

# Learning Objectives: GitHub

Learners will be able to...

- Differentiate git and GitHub
- Differentiate a local and remote repository
- Describe the complete process of pushing content to a GitHub repo

info

## Make Sure You Know

You are comfortable using the terminal.

## Limitations

We use a combination of a browser, command line commands, and GUI tools in Codio to get our work from Codio to a repo on GitHub. This is not a course on using git in an in-depth manner.

# What is GitHub?

## Understanding Git

When people talk about GitHub, they are really talking about git. The two things are very similar, but nonetheless different. Let's start by defining git. **Git** is a command line tool used to track changes to files. Git was developed by Linus Torvalds, who wrote the kernel for the Linux operating system. Git is most often associated with software developers, but it can work with any file type.

The most common use of git is to back up your work. In fact, that is what we are going to do with it. Every time you save your work with git, it stores a date and time stamp as well. This means we can roll back to any previously saved snapshot. But it is important to note that git does so much more than back up code.

In addition, other developers on your team can maintain a separate copy of the code (called a branch). Branches can be merged back into the main project. Developers outside your organization can submit code to be included in the project (called a pull request). Git provides important features that go above and beyond simple backups.

## Understanding GitHub

The problem with git is that it is not user-friendly. Everything is done through the command line. Because it is so powerful, git commands are long and have lots of options. The end result is that new users have a hard time getting comfortable using git. Enter **GitHub**.

GitHub (now owned by Microsoft) is a company that provides a user-friendly service built on git. They have a website and desktop app that gives you the ability to interact with git while not using the command line (you can still use the command line if you wish). GitHub also provides cloud hosting for your work. So when we save our Django project to GitHub, it will be stored on a server run by Microsoft. You can search the large collection of public projects on GitHub, make copies of projects, leave stars on projects you like, leave comments for other users, etc.

Git is the underlying technology developers use to manage their software projects. GitHub is a collection of services built on top of git. GitHub is the most popular tool developers use to manage their work. So git and GitHub are often used interchangeably, even though they are different in a few important ways.



# Connecting to GitHub

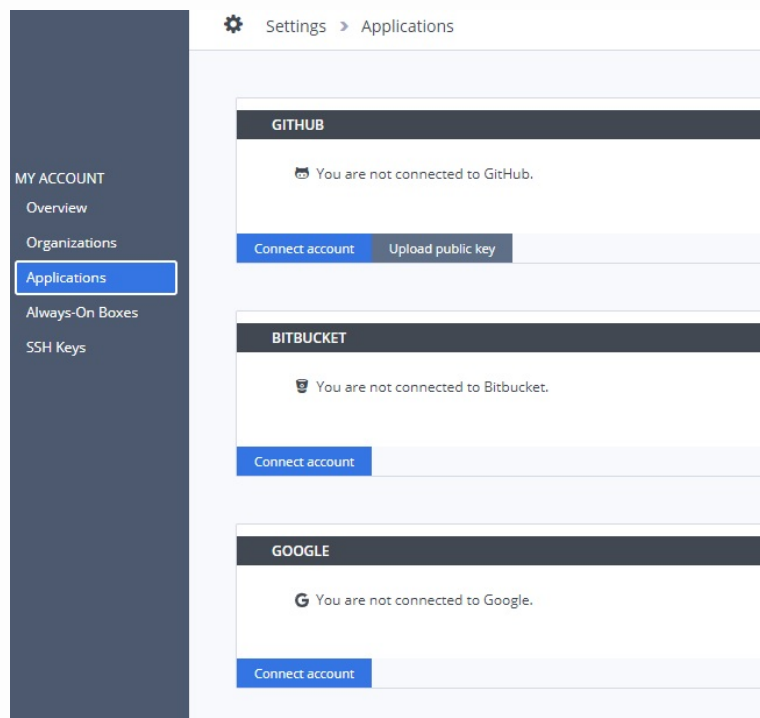
## GitHub Account

This assignment assumes that you have a GitHub account. If you do not have one, click on [this link](#) and make one. GitHub offers paid and free accounts. A free account is sufficient for our needs. Please verify that you have a valid GitHub account before continuing.

## Connecting GitHub and Codio

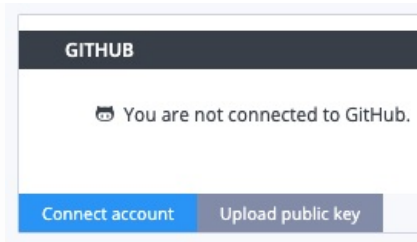
In order for Codio to talk to your GitHub account, you need to [connect](#) GitHub to your Codio account. This only needs to be done one time.

