**GIT – GITHUB NOTEBOOK**

# **03. Git Commands**

*# Remote & Local*

The GitHub repo we created is empty and sad right now, so let's work on adding some files. Think of something that you can share online, it could be some code (like a .html, .css, .js, .py, .cpp, .java file) from one of our courses, a personal project, or a homework assignment.

To get our code online, we have to work with two types of repos that connect with one another:

**☁️ Remote repository:** a GitHub repo stored somewhere on the internet (for the latest version).

**🏡 Local repository:** a Git repo that lives on your own computer (for drafts).

We have the empty remote repo on GitHub, so now we need a local repo. Let's go over some basic Git terminal commands to create a local repo and connect it to the remote repo.

***## Git Init***

The git init command initializes a new Git repo (local repo). It's the first command you run when you have a new project and want to start tracking changes. The history of your project starts here!

git init

When you run this command, Git will create a new directory named .git in your project folder. This directory contains all the information about your project's history and configuration.

The terminal will return something like this once done successfully:

Initialized empty Git repository in /Users/username/Desktop/python/.git

Think of this like turning a normal computer folder into a Git-tracked project.

***## Git Remote***

Local repo? Check. Remote repo? Check. Now, we can connect our local repo with our remote repo.The git remote command manages connections to remote repos. We can add a connection by:

git remote add origin <repository\_url>

**add:** Add a new remote connection.

**origin:** Give a nickname for the remote repo's URL (origin is a common one).

**<repository\_url>:** Placeholder for the remote repo's URL.

Our example would be:

git remote add origin https://github.com/codedex-io/first-repo.git

Here we're telling Git: "Yo, connect my local repo to the first-repo remote repo, and I’ll call the URL origin." Now, we have our local and remote repos connected! 🤝

¡Buena pregunta! Si quieres crear **otro repositorio local diferente** y conectarlo a **otro repositorio remoto distinto**, puedes hacerlo sin problema. Git te permite tener tantos repositorios como quieras, cada uno con su propio historial y conexión remota.

Aquí te muestro los pasos:

**✅ 1. Crear otro local repo**

bash

mkdir nuevo-proyecto

cd nuevo-proyecto

git init

Esto crea un nuevo repositorio local vacío en la carpeta nuevo-proyecto.

**✅ 2. Conectar a un nuevo remote repo**

Supongamos que tienes un nuevo repositorio en GitHub llamado otro-repo. Puedes conectarlo así:

bash

git remote add origin https://github.com/tu-usuario/otro-repo.git

**🔄 ¿Qué pasa ahora?**

Con ese comando, le estás diciendo a Git:

“Este nuevo repositorio local debe comunicarse con este repositorio remoto llamado origin.”

Después de eso puedes hacer:

* git push -u origin main para subir tus cambios.
* git pull origin main para traer cambios desde el remoto.

**🧠 Recuerda:**

* Cada carpeta con git init es un **repo local independiente**.
* Cada repo puede tener su **propio remote**, con su propia URL.
* El nombre origin es solo un **alias**. Puedes llamarlo como quieras, pero origin es el estándar.

**📂 ¿Dónde se ve este alias?**

Puedes ver los alias configurados con:

bash

git remote -v

Ejemplo de salida:

origin https://github.com/usuario/mi-repo.git (fetch)

origin https://github.com/usuario/mi-repo.git (push)

**🔁 Puedes cambiarlo o eliminarlo:**

*Cambiar de nombre:*

bash

git remote rename origin github

*Eliminarlo:*

bash

git remote remove origin

***## Git Branch***

Let's now rename the branch to main. This will be the branch we push code to, and will be default branch.

git branch -M main

The uppercase -M flag means "move" (or "rename"). We'll discuss what branches are later in the chapter. Let's put our new Git commands to the test!

**📌 ¿Qué es un branch en Git?**

Un **branch** (rama) en Git es una **línea de desarrollo** independiente. Piensa en ella como una copia del proyecto en la que puedes hacer cambios **sin afectar la rama principal** hasta que estés lista para unirlos.

