1.

Unit 3: command line and git

Introduction
Reading Time: About 2 minutes
Mark Complete

Welcome to the third unit of Build Websites from Scratch!
In this unit, you will learn how to use the command line and Git to create versions of your projects, back up your code on GitHub, and deploy your sites for everyone to see using GitHub Pages.
You'll start by learning command line navigation commands to allow you to access and modify the contents of your computer without your fingers ever leaving the keyboard.
You'll then learn how to use the command line with Git to create versions of your code so that you can look back or even restore your code to an earlier draft.
Next, you'll set up the command line and Git on your own computer so that you can apply your new skills to personal projects. You'll additionally connect your terminal to GitHub so that you can back up up your projects in the cloud and deploy them, all through command line.
Finally, you'll build and style two websites from scratch on your computer, creating versions and deploying using Git at every milestone.
By the end of this unit, you'll be able to share your projects for everyone to see and begin creating your own developer workflow. Being able to share your work with the world is the reason many people get into development, so be proud of this moment. By this point, you'll have picked up a lot of tools in a short period of time.
Good luck!

Day 1

Today you will learn how to use the command line to quickly navigate directories and modify files.

Command Line Navigation

The Command Line Interface (CLI) is a powerful tool. Many professional developers swear by their CLI, but it can be overwhelming for those who have never experienced it. In this lesson, we'll get our feet wet with the CLI by learning to navigate files and directories.

1/10
Your First Command
The command line is a text interface for your computer. It's a program that takes in commands which you type. These commands are then passed on to the computer's operating system to run.
From the command line, you can navigate through files and folders on your computer, just as you would with Finder on Mac OS or Windows Explorer on Windows. The difference is that the command line is fully text-based.
The advantage of using the command line is its power. You can run programs, write scripts to automate common tasks, and combine simple commands to handle difficult tasks — making it an important programming tool.

To access the command line, we use a terminal emulator, often just called the *terminal*.

> In the terminal, after the `$` type:
> `ls`
> and press Enter. Be sure to type the letter `l` as in "lemon" and not the number `1`.
> You should see three items print out below the command. Click 'Next' to learn how this command works.

2/10

ls

What's going on here?

```
$ ls
2014  2015  hardware.txt
```

1. In the terminal, first you see $. This is called a *shell prompt*. It appears when the terminal is ready to accept a command.
2. When you type ls, the command line looks at the folder you are in, and then "lists" the files and folders inside it. The directories **2014**, **2015**, and the file **hardware.txt** are the contents of the current directory.
   ls is an example of a *command*, a directive to the computer to perform a specific task.
   When using the command line, we refer to folders as *directories*. Files and directories on your computer are organized into a *filesystem*.
   Click 'Next' to find out how the filesystem works.

3/10

Filesystem

A filesystem organizes a computer's files and directories into a tree structure:

1. The first directory in the filesystem is the *root directory*. It is the parent of all other directories and files in the filesystem.
2. Each parent directory can contain more child directories and files. Here, **my-blog/** is the parent of **2014/**, **2015/**, and **hardware.txt**.
3. Each directory can contain more files and child directories. The parent-child relationship continues as long as directories and files are nested.
   You're probably already familiar with this tree structure — Mac Finder and Windows Explorer represent the filesystem as trees as well.
   At any point, you can reference the filesystem for this lesson here.

4/10

pwd

```
$ pwd
/home/ccuser/workspace/my-blog
```

pwd stands for "print working directory." It outputs the name of the directory you are currently in, called the *working directory*.

Here the working directory is **my-blog/**. In Codecademy courses, your working directory is usually inside the **home/ccuser/workspace/** directory.

Together with ls, the pwd command is useful to show where you are in the filesystem.

5/10

clear

What happens when you enter this command?

```
$ clear
```

The clear command clears the terminal window of any text, including previous commands and output. The terminal still knows the directory you are in and the projects you are working on. The difference is that the clear command clears any previous visible output from your window.

It is common practice for developers to clear their terminal to keep their workflow clean, and we'll be practicing this in this lesson.

6/10

cd I

In the last exercise you used this command to move into the 2015 directory:

```
$ cd 2015
```

1. cd stands for "change directory." Just as you would click on a folder in Windows Explorer or Finder, cd switches you into the directory you specify. In other words, cdchanges the working directory.
2. The directory we change into is 2015. When a file, directory, or program is passed into a command, it is called an *argument*. Here the 2015 directory is an argument for the cd command.
   The cd command switches into a directory. It takes the name of the directory as an argument.

7/10

cd II

```
$ cd jan/memory
```

To navigate directly to a directory, use `cd` with the directory's path as an argument. Here, `cd jan/memory/` command navigates directly to the **jan/memory** directory.

```
$ cd ..
```

To move up one directory, use `cd ..`. Here, `cd ..` navigates up from **jan/memory/** to **jan/**.

8/10

mkdir

```
$ mkdir media
```

The `mkdir` command stands for "make directory". It takes in a directory name as an argument, and then creates a new directory in the current working directory.

Here we used `mkdir` to create a new directory named **media/**inside the **feb/** directory.

9/10

touch

```
touch keyboard.txt
```

The `touch` command creates a new file inside the working directory. It takes in a filename as an argument, and then creates an empty file in the current working directory.

Here we used `touch` to create a new file named **keyboard.txt**inside the **2014/dec/** directory.

10/10

Generalizations

Congratulations! You've learned five commands commonly used to navigate the filesystem from the command line. What can we generalize so far?

- The *command line* is a text interface for the computer's operating system. To access the command line, we use the terminal.
- A *filesystem* organizes a computer's files and directories into a tree structure. It starts with the *root directory*. Each parent directory can contain more child directories and files.
- From the command line, you can navigate through files and folders on your computer:
- `pwd` outputs the name of the current working directory.
- `ls` lists all files and directories in the working directory.
- `cd` switches you into the directory you specify.
- `mkdir` creates a new directory in the working directory.
- `touch` creates a new file inside the working directory.

Day 2

Today you will start with a quiz to review the CLI commands you learned yesterday.