- In your Codio account, click on your username
- Click on **Applications**



Codio Account

- Under GitHub, click on **Connect account**



connect account

- You will be using an SSH connection, so you need to click on **Upload public key**

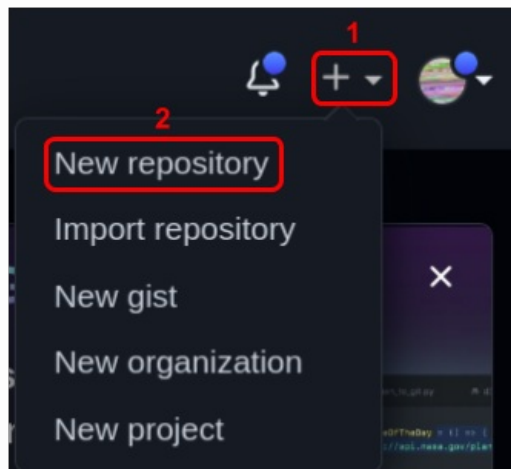
# Repositories

## What is a Repository?

GitHub uses the term repository (also referred to as a repo) for a named location where you store files and folders for your software projects. Repositories are the organizational building blocks of GitHub. If you want to push something to GitHub, it needs to be in a repo. If you want to share a project with somebody else, you direct them to a repo. As you build out your GitHub portfolio, you will add to an ever-growing collection of repos.

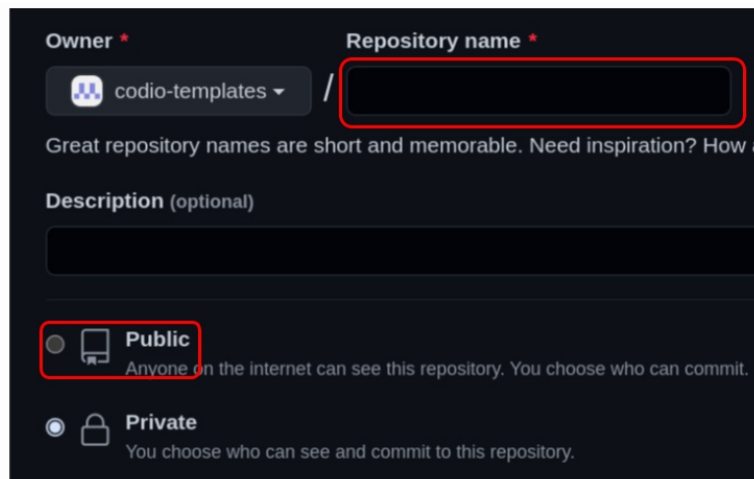
## Create a Remote Repository

Before using GitHub, you need to first create a repo. Go to the browser tab with Github. In the top-right corner of the page, you should see a plus sign (+). Click it. The click “New repository” from the list.



The image depicts part of the menu when you click the plus sign icon. There is a red box around the plus sign and a number 1 since this is clicked first. There is another red box around the “New repository” button with the number 2 since it is pressed second.

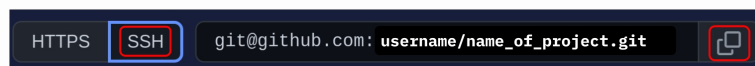
On the next page, give your repo the name `movie_review`. If GitHub does not default to a public repo, you should changing it to public. A public repo does not mean other people can alter your work. Instead, it means that other can see your work. They can fork the repo, which makes a copy for them, but they cannot alter your code.



The image depicts a portion of the new repository page. There is a red box around the field for the repository name. There is also a red box around the radio button for public visibility.

After your repo is complete, GitHub will provide an address for it.

**Important**, make sure that you select the SSH address. You can tell if it an SSH address because it should start with `git@github.com:`. Once you have selected the SSH address, click the button on the right to make a copy of it. We need this address to connect your GitHub repo to your Django project.

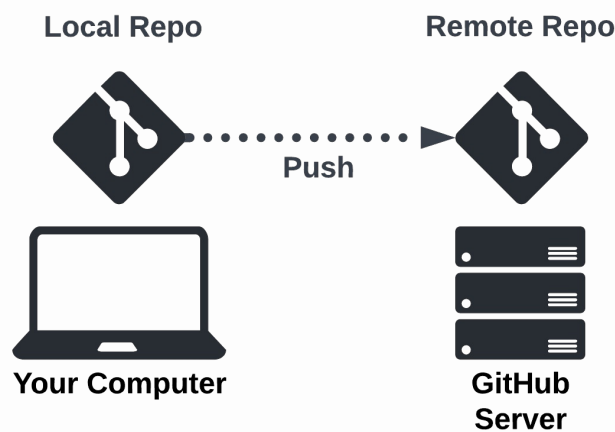


The image depicts the SSH address of the repository. There is a red box around the the SSH button and a red box around the copy icon.