* Por defecto, Git crea una rama llamada master o main cuando haces git init (dependiendo de la versión de Git).
* Hoy en día, **main es el nombre estándar recomendado** para la rama principal.

**🧩 ¿Qué hace este comando?**

bash

git branch -M main

**git branch:** comando para gestionar ramas.

**-M:** significa "**Move**" (mover o **renombrar**), con fuerza. Cambia el nombre de una rama existente.

**main:** es el **nuevo nombre** que le vas a dar a la rama actual.

Entonces este comando: Renombra la rama actual a main.

⚠️ Si ya existía una rama con ese nombre, -M la sobrescribe. Si no quieres sobrescribir nada, puedes usar -m (minúscula), que es más seguro.

**✅ ¿Por qué hacer esto?**

Es común usar git branch -M main justo después de hacer git init para:

* Establecer que la rama principal se llama main, en lugar de master.
* Asegurar compatibilidad con plataformas como GitHub, donde main ya es el nombre por defecto.

**🧪 Ejemplo práctico**

bash

mkdir mi-proyecto

cd mi-proyecto

git init

git branch -M main

Ahora la rama principal del proyecto se llama main.

Puedes verificarlo con:

bash

git branch

Te mostrará:

css

\* main

**Instructions**

First, use the terminal to navigate to a project folder on your computer with some code inside. Open up your selected folder in a code editor of your choice! (We like VS Code!)

Now, open a terminal or command prompt! In VS Code, you can open the terminal with ctrl + ` or go to Terminal > New Terminal.

Once you're here, we're ready to connect our code on our device to our GitHub repo.

Before we begin, make sure your terminal is in the right place to initialize the repository. You can check the full path of your current working directory by running the pwd (print working directory) command:

pwd

You should get a response with the path to your current directory, like /Users/taylor/Desktop/test!

Next, create a connection between the file in our machine and the GitHub repo by linking them with the git init, git remote add and git branch commands. ⋆˙⟡

To check we have successfully connected our local and remote repos and renamed the branch, run:

