Exercise Monday

Introduction to Programming (/introduction-to-programming) / Git, HTML and CSS (/introduction-to-programming/git-html-and-css) / Practice: Git Project Setup

Text Cheat sheet

Goal: In this lesson, we'll create a one-page website that says "hello" to the world. In the process, we'll learn about Git, a tool for tracking changes to our code. The steps we follow in this lesson will reflect the daily workflow we'll use when coding all of our projects throughout our time at Epicodus. You can also use the cheat sheet as a reference for starting your projects during this section — but using Git for your projects will soon become second nature.

Configuring Your Git Name and Email

On Personal Machines

In order to save code using the Git version control system, Git needs to have our name and email. On our personal machines, we can do this just **once** with a **global** configuration in the terminal:

```
$ git config --global user.name "Padma Patil"
$ git config --global user.email padma@email.com
```

A **global** configuration sets a configuration *everywhere* on a machine — not just in one directory or file. When we create a configuration for just one project directory on a machine, it's called a **local** configuration.

On Campus on Shared Computers

However, even on the shared computers on campus at Epicodus, we won't use local Git configurations. Instead, at the beginning of each class session **one** student in a pair will set up **global** Git credentials. The instructions on how to do this are the same as above. Note that these global Git credentials will be wiped when students shut down the computers at the end of every work day.

Later on, the same student who set up the global Git credentials will use other Git commands to save the local project on their remote GitHub account. Then, the other student(s) in the pair will copy that project, saving it to their own remote GitHub account. We'll go over this workflow in detail when we are ready for this step.

Getting Authorship on Shared Projects — On Campus or Remote

We pair program at Epicodus, which means there are always multiple authors for one project. To give credit to all the authors on a project, we use Git's built-in tool called **commit trailers**. We'll learn about commit trailers later on when we start making **commits**. However, we're not ready for this step — we're still working on setting up our first project!

Starting a New Project

Whenever we start a new project, we create a new project directory.

At Epicodus, we generally create new projects on the `Desktop` directory. However, you may want to create a new directory on your personal machine for storing projects as well.

If we open the terminal and enter `ls`, we'll see a list of the files and directories that in our home directory. Chances are, you won't want to store your newly-created projects in the home directory.

Creating a Project Directory

Instead, we'll want to navigate to the Desktop directory for easy access. Let's change directories from our home directory into our Desktop using the `cd` command:

```
$ cd Desktop
```

You can create a new project here, or, if you prefer, create a new directory where your projects will be stored and then `$ cd` into that directory.

Next, we'll create a project directory called `hello-world`:

```
$ mkdir hello-world
```

Remember, `mkdir` is short for *make directory*.

We can run the `ls` command to see that `hello-world` has been added to the list of directories on our Desktop (or wherever you've chosen to create your new directory).

Next, we'll move into the `hello-world` directory:

```
$ cd hello-world
```

To confirm that we're in the correct directory, we can check our location with a `pwd`:

```
$ pwd
```

Initializing Git

Before we start writing any code, we'll create a Git directory within our project directory that will track everything we add, modify and delete within this directory.

We do this by **initializing** a new Git repository:

```
$ git init
Initialized empty Git repository in /Users/staff/Desktop/he
llo-world/.git/
```

If we run `ls`, though, we won't see the new directory. Why not?

Well, if you take a look at the terminal's response to our `$ git init` command, you'll see that the following file was initialized:

```
Initialized empty Git repository in /Users/staff/Desktop/he
llo-world/.git/
```

Note that the exact **path**, the list of all the directories we'd need to navigate through to get to `hello-world`, will be different on your personal machine.

Next, note the name of the file that was created inside `hello-world`:

```
.git
```

Whenever a file has a period in front of the name, it will be **hidden**. That means they won't appear when we run the `ls` command. They also won't show up if we navigate to the directory using the point-and-click interface in the GUI (graphical user interface) of our computer.

If we want to see hidden files in the terminal, we need to add a modifier to our `ls` command:

```
$ ls -a
.git
```

The `-a` stands for *all*, so `$ ls -a` means list *all* files, even hidden ones. Modifiers added to terminal commands are also known as **flags**. There are many flags we can use to modify terminal commands.

To see all files on a Windows machine, run the following command:

```
> ls -force
```

We'll see that the `.git` directory has been created in our `hello-world` directory. Let's `cd` into this directory and take a quick peek at its contents.

```
$ cd .git
$ ls
HEAD        description info        refs
config      hooks       objects
```

These are all the files Git uses to track our project and we don't need to worry about *any* of them. In fact, we should never modify the `.git` folder because Git will take care of all tracking automatically. In general, it's common for files and directories that shouldn't be modified to be hidden — that ensures we don't accidentally modify them.

As we add, update and delete files, Git will be in the background, automatically making notes of every change in our project directory.

Let's return to the top level of our project directory by changing directories and moving up one level:

```
$ cd ..
```

Now, we are ready to add a new file to our project. This will be the HTML page that will say "Hello" to the world.

Previous (/introduction-to-programming/git-html-and-css/pair-programming-and-using-discord-when-practicing-the-git-workflow)
Next (/introduction-to-programming/git-html-and-css/practice-tracking-changes-with-git)

Lesson 7 of 64
Last updated March 24, 2023

disable dark mode