Then you will learn how to configure your personal computer's command line so that you can use the command line locally.

Finally, you will learn about a piece of software called Git, which allows you to save versions of your code. You'll learn how to use Git to create repositories, save versions of your work, and store your code in the cloud.

Setting Up Command Line

Setting Up Your Command Line

The command line is a powerful tool used by developers to find, create, and manipulate files and folders. This short tutorial will walk you through the steps for setting up the command line application on your computer.

Command Line Interfaces (CLIs) come in many forms. The CLI we'll use is called Bash.

WHAT IS BASH?

Bash, or the **B**ourne-**A**gain **SH**ell, is a CLI that was created over twenty-seven years ago by Brian Fox as a free software replacement for the Bourne Shell. A **shell** is a specific kind of CLI. Bash is "open

source" which means that anyone can read the code and suggest changes. Since its beginning, it has been supported by a large community of engineers who have worked to make it an incredible tool. Bash is the default shell for Linux and Mac. For these reasons, Bash is the most used and widely distributed shell. If you want to learn more about Bash, this Wikipedia articleis a good place to start.

BASH SETUP FOR MAC AND WINDOWS

MAC USERS:

As mentioned before, Bash is the default shell on Linux and Mac OS X, so good news, you don't have to install anything!

To access Bash in OS X, you can use an application called ***Terminal***.

1. First open the **Applications** folder, then open the **Utilities** folder.
2. Once you're in the **Utilities folder** you will see the application **Terminal**. Open the **Terminal** application and you're ready to go!
3. For ease of access later, you can keep Terminal in your Dock. Simply right click (alt-click) the Terminal icon in your dock, then select "Options", then "Keep In Dock."

Continue to the "Try it Out!" section below for some simple first steps with your new tool.

WINDOWS USERS:

Windows has a different CLI, called **Command Prompt**. While this has many of the same features as Bash, Bash is much more popular. Because of the strength of the open source community and the tools they provide, mastering Bash is a better investment than mastering Command Prompt.

To use Bash on a Windows computer, we will download and install a program called **Git Bash**. Git Bash allows us to easily access Bash as well as another tool we'll be using later called Git, inside the Windows environment.

HOW TO INSTALL GIT BASH:

1. Navigate to the Git Bash installation page and click the Download button.
2. Once Git Bash is downloaded, run the downloaded **.exe** file and allow the application to make changes to your PC. You will get a prompt that says "Do you want to allow this app to make changes to your device?" Click **Yes**.
3. To keep things simple, we will use the default settings for everything in this installation, so all you need to do now is keep clicking **Next**, and finally **Finish**.
3. 4. Open the Start menu by clicking on the Windows icon and typing "Git Bash" into the search bar. The icon for Git Bash and the words "Git Bash Desktop App" will appear. Click on the icon or the words "Git Bash Desktop App" to open Git Bash.
5. A new window will open. This is the Git Bash CLI where we will run Bash commands. Whenever a new window of the Git Bash app is opened, you will always be placed in the same directory, your **home directory**.

The home directory is represented by the tilde sign, `~`, in the CLI after `MINGW64`. The tilde is another way to say `/c/Users/username` in Git Bash or `C:\home\Users\username` in Windows' Command Prompt.

The absolute path of your current working directory, how you got from the root directory to the directory you are currently in, will always be noted at the top of the window:

Git Bash works by giving you a CLI that acts like a Bash CLI. That means you can now work with your files and folders using Bash commands instead of Windows commands.

Congratulations, you now have Bash installed on your computer, ready to use!

TRY IT OUT!

Now that you have your Command Line Interface open on your desktop, you are ready to use it. Go ahead and try some of the commands on your personal computer. Here are some good commands for practice:

`ls` to list the contents of the current directory

1. `mkdir test` to make a new directory named **test**
2. `cd test` to navigate into the new directory
3. `echo "Hello Command Line" >> hello_cli.txt` to create a new file named **hello_cli.txt** and add `Hello Command Line` to that file

4. `cat hello_cli.txt` to print the contents of the **hello_cli.txt** file to the terminal

Good job! You're ready to explore the world of the Command Line Interface on your own computer.

Git Workflow

Git is a command line tool for "version control," which will be explained shortly.

While there are graphical user interfaces for using Git, it is most powerful and flexible in its original, textual interface. Mastering Git and Command Line Interfaces will set you up for a smooth, powerful, flexible workflow for web development.

Let's "Git" to it!

1/12

Hello Git

Git is a type of versioning software that allows you to keep track of changes you make to a project as you build it. Git records changes you make to a project, stores those changes, then allows you to reference the changes as needed.

What exactly is versioning (or "version control"), and why do we use it in software development? Version control is akin to creating different drafts of a story. You would not write an entire story in one sitting. Instead, you would write multiple drafts, make changes along the way, keep final versions intact, and incorporate changes as needed. In software development, **version control** allows you to create different versions of your code as you complete each milestone.

In this lesson, we'll get started with using Git for version control. We'll do this for a screenplay called *Harry Programmer and the Sorcerer's Code*.

2/12

`git init`

Now that we have started working on the screenplay, let's turn the **ready-sorcerers-code** directory into a Git project. We do this with:

```
git init
```

This command uses the `init` keyword to *initialize* a local Git repository on your own computer.

A **repository** refers to a directory where we store the versions of our code. As you create versions of your project, your local repository will keep track of each new version.

The `git init` command sets up the Git repository and tools Git needs to track changes to the project.

3/12

`git workflow`

Nice! We have a Git project initialized as a Git repository on our local computer.

Let's take a step back to understand the Git workflow. A Git project can be thought of as having three parts:

1. A *Working Directory*: where you'll be doing all the work: creating, editing, deleting and organizing files
2. A *Staging Area*: where you'll list changes you make to the working directory
3. A *Repository*: where Git permanently stores those changes as different *versions* of the project

The Git workflow consists of editing files in the working directory on your computer, adding files to the staging area, and saving changes to a Git repository. In Git, we save changes with a *commit*, which we will learn more about in this lesson.