git branch

If you see main in the output, you're all set! 👌

\*\*\*

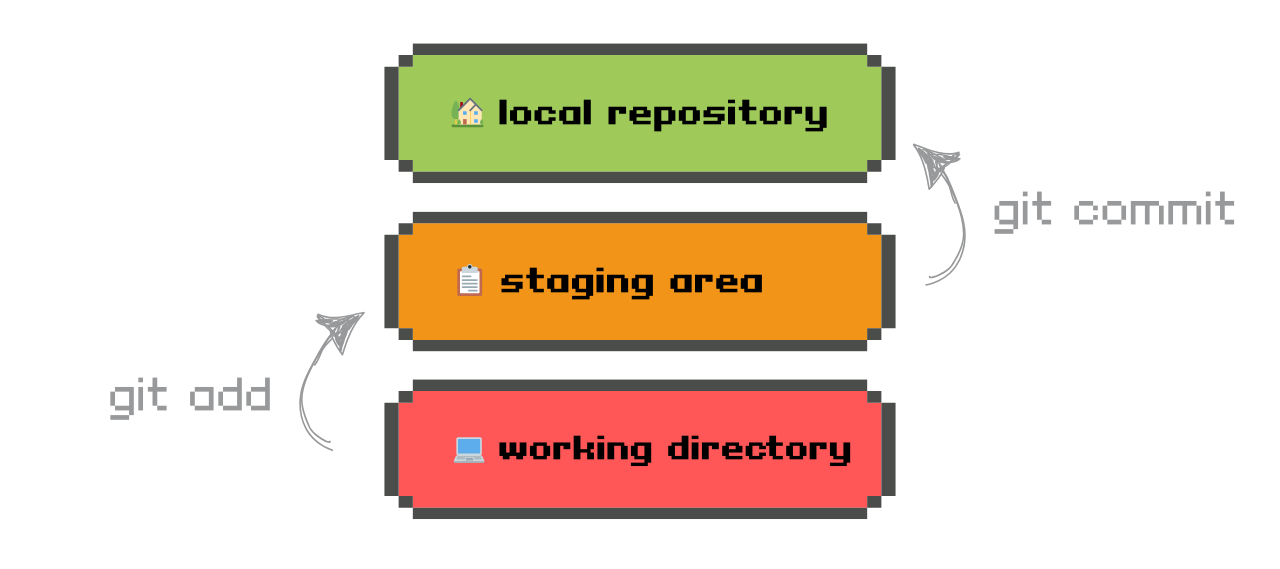
**04. Git Workflow**

[**#**](https://www.codedex.io/git-github/04-git-workflow#working-directory--staging)**Working Directory & Staging**

Now that we've connected our local & remote repos, let's turning the project folder into the local repo. The two main Git commands that we will use are git add and git commit.

But first, we have to make a distinction between the working directory and the staging area:

* 💻 **Working directory** is the project folder on your computer! When you make changes to the files, Git tracks them, and you can move selected changes to the staging area using the git add command.
* 📋 **Staging area** is where we prep changes, like "I'm almost ready! One more min!" It's a temporary area where you choose what files you want to "commit" to the local repo with git commit.



[**##**](https://www.codedex.io/git-github/04-git-workflow#git-add)**Git Add**

The git add command tells Git *which* changes you want to include in the next commit. Think of it like packing your suitcase before a trip – it's choosing what to bring.

Here are three variations...

* **Add one file**

To add a single file to the staging area:

git add example.txt

Here, the command will only add the **example.txt** file.

* **Add all files**

To add all the changed files to the staging area:

git add .

Here, any new file added or changed to the working directory will be included.

* **Add all files with an extension**

To only stage files with a specified extension:

git add *\*.html*

Here, the command will only add **.html** files.

[**##**](https://www.codedex.io/git-github/04-git-workflow#git-commit)**Git Commit**

Committing files are about saving a "snapshot" of the current code. 💾 Each commit captures a moment in time and includes a helpful message to explain what changed. This helps you track your progress and separate different actions in your code.

Now that we are ready to commit your staged files, use the following command:

git commit -m 'Your commit message here!'

The lowercase -m flag means "message".

Here's what some commit messages might look like for a typical project:

'Initial commit'

'Add pics to homepage'

'Fixed again fr this time!!'

**Note:** Commit messages get silly when you're working on your own for a while 😛, but short, clear, and descriptive messages are needed when working on a team! We want messages to help you (and your team) understand *what* changed and *why*.

If the commit is succesful, you should see a message appear in the terminal, something like:

git commit -m 'Updated index.html with a new line'

[main 09f4acd] Updated index.html with a new line

1 file changed, 1 insertion(+)

Pretty useful, right? Commit messages are almost like journal entries.

Let's continue and get our project to the local repo!

**Instructions**

Go back to your terminal or command prompt, and navigate to your project folder.

Use git add . to bring all the files in your project folder to the staging area.

Then, use git commit -m 'Some message here' to commit and write your first commit message! 💬

**05. Local Push**

[**#**](https://www.codedex.io/git-github/05-local-push#finally)**Finally...**

We are almost done! It's time to move our code from the local repo to the remote repo on GitHub. But before we push our code to GitHub, we need to make sure we have the correct files commited!

[**##**](https://www.codedex.io/git-github/05-local-push#git-status)**Git Status**

The git status command is used to check the status of your files. It is a handy command that will show you which files are staged, unstaged, and untracked.

* **Staged** files are ready to be committed.
* **Unstaged** files are not yet ready to be committed.
* **Untracked** files are new files that Git has not seen before.

Simply run the following command to check the status of your files:

git status

In short, you are saying, "Hey Git, what's the situation right now?"