# Local vs Remote

## Local and Remote Repositories

There are two kinds of repos that concern us — **local** and **remote** repos. A local repo lives on your computer, or in our case Codio. A remote repo lives on another computer, in our case a GitHub server. We previously talked about how we are using GitHub as a backup for our work. That means we need to push our local repo to our remote repo.



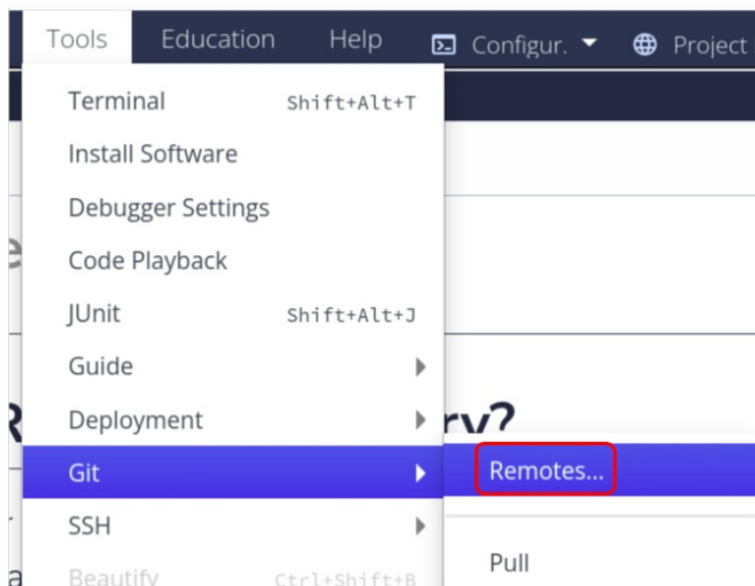
This image depicts a local repository which resides on your computer. It also shows a remote repository which resides on a GitHub server. The act of pushing moves the code from the local repository to the remote repository.

## Adding Our Remote Repository

You should have clicked on the icon to copy the SSH address of our newly created repo on GitHub. If not, go to the tab with GitHub and copy the address that starts with `git@github.com:`. This will be our remote repo.

Once you are back in Codio, click **Tools** in the menu bar. Hover over **Git** and then select **Remotes . . .**





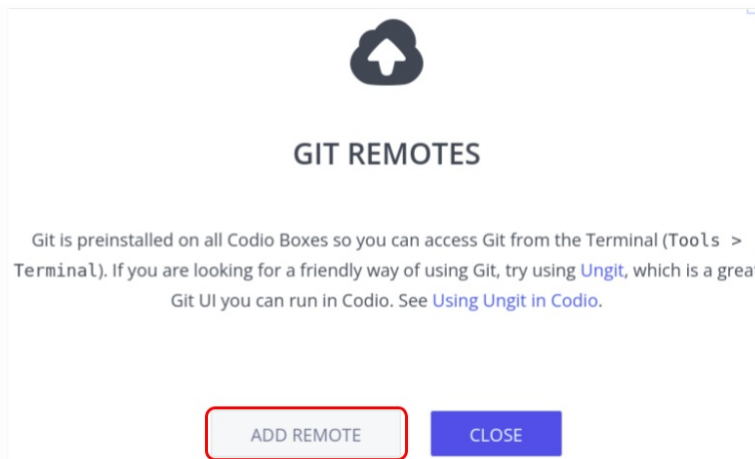
The image depicts the Tools menu bar where Git and Remotes have been selected. There is a red box around Remotes

We already have a remote repo, which we already created. However, a remote repo depends on a local repo, which we have not created. This is why Codio issues a message about initializing our directory. This will create a local repo on Codio. Click YES.



The image depicts a message about initializing the directory as a git repo. There is a red box around the Yes button.

The next screen you see is an empty list, because there are no remote repositories associated with our work in Codio. Click the button that says ADD REMOTE.

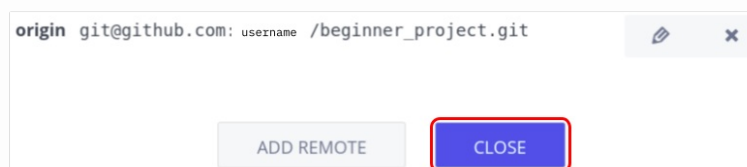


The image depicts an empty list of remotes for the directory. There is a red box around the Add Remote button.

You are then asked to give a name and URL for the remote. In the name field type origin. In the URL field, paste the SSH address for our repo on GitHub. Press SAVE.