4/12

`git status`

As you write the screenplay, you will be changing the contents of the working directory. You can check the status of those changes with:

```
git status
```

The command prints a log of the changes that have occurred in the current version of a project.

5/12

`git add`

We'll add files for our new Git repository to track. Git doesn't know about any of the files it should track until we add them to the **staging area**. Imagine for instance, we want to make changes to **scene-1.txt** and add this new file to the Git repository.

We add the file to the staging area

```
git add filename
```
The word filename here refers to the name of the file you are editing, such as **scene-1.txt**.
6/12

git add ii
Nice! We just added a file to the staging area using git add.
Imagine we need to create several more files for the latter scenes in our screen play. Would we need to add each file individually?
Luckily, we can add all files in a project to the staging area at once by modifying the command as such:
```
git add .
```
The . here refers to all files and directories in the current directory.
When we enter this command from the root directory of a project, we add all files and directories of a project are added to the staging area at once.
7/12
git commit
A *commit* is the final step in our Git workflow. A commit permanently stores changes from the staging area inside the repository.
git commit commits of all the files in your staging area. However, one more piece of code is needed for a commit: the *option* -m followed by a message. Here's an example:
```
git commit -m "Complete first line of dialogue."
```
A commit message starts with an imperative verb and ends with a period, for example “Fix broken registration link.”
A good commit message is also written:

- in quotation marks
- in present tense
- in 50 characters or less when using -m

After committing, we can check the status of the project and see that we have no new changes. This means we created a new version of our project, and are now comparing any changes to that version.
8/12
Create a Remote Repository on GitHub
Oftentimes, you will be storing your code to a **remote repository**. A remote repository is a repository hosted on the internet or other network. This will be an identical repository, but stored in the cloud, so that if anything happens to your computer (or somebody else wants to work with your code), it will be safe and easy to access online.
GitHub is one such service for hosting remote repositories on the web. Once the repository is stored online, you can make Git repositories open source, track tasks and issues, manage projects, and collaborate with others.
We can create a remote repository through the GitHub web interface. However, much of the work you do to interact with GitHub happens through the terminal.
9/12
git remote add
Next, Git (the local repository) needs to know about the repository you created on GitHub (the remote repository) so it can store your project there.
To specify the remote GitHub repository, we add the remote and label it as the origin as such
```
git remote add origin https://github.com/your-username/your-repository-name
```
1. The **remote** is the URL of the repository that will store your project's contents.
2. The **origin** is an alias for the remote repository. This means that instead of typing the lengthy remote GitHub URL each time, you can henceforth refer to it as origin.
3. The URL
10/12
git push upstream
To push any new versions (commits) to our remote repository on GitHub, we type

```
git push -u origin master
```

Specifically, in this command:

1. `git push` informs Git that we want to update a remote repository. In other words, we want to 'push' changes from a local repository to a remote repository.
2. `origin` refers to the remote repository that we are pushing to on GitHub.
3. `master` refers to the place where our work lives on our local repository. (While we don't learn about branches in this lesson, just know that all of our work in this lesson happened on the `master` branch, and thus, we refer to our local repository as `master`).
4. `-u` stands for "upstream". By writing the command with the upstream option, we establish a link between the local repository and the remote master branch. Git now knows which remote repository to push our changes to!

   An important thing to note here, we only need to use the `-u` option the first time we push to the remote repository. We'll see how to push all successive changes in the next exercise.

11/12

git push

Imagine that we wanted to add a fourth and final scene to the screen play, and push these changes up to our remote repository. We would add our changes, commit them, and then push the code to GitHub with this command.

```
git push origin master
```

Notice this is the same command as in the previous exercise, *without* the `-u` option (for "upstream"). Since the local and remote repositories are already connected, we do not need to specify this anymore.

We can enter this command as frequently or infrequently as we like, our code will not be backed up on GitHub until we do though. It is good practice to push after every commit to make sure we don’t lose any work!

Project

Bicycle World

Cycle World

We have learned a number of commands to navigate files and directories from the command line. In this project, you'll use the commands you learned to navigate through the records of Cycle World, a bicycle shop in your neighborhood.

Mark the tasks as complete by checking them off

1.

Print the working directory.

Hint

```
pwd
```

2.

List the files and directories in the working directory.

Hint

```
ls
```

3.

Change directories to the **freight/** directory.

Hint

```
cd freight
```

4.

List the files and directories in the working directory

Hint

```
ls
```

5.

Change directories to the **porteur/** directory.

Hint

```
cd porteur
```

6.

Change directories up two levels to the **cycle-world/**directory. List the files and directories in the **cycle-world/** directory.

Hint

```
cd ../../
ls
```

7.

Change directories to the **mountain/downhill/**directory.

Hint

```
cd mountain/downhill
```

8.

Make a file called **dirt.txt**

Hint

```
touch dirt.txt
```

9.

Make a file called **mud.txt**

Hint

```
touch mud.txt
```

10.

List the files and directories in the **downhill/**directory.

Hint

```
ls
```

11.

In the **downhill/** directory, make a directory called **safety/**.

Hint

```
mkdir safety
```

12.

Change directories to the **cycle-world/** directory.

Hint

```
cd ../..
```

13.

List the contents of the **cycle-world/** directory.

Hint

```
ls
```

14.

In **cycle-world/**, make a directory called **triathalon/**.

Hint

```
mkdir triathalon
```

15.

Without changing directories from **cycle-world/**, make a file in the **triathalon/** directory called **speed.txt**.

Hint

```
touch triathalon/speed.txt
```

16.

List all files and directories in the current directory.