An example response would be:

On branch main

No commits yet

Untracked files:

.DS*\_Store*

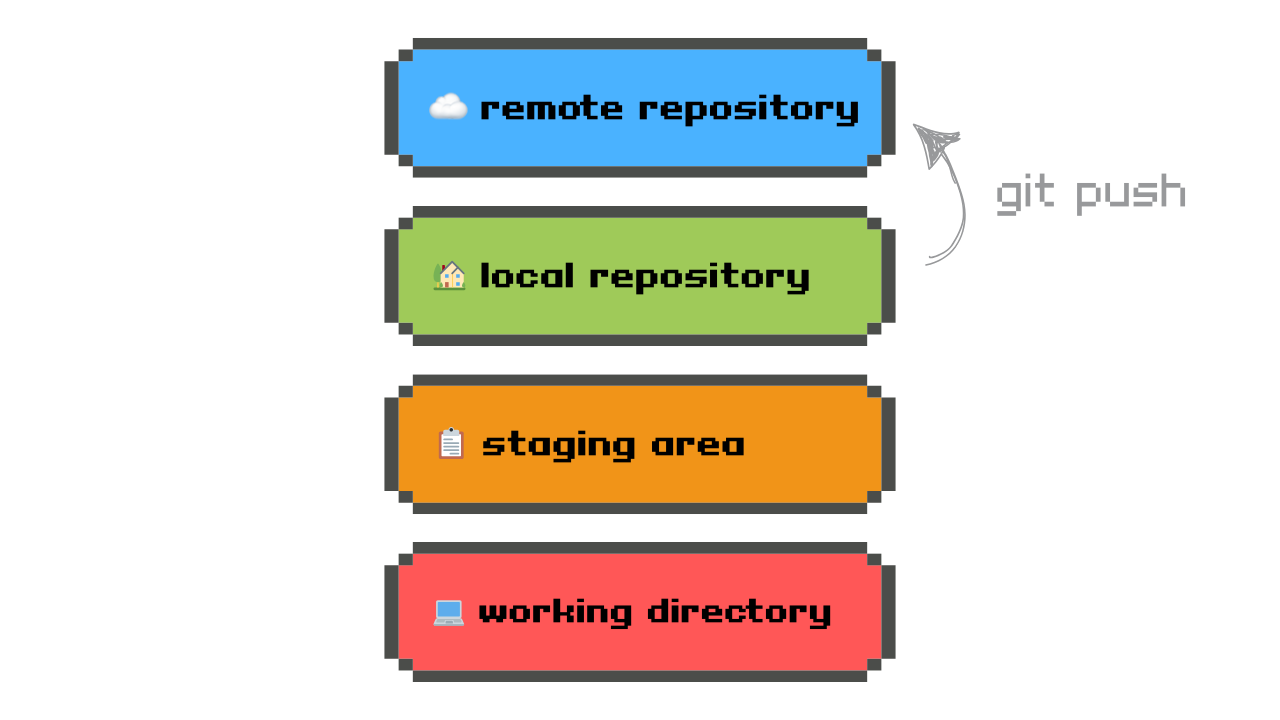
*test.py*

*output.gif*

Use git status anytime you're confused about the state of your repo.

[**##**](https://www.codedex.io/git-github/05-local-push#git-push)**Git Push**

Let's now use the command we'll use to finally "push" or publish our code! This is the last step.



The git push command is used to send your locally committed changes to your remote repository. You'll see all the changes you've made on GitHub.

***1️. First time***

The very first time you push to a branch, the command usually looks like this:

git push -u origin main

The -u flag stands for "upstream". This links your local branch to the remote branch so that future push commands will automatically apply to this branch without needing to specify it each time. ✨

***2️. Every other time***

Once that's set, any commits you push to this branch will just require:

git push

When that's done, you'll be able to refresh your GitHub repository URL, and see your changes online!

**Instructions**

Using the commands from this exercise, push your code to GitHub! 🥳 First, check the status using git status. Then, push to GitHub using git push -u origin main. Check to make sure this was completed successfully by refreshing the GitHub page! Your code is online!