored in a subdirectory, so let's create a subdirectory named *CSS* and add it to the repo.

1. Begin by creating a subdirectory named *CSS* in the project directory:

Bash

mkdir CSS

1. Then, do a git status:

Bash

git status

Why does Git report that there's nothing to commit?

People often are surprised to learn that Git doesn't consider adding an empty directory to be a change. That's because Git tracks only changes to *files*, not changes to directories.

Sometimes, especially in the initial stages of development, you *want* to have empty directories as placeholders. A common convention is to create an empty file, often called *.git-keep*, in a placeholder directory.

1. Use the following commands to create an empty file with that name in the *CSS* subdirectory and add the contents of the subdirectory to the index:

touch CSS/.git-keep

git add CSS

1. Follow up by using git status again to check the status of the repo. Confirm that it reports one new file.

**Replace a file**

Now, let's replace *.git-keep* with a CSS file and update *index.html* to reference it.

1. Delete *.git-keep* from the *CSS* subdirectory:

rm CSS/.git-keep

1. Use the built-in editor to create a file named *site.css* in the *CSS* subdirectory:

cd CSS

code site.css

1. Add the following CSS to the file. Then, save and close the file.

h1, h2, h3, h4, h5, h6 { font-family: sans-serif; }

body { font-family: serif; }

1. Now, add the following line to *index.html* (don't forget to go back up to the *Cats* directory) after the <title> line, and save the modified file:

HTML

<link rel="stylesheet" href="CSS/site.css">

1. Use git status to see a summary of the files that have changed. Then, use the following commands to stage untracked files to version control and commit your changes to *site.css* and *index.html*:

git add .

git commit -m "Add a simple stylesheet"

Unlike most VCSes, Git records the contents of your files rather than the deltas (changes) between them. That's a large part of what makes committing, branching, and switching between branches so fast in Git. Other VCSes have to apply a list of changes to get between one version of a file and another. Git just unzips the other version.

**List commits**

Now that you have a reasonable number of changes recorded, you can use git log to look at them. As with most Git commands, there are plenty of options to choose from. One of the most useful is --oneline.

1. Use the following command to review all your commits:

git log

1. Check the output. It should look similar to this example:

Output

commit ae3f99c45db2547e59d8fcd8a6723e81bbc03b70

Author: User Name <user-name@contoso.com>

Date: Fri Nov 15 22:04:05 2019 +0000

Add a simple stylesheet

commit 4d07803d7c706bb48c52fcf006ad50946a2a9503

Author: User Name <user-name@contoso.com>

Date: Fri Nov 15 21:59:10 2019 +0000

Make small wording change; ignore editor backups

...

1. Now, use this command to get a more concise listing:

Bash

git log --oneline

1. Check the output again. This time, it should look like this example:

Output

ae3f99c Add a simple stylesheet

4d07803 Make small wording change; ignore editor backups

f827c71 Add HTML boilerplate to index.html

45535f0 Add a heading to index.html

a69fe78 Create an empty index.html file

You can see why, when you're hundreds (or thousands) of commits into a project, the --oneline option might be your best friend. Another useful option is -nX, where X is a commit number: 1 for the latest commit, 2 for the one before that, and so on. To see for yourself, try a git log -n2 command.

We've made substantial progress by using the basic functionality of Git. Next up: Step it up a level and learn how to use Git to recover from common mistakes.