Hint

```
ls
```

Days 4 - 5

Reading Time: About 1 minute

Mark Complete

Today you will learn how to configure Git on your local machine and use Git and GitHub to deploy your first website to the Internet! You will then have the remainder of today and all of tomorrow to build an HTML and CSS project from scratch and incorporate your new Git skills into your workflow. We'll point out good places to create Git commits of your work and walk you through the process to help you get into the groove of committing.
As always, make sure to review any material that still isn't sticking. All of the material will keep building from here on out, so you can never review too much.
Good luck!
Article
Getting Started with Git and GitHub
Reading Time: About 8 minutes
Getting Started with Git and GitHub
In this tutorial, we walk through the process for using Git locally on your personal computer, and using GitHub to back it up. We walk through creating your personal GitHub account, setting up Git on your computer, starting your first Git repository, and connecting that repository to a GitHub repository.
This tutorial assumes that you've completed the lessons on Learn Command Lineand Learn Git. Now, prepare to use those skills on your personal computer! If some steps in this tutorial are confusing, have no fear; it will all come together by the end.
WHAT ARE GIT AND GITHUB?
This tutorial refers to Git and GitHub repeatedly. *Git* is a widely-used version control system used to manage code. Git allows you to save drafts of your code so that you can look back at previous versions and potentially undo complicated errors. A project managed with Git is called a *Git repository*. *GitHub* is popular hosting service for Git repositories. GitHub allows you to store your local Git repositories in the cloud. With GitHub, you can backup your personal files, share your code, and collaborate with others.
In short, GitHub is a tool for working with Git. There are other services to host Git repositories, but GitHub is a trusted, free service used by organizations across the world, big and small.
CREATE A GITHUB ACCOUNT
To use GitHub, you will need a GitHub account.
In your own browser:
1.Open a new browser tab
2.Navigate to https://github.com/
3.Create an account

If you already have GitHub account, continue to the next exercise.
After you sign up, you will receive a verification e-mail. Be sure to verify your e-mail address to GitHub by following the instructions in that e-mail.
GIT SETUP FOR MAC AND WINDOWS
Next, we will set up Git on your personal computer. Follow the instructions for your operating system.
MAC USERS:
1. Launch the ***Terminal*** application. You can find it in **/Applications/Utilities/**. You can also use the ***Spotlight*** search tool (the little magnifying glass in the top right of your screen) to search for ***Terminal***. Once ***Spotlight*** locates it, click on the result that says ***Terminal***.
2. When ***Terminal*** opens, type in git and press enter.
3. If you don't already have Git installed, a dialog will appear saying that "The 'git' command requires the command line developer tools. Would you like to install the tools now?" Click "Install".
Picture

Then click "Agree to the Terms of Service" when requested.

Picture

4. When the download finishes, the installer will go away on its own signifying that Git is now installed! Click "Done" to finish the installation process.

Picture

5. Navigate to GitHub's articles on setting up your Git username and email and follow the instructions for each using Terminal.
6. GitHub offers two authentication options, HTTPS and SSH, to keep your work secure. This is a security measure which prevents anyone who isn't authorized from making changes to your GitHub repository. In this article, we will use HTTPS. Navigate to GitHub's article on caching your password and follow the instructions to configure your computer to be able to use HTTPS.
Now skip down to the "Try it Out!" section below.
WINDOWS USERS:
This portion of the guide assumes you have already installed a program called Git Bash which allows us access to Git on Windows. If you have not installed Git Bash, please refer to the previous tutorial on Command Line Interface (CLI) Setup and follow the instructions for installing Git Bash on Windows. Once you complete that you can continue with this guide.
1. Open the Start menu and search for the app, git bash. You should see 'Git Bash Desktop app' appear. Press Enter or click on the Git Bash icon to open the app.
Picture

A new window will open that looks like this:

picture

This window is our CLI, where we will use our Git commands.
2. If you want to make sure that Git is installed, run `git --version` in the CLI. You should see a response that gives you the version of Git installed. It will look like this:
picture

Git can now be used in the Git Bash app!
3. Navigate to GitHub's articles on setting up your Git username and email and follow the instructions for each using Git Bash.
4. GitHub offers two authentication options, HTTPS and SSH, to keep your work secure. This is a security measure which prevents anyone who isn't authorized from making changes to your GitHub repository. In this article, we will use HTTPS. Navigate to GitHub's article on caching your password and follow the instructions to configure your computer to be able to use HTTPS.
TRY IT OUT!
Now you have everything you need to practice your Git skills on your local computer. Take a moment to run the commands below to initialize a Git repository. We will use this Git repository again later in this tutorial so make sure you complete these steps exactly as described.
1. `mkdir git_practice` to make a new directory to practice.
2. `cd git_practice` to make the new directory your working directory.
3. `git init` to turn the current, empty directory into a fresh Git repository.
4. `echo "Hello Git and GitHub" >> README.txt` to create a new README file (more on this later) with some sample text.
5. `git add README.txt` to add the new file to the Git staging area.
6. `git commit -m "First commit"` to make your first commit with the new README file.

YOUR FIRST REMOTE REPOSITORY ON GITHUB
Finally, we'll create a repository on GitHub and then link it to a local repository on your computer. This allows you to backup your work constantly and safely, so you never need to worry about losing your work again!
Now, let's connect our local Git repository to GitHub.

INSTRUCTIONS
1. In your Command Line Interface, make sure your current working directory is your new Git repository. Navigate there if not.
2. Check the status of which files and folders are new or have been edited. There should be no files modified.
$ git status
3. On GitHub, create a new repository by clicking the **New repository** button on the home page.
Picture....your repositories new repository

4. On the new repository page, give your repository a name. It's not necessary, but it would be convenient to name it the same as the directory, **git_practice**. After naming the repository, click **Create repository**.

Picture...name and description...make public

5. After creating a repository, GitHub displays the repository page. At the top of the page, make sure "HTTPS" is selected.