The image depicts a field for the name of the remote and a field for the URL of the remote. There is a red box around the Save button.

You should now see a list of remotes associated with the directory. The remote name origin is listed in bold as well as the SSH URL for the remote. Press CLOSE.



The image depicts the newly added remote. The name “origin” is in bold and the SSH address is to the right. There is a red box around the Close button.

# Ignoring Files

## Ignoring Specific Files

The easiest way to make sure all of our files are pushed to GitHub. The problem with this is that all of the files will be added to the GitHub repo. More often than not, there are more files and directories that needed to be added to GitHub than those that should not be added. The best solution to this problem is the `.gitignore` file.

### ▼ Did you notice?

There is a leading `.` for the filename. This means the file is hidden. Be sure to use a period before the filename when creating the ignore file.

Directories and files that can be easily created by the user should be ignored. For example, ignore all Java class files. As you compile the `.java` files, the associated class files will automatically be created. If you are developing a project in JavaScript, you do not need to include the directory of the installed packages. You can run `npm install` to install them on your system.

## Ignoring Codio Files

In our case, we want to ignore any Codio-specific files and directories. Including them in your GitHub repo will not hurt anything. However, we want to get into the habit of only pushing those files and directories that are relevant to our project.

In the terminal, use the following command to create the `.gitignore` file for our project. Then open the file with the link.

```
touch .gitignore
```

We want to ignore the `.guides` directory which contains information for the instructional content for these pages. The `.settings` file provides settings only for Codio and not Django. You do not need this file for your project to run outside of Codio.

```
/.guides  
.settings
```

We are finally ready to start pushing our content to GitHub.

# Pushing to GitHub

## Adding and Committing

A previous image showed that the “push” command copies code from the local repo to the remote repo. However, it is a little more complicated than that. Before you can push, you need to get all of your code files into the local repository.

Start by making edits to your code file. Then copy them to the staging area with the add command. From there, you copy the code files to the local repository with the commit command. Committing changes requires a message. Best practices state you should give a clear explanation of what changes were made.



The image depicts three main areas: code file, staging area, and local repository. An arrow labeled “Add” connects the code file to the staging area. Another arrow labeled “Commit” connects the staging area and the local repo.

The final step is to copy your files to the remote repository with the push command. We will cover that in more detail below. But first, let’s add and commit our code files.

Use the following command to add all of the files and directories. Remember, git will only add files and directories if they do not appear in the .gitignore file.

```
git add .
```

Use the commit command to commit any changes to the code. The -m flag is for the message attached to the commit. The message should appear between double quotes. Developers often use "initial commit" as their message for their first commit. We will do the same.

```
git commit -m "initial commit"
```

## Pushing

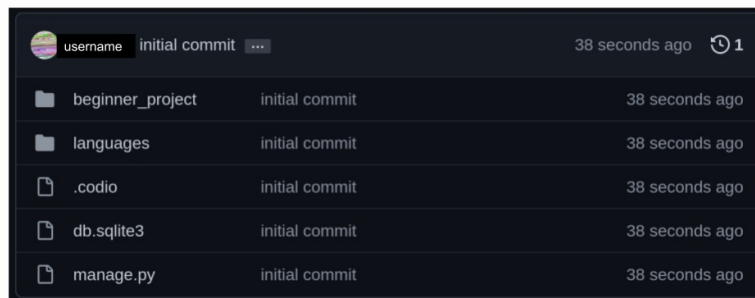
The final step is to push. You do not need to push your changes every time you commit them. Many developers will commit their changes throughout the day and only push once they are done for the day. **Important**, your code does not get backed up until you use the push command. If you want to be absolutely sure that your code has a backup, then push with more regularity. Use the push command to send your code to GitHub.

```
git push
```

Because this is the very first time you are pushing to this remote repo, you need a special command. I can never remember the exact command, so I use `git push`. Git will tell me what I should use instead. Copy the correct command from GitHub and paste it into the terminal. Press Enter or Return again.

```
git push --set-upstream origin master
```

Again, because this is the first push, git will ask if you want to store the SSH keys. Type yes and press Enter or Return. You will not have to answer this question again. Go back to the tab with the GitHub website and refresh the page. You should see your Django files. **Note**, your repo may look different from the one below. Regardless, your files should now reside in the GitHub repo.



The image depicts all of the Django folders and files residing in the GitHub repo.

## Future Changes to Your Code

As far as our project is concerned, we are done. Our Django project is now living on a GitHub server. However, if you need to make future changes to your Django project you will follow the same steps above, but with some minor differences. You are still going to add and commit (including a good

message) just as before. Using just `git push` will work as is. You do not need to use different commands or answer other questions. That's it, that's all you have to do.