Picture

6.The repository is empty, so it's time to connect it to your existing work. Copy the Git commands on the GitHub page, under the title "...or push an existing repository from the command line", and paste them into your Command Line Interface. Running these commands will add a remote repository, and then push your local repository to the remote repository.
When asked for a username and password, type in your GitHub username and password and press enter after each. Don't be alarmed if you can't see the characters you are typing, they are intentionally hidden as a security measure.
Picture

Note: If you set up two-factor authentication with GitHub (don't worry if you didn't), follow these instructions to generate an OAuth token to be used instead of your password in bash. By default, GitHub does not set up two-factor authentication. If you are not familiar with two-factor authentication, you don't have to generate an OAuth token.
7. Once your Command Line Interface reports that the push is complete, refresh the page on GitHub. You should now see the text you wrote earlier in the README file, "Hello Git and GitHub."
GitHub automatically displays the contents of a file named **README.txt** if it exists in the repository. The README file is the perfect place to write a description of your project.
There you have it! Your first GitHub repository, linked to your local Git repository. You've taken some huge leaps, so be proud! Now you can use your knowledge of Git to track progress on your local computer, and push that progress to GitHub whenever you want. You can rest easy knowing that each step of your progress is safely stored in GitHub.

Stopping before remote instructions.
On git...bring up correct dir by typing in
$ cd git_practice

---

did the rest of the project....
Deploying to GitHub Pages
GitHub is a great tool to store projects and to collaborate with others, but its usefulness does not stop there. We'll use a service called GitHub Pages to share our web page creations on the World Wide Web.
WHAT IS GITHUB PAGES?
There are many different ways to deploy a website to the public Internet. We'll be using GitHub's free service called GitHub Pages.

Why GitHub Pages? GitHub Pages offers a lot of features and flexibility, all for free. Some of the benefits include:

•Easy setup
•Seamless collaboration using Git and GitHub
•Free hosting with >95% uptime
•Live updating with normal GitHub workflow

WHAT IS DEPLOYING?

Deploying is like publishing. When authors are ready for their work to be seen by the world, they publish it. When web developers are ready to share their projects, they deploy to the World Wide Web. Deployment is when a project is packaged and shared on the Internet. Unlike publishing, however, deployment may occur many, many times over the course of a software project.

DEPLOYMENT ON GITHUB PAGES

Deploying to GitHub Pages is automatic. Once it's set up, deploying happens whenever you push your local changes to your remote, GitHub-hosted repository. Head to GitHub Pages' setup instructions and follow the steps exactly to get your main GitHub Pages page setup.

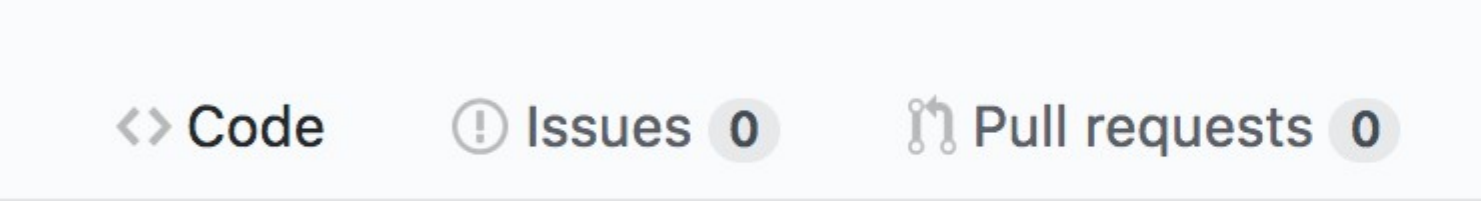
VIEWING YOUR LIVE WEB PAGE

That's it! Your website is deployed to the Internet! You and anyone with whom you share this link can view your project by navigating in your browser to the URL http://<your-github-username>.github.io.

ADDING GITHUB PAGES PROJECTS

You can set up your GitHub Pages to deploy every one of your repositories in addition to <username>.github.io. This will allow you to ensure all of your sites are deployed automatically whenever you push to GitHub.

In GitHub, navigate to your <username>.github.io repository and click **Settings**.



Within **Settings**, navigate to the **Source** section within the **Github Pages** section. From the dropdown menu, select **master branch** and then click **Save**.

Now, all of your repositories can be found at http://<username>.github.io/<repository-name>. Try creating a new repo with an HTML project inside it (perhaps push an old project to GitHub) and then navigate to the deployed page.

DEPLOYING NEW CHANGES

Now that your GitHub Pages site is set up, deploying new changes is easy. Every time you make a change to your site, use the normal GitHub flow. That is, use git commit and git push to send your changes to GitHub. After this, the GitHub site should update within a few seconds. Just refresh the page in your browser, and you're good to go!

Congratulations on your first live web page!

Project

Excursion

Mark Complete

You've been hired to create the landing page for a new mobile video app.

You'll build the web page, set up version control, and deploy it to the Internet. You're getting more comfortable with HTML and CSS, so we'll leave that up to you almost entirely. Since Git and CLI are newer concepts, we'll walk through how we use those tools a bit more carefully.

Have fun!

==========
this is the frame code for the index.html
==========

```
<!DOCTYPE html><html><head>
<link rel="stylesheet" href="resources/css/style.css"/>
</head>
<body>
<!-- "Main section" -->
<h1>Discover hidden places in the world around you</h1>

<p><a class="cta" href="#">Download Excursion (Coming soon!)</a></p>

<video autoplay="" muted="" loop="">
<source src="https://s3.amazonaws.com/codecademy-content/programs/freelance-one/excursion/videos/excursion.mp4" type="video/mp4"/>
</video>

<!-- "First supporting section" -->
<h2>Your personal travel guide</h2>

<p>Excursion remembers places you like, and recommends new points of interest around you.</p>

<p><img src="https://s3.amazonaws.com/codecademy-content/programs/freelance-one/excursion/images/camp.jpg" alt=""/></p>

<!-- "Second supporting section" -->
<p><img class="app" src="https://s3.amazonaws.com/codecademy-content/programs/freelance-one/excursion/images/phone.png"/></p>

<h2>Coming Soon for iPhone and Android</h2>

<p><a class="cta" href="#">Download Excursion (Coming soon!)</a></p>

<!-- "Footer" -->
<div class="footer">
<p>© Excursion</p>
</div>
```

This is the css.index code

```
body {
    font-family: Verdana, sans-serif;
    text-align: center;
    background-color: black;
    color: white;
}
```

```
.cta {
    color: Aquamarine;
    font-size: 16px;
}

h1 {
    font-size: 50px;
}

h2 {
    font-size: 42px;
}

p {
    color: Gray;
    font-size: 21px;
}

.footer {
    text-align: right;
}
```

Excursion Project

In this project, you'll create a web page which advertises a product called "Excursion." You'll use your growing toolset including HTML, CSS, Command Line Interface, Git, and GitHub. You'll be proud of yourself when it all comes together!

The web page we'll build advertises a mobile app which helps users record and share their experiences, so we'll use video and landscape imagery to set the scene. A landing page is a vital tool in marketing a product these days, and the goal will be to entice potential customers into using the product.

We'll work with Git and GitHub on our local machines, so if you haven't yet, refer to the articles on Command Line Interface Setup and Git Setup.

A preview of the page is available here!

Download design mock:
https://s3.amazonaws.com/codecademy-content/programs/freelance-one/excursion/mocks/excursion.png

---

use command line instead of atom to create site
bring up correct directory

```
Chaya@ChayaRauh MINGW64 ~
$ cd desktop

Chaya@ChayaRauh MINGW64 ~/desktop
$ cd projects/

Chaya@ChayaRauh MINGW64 ~/desktop/projects
$ mkdir excursion

Chaya@ChayaRauh MINGW64 ~/desktop/projects
$ cd excursion
```

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion
$ touch index.html

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion
$ mkdir resources

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion
$ cd resources

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources
$ mkdir css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources
$ cd css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources/css
$ touch style.css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources/css
$

worked fine...

cd ..
cd..
go back to excursion dir

initialize a Git repository
Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion
$ git init
Initialized empty Git repository in C:/Users/Chaya/Desktop/projects/excursion/.git/

no files are in the remote repository yet

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        index.html
        resources/

nothing added to commit but untracked files present (use "git add" to track)

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git add index.html

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git add resources/

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ cd resources

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources (master)
$ ls
css/

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources (master)
$ git add css/

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources (master)
$ cd css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources/css (master)
$ ls
style.css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources/css (master)
$ git add style.css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources/css (master)
$ cd ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion/resources (master)
$ cd  ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ cd ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects
$ cd excursion

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git commit -m "commit first excursion files"
[master (root-commit) 4c71042] commit first excursion files
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.html
 create mode 100644 resources/css/style.css
```

I am stopping at #6 setting up excursion on the GitHub

Warning: Never git add, commit, or push sensitive information to a remote repository. Sensitive information can include, but is not limited to:

•Passwords

•SSH keys

•AWS access keys

•API keys

•Credit card numbers

•PIN numbers

==========

this is what I still need to do

At the top of your GitHub repository's Quick Setup page, click to copy the remote repository URL.

1.In the Command prompt, add the URL for the remote repository where your local repository will be pushed.
git remote add origin *remote repository URL*
Sets the new remote
git remote -v
Verifies the new remote URL
2.Push the changes in your local repository to GitHub.
git push origin master
Pushes the changes in your local repository up to the remote repository you specified as the origin

---

make sure you are in the excusion directory...folder
cd until you are...

---

go to the github site and make a repository called excursion
click on the cat icon to get to the start a project page.
Then click the start a project button
when you make the excursion repository it opens in another page...
copy the url at the top of the page
https://github.com/rxr2090/excursion

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git remote add origin https://github.com/rxr2090/excursion

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git remote -v
origin https://github.com/rxr2090/excursion (fetch)
origin https://github.com/rxr2090/excursion (push)

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git push origin master
Counting objects: 5, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (5/5), 336 bytes | 168.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/rxr2090/excursion
- [new branch]   master -> master

A popup asked for my user name and password...

==========

the github page after making the new repository tells you what to do
I could have just follow those instructions:

…or create a new repository on the command line

```
echo "# excursion" >> README.md
git init
git add README.md
git commit -m "first commit"
```

```
git remote add origin https://github.com/rxr2090/excursion.git
git push -u origin master
```

…or push an existing repository from the command line

```
git remote add origin git remote add origin
git push -u origin master
```

---

at #7 set up the website html....link to the css, etc.

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="resources/css/style.css"/>
  <head>
    <meta charset="utf-8">
    <title>Excursion</title>

  </head>
  <body>
      <h1>Discover hidden places in the world around you</h1>
      <h3>Download Excursion (Coming soon!)</h3>

  </body>
</html>
```

```
body {
    font-family: Verdana;
    text-align: center;
    background-color: black;
    color: white;
}
h1 {
    font-size: 50px;
    font-weight: normal;
}

h2 {
    font-size: 42px;
}
h3 {
    color: Aquamarine;
    font-size: 16px;
    text-align: center;
    text-decoration: underline; /* change to hyperlink later */
}
```

#8

record changes to existing files

```
git add .
git commit -m "message"
```

#9

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="resources/css/style.css"/>
  <head>
    <meta charset="utf-8">
    <title>Excursion</title>

  </head>
  <body>
      <h1>Discover hidden places in the world around you</h1>
      <h3>Download Excursion (Coming soon!)</h3>
      <video autoplay="" muted="" loop="">
           <source src="./resources\videos\excursion.mp4" type="video/mp4"/>
         </video>
  </body>
</html>

body {
    font-family: Verdana;
    background-color: black;
    color: white;
    text-align: center;
}
h1 {
    font-size: 50px;
    font-weight: normal;
    text-align: center;
}

h2 {
    font-size: 42px;
}
h3 {
    color: Aquamarine;
    font-size: 16px;
    text-align: center;
    text-decoration: underline; /* change to hyperlink later */
}

#10
Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git add .

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git commit -m "added cellphone video"
[master e57ecf5] added cellphone video
 5 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 resources/images/cellphone.png
 create mode 100644 resources/images/landscape.jpg
 create mode 100644 resources/videos/excursion.mp4

only changed html to finish project:
```

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="resources/css/style.css"/>
  <head>
    <meta charset="utf-8">
    <title>Excursion</title>

  </head>
  <body>
      <h1>Discover hidden places in the world around you</h1>
      <h3>Download Excursion (Coming soon!)</h3>
      <!-- "create into a hyperlink when download available" -->
      <video autoplay="" muted="" loop="">
           <source src="./resources\videos\excursion.mp4" type="video/mp4"/>
         </video>

      <h2>Your personal travel guide</h2>
      <p>Excursion remembers places you like, and recommends new points of interest around
you.</p>
      <p><img src=""./resources\images\landscape.jpg" alt=""/></p>
      <p><img class="app" src="./resources\images\cellphone.png"/></p>

      <h2>Coming Soon for iPhone and Android</h2>

      <h3>Download Excursion (Coming soon!)</h3>
      <!-- "create into a hyperlink when download available" -->
  </body>
</html>
```

#11.
Push your local changes to your repository on GitHub
http://<username>.github.io/<repository-name>
http://rxr2090.github.io/<repository-excursion>

I did not get anything from the above....
tried this....

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git push -u origin master
```

that code generated the following

```
Counting objects: 28, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (22/22), done.
Writing objects: 100% (28/28), 15.79 MiB | 1.12 MiB/s, done.
Total 28 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), done.
To https://github.com/rxr2090/excursion
   4c71042..654a984  master -> master
Branch master set up to track remote branch master from origin.
```

That worked:

https://rxr2090.github.io/excursion/

add footer to html and css

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="resources/css/style.css"/>
  <head>
    <meta charset="utf-8">
    <title>Excursion</title>

  </head>
  <body>
      <h1>Discover hidden places in the world around you</h1>
      <h3>Download Excursion (Coming soon!)</h3>
      <!-- "create into a hyperlink when download available" -->
      <video autoplay="" muted="" loop="">
           <source src="./resources\videos\excursion.mp4" type="video/mp4"/>
         </video>

      <h2>Your personal travel guide</h2>
      <p>Excursion remembers places you like, and recommends new points of interest around
you.</p>
      <p><img src=""./resources\images\landscape.jpg" alt=""/></p>
      <p><img class="app" src="./resources\images\cellphone.png"/></p>

      <h2>Coming Soon for iPhone and Android</h2>

      <h3>Download Excursion (Coming soon!)</h3>
      <!-- "create into a hyperlink when download available" -->

      <div class="footer">
           <p>© Excursion</p>
      </div>
  </body>
</html>
```

```
body {
    font-family: Verdana;
    background-color: black;
    color: white;
    text-align: center;
}
h1 {
    font-size: 50px;
    font-weight: normal;
    text-align: center;
}

h2 {
    font-size: 42px;
    font-weight: 300;
}
```

```
h3 {
    color: Aquamarine;
    font-size: 16px;
    text-align: center;
    text-decoration: underline; /* change to hyperlink later */
}
.footer {
    text-align: right;
}
```

then git bashed the final step-by-step

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git add .

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git commit -m "added footer"
[master bb6849f] added footer
 2 files changed, 7 insertions(+)

Chaya@ChayaRauh MINGW64 ~/desktop/projects/excursion (master)
$ git push -u origin master
Counting objects: 6, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 658 bytes | 329.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/rxr2090/excursion
   654a984..bb6849f  master -> master
Branch master set up to track remote branch master from origin.
```

Excursion final project is online at: https://rxr2090.github.io/excursion/

Days 6 - 7
Reading Time: About 1 minute
Completed

Over the next two days, you will build and deploy a website from the scratch, making Git commits along the way. This is a common developer workflow.
As you make your way through this project, you may find yourself getting into the groove of development. This is an awesome achievement, and you should be proud of all you've accomplished. As always, you may run into some tricky problems along the way in this project. Keep reviewing what you know and using all of the tools at your disposal. Soon you'll find yourself with another great final product.
Good luck!
Project
Broadway
Mark Complete

You've been selected to create the web page for a new design firm named Broadway.

You'll build the web page using all the tools you've learned thus far. We've set up this real-world scenario, but the development choices are in your hands.

Broadway

In this project, you'll create a website, Broadway, an image of which is provided in the steps below. You are to use your knowledge of HTML & CSS to recreate the web page, with several important considerations:
1. The website will look like this. You can also find a detailed description of the styles in these redline mocks.
2. You are to use Git and GitHub for version control, to initialize the Git repository, commit each change, and push the changes to GitHub.
3. You are to use the Command Line Interface for each file or directory you create for the project.
4. You will find the images used in this site here:

- The background image to be used in two places, at the top of the page and then again further down the page under the word "Broadway.'
- The identify, understand, and execute icons.

This is not the entire line command...the last project is not here.

This is now the last project in command line Unit 3

go back to page 14 and do what you did in the excursion project.

Make sure you are in the projects folder/directory

Chaya@ChayaRauh MINGW64 ~
$ cd desktop

Chaya@ChayaRauh MINGW64 ~/desktop
$ cd projects

Chaya@ChayaRauh MINGW64 ~/desktop/projects
$ mkdir broadway

Chaya@ChayaRauh MINGW64 ~/desktop/projects
$ cd broadway

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway
$ touch index.html

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway
$ mkdir resources

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway
$ cd resources

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources
$ mkdir css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources
$ cd css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources/css
$ touch style.css

go back to broadway folder/directory

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources/css
$ cd ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources
$ cd ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway
$

notice that the last word is broadway so you are in the broadway folder/directory
now do all the git stuff

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources/css
$ git add style.css
fatal: Not a git repository (or any of the parent directories): .git

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources/css
$ cd ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources
$ cd ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway
$ git init
Initialized empty Git repository in C:/Users/Chaya/Desktop/projects/broadway/.git/

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        index.html
        resources/

nothing added to commit but untracked files present (use "git add" to track)

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git add index.html

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git add resources/

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ cd resources

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources (master)
$ git add css/

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources (master)
$ cd css

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources/css (master)
$ git add style.css
```

go back to the broadway folder/directory

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources/css (master)
$ cd ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway/resources (master)
$ cd ..

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$
```

broadway is the last word in the location.

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git commit -m "commit first broadway files"

it generated the following lines:
[master (root-commit) a24d7ed] commit first broadway files
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.html
 create mode 100644 resources/css/style.css
```

leave the command line....do this...

go to the github site and make a repository called broadway
if you do not see...start project ...then....
click on the cat icon (at the top) to get to the start a project page.
Then click the start a project button
when you make the broadway repository it opens in another page...
copy the url at the top of the page

mine is:
https://github.com/rxr2090/broadway

return to the command line:

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git remote add origin https://github.com/rxr2090/broadway

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git remote -v

gave these lines:
origin  https://github.com/rxr2090/broadway (fetch)
origin  https://github.com/rxr2090/broadway (push)

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git push origin master

gave these lines:  notice that this did not work but it did in excursions...
To https://github.com/rxr2090/broadway
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/rxr2090/broadway'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$
```

the git push command failed...will try a few things and post what worked for me...
the -f is a fetch. The remote had a master but my local did not have a master.
This did something but not enough to correct the problem.

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git remote add -f  rxr2090  https://github.com/rxr2090/broadway

generated the following lines:
Updating rxr2090
From https://github.com/rxr2090/broadway
 * [new branch]      master     -> rxr2090/master

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$
```

No...it is not fixed.

Tried this...

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git pull --rebase https://github.com/rxr2090/broadway

generated these lines:
From https://github.com/rxr2090/broadway
 * branch            HEAD       -> FETCH_HEAD
First, rewinding head to replay your work on top of it...
Applying: commit first broadway files
```

this worked...now can do the git push command

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git push origin master

generated the following lines which is what should have happened:
Counting objects: 5, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (5/5), 407 bytes | 203.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/rxr2090/broadway
   fd62c51..5357acc  master -> master

the next step is to write the html and style css pages.
Create the website...

I will copy the code for the last project and then modify it for this red line specs.

Have not fully checked the red line specs....

<!DOCTYPE html>
<html>
<link rel="stylesheet" href="resources/css/style.css"/>
  <head>
    <title>Broadway</title>

  </head>
  <body>
      <h1>Broadway Design</h1>


<div><h3>Home</h3></div><div><h3>About</h3></div><div><h3>Work</h3></div><div><h3>Team</h3>
</div><div><h3>Contract</h3></div>
      <!-- "create into a hyperlink when download available" -->
      <br><br>
      <h4>Our Process</h4>

      <p><img src="./resources\identify.svg" alt=""/></p>
      <h3>Identify</h3>
      <p>We focus on the most important details.</p>

      <p><img src="./resources\understand.svg" alt=""/></p>
      <h3>Understand</h3>
      <p>We study our projects inside and out.</p>
```

```
      <p><img src="./resources\execute.svg" alt=""/></p>
      <h3>Execute</h3>
      <p>We put everything we have into our work.</p>

      <h2>Designed in New York</h2>

      <h3>Portfolio</h3>
      <p>Coming Soon</p>

      <h3>Team</h3>
      <p>We're Hiring!</p>

      <h3>Contact</h3>
      <p>Phone: (555)-555-5555</p>
      <p>E-mail: test@example.com</p>
      <p>Twitter: @test</p>

      <p><img class="app" src="./resources\images\cellphone.png"/></p>



      <div class="footer">
          <span class="nowrap"><p>© Broadway 2017  
<a href="#">Home</a> 
<a href="#">About</a> 
<a href="#">Team</a> 
<a href="#">Contact</a> 
          </span></p>
      </div>
  </body>
</html>


body {
   font-family: Helvetica;
   text-align: center;
   color: black;
   background-image: url(https://s3.amazonaws.com/codecademy-content/programs/freelance-
one/broadway/images/the-flatiron-building.png);
   background-size: 100% 28%;
   background-repeat: no-repeat;

}
ul {
   color: white;
   font-size: 18px;
   text-decoration: underline;}
h1 {
   text-align: center;
   font-size: 70px;
   color: white;
   font-weight: bold;
  }

h2 {
   font-size: 23px;
   font-weight: 300;
   background-image: url(https://s3.amazonaws.com/codecademy-content/programs/freelance-
one/broadway/images/the-flatiron-building.png);
   background-size: 100% 70px;
   color: white;
}
h3 {
```

```
    font-size: 16px;
    text-decoration: underline; /* change to hyperlink later */
}

h4 {
    font-size: 32px;
  }

  p {
      font-size: 14px;
    }
  div {
    color: white;
  }
.footer {
    text-align: left;
    font-size: 11px;
    background-color: gray
}
.nowrap {
  white-space: nowrap;
}

a { color: white;
font-size: 16px;
}
```

Do final git code

be sure you are in broadway folder

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects
$ cd broadway

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git push origin master
Everything up-to-date
```

`http://<username>.github.io/<repository-name>`

https://rxr2090.github.io/

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git push origin master
Everything up-to-date

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git add .
warning: LF will be replaced by CRLF in resources/execute.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in resources/identify.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in resources/understand.svg.
The file will have its original line endings in your working directory.
```

```
Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git commit -m "maybe done"
[master 39da74c] maybe done
 7 files changed, 141 insertions(+)
 create mode 100644 resources/broadway_redline.jpg
 create mode 100644 resources/execute.svg
 create mode 100644 resources/identify.svg
 create mode 100644 resources/the-flatiron-building.png
 create mode 100644 resources/understand.svg

Chaya@ChayaRauh MINGW64 ~/desktop/projects/broadway (master)
$ git push -u origin master
Counting objects: 11, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (11/11), 1.14 MiB | 903.00 KiB/s, done.
Total 11 (delta 0), reused 0 (delta 0)
To https://github.com/rxr2090/broadway
   5357acc..39da74c  master -> master
Branch master set up to track remote branch master from origin.
```

You have to go to the github website to your ….broadway

go to setting...then go down the page...the GitHub Pages section
at source...click on Master Branch and save
dont do any theme...
tit is published when you choose master branch and save...
a line should appear that is similar to
Your site is ready to be published at https://rxr2090.github.io/broadway/.

And it does come up...
https://rxr2090.github.io/broadway/