# Git Install

You can download Git for free from the following website: [https://www.git-scm.com/](https://git-scm.com/)

**Using Git with Command Line**

To start using Git, we are first going to open up our Command shell.

For Windows, you can use Git bash, which comes included in Git for Windows. For Mac and Linux you can use the built-in terminal.

The first thing we need to do, is to check if Git is properly installed:

Exa:

git --version

If Git is installed, it should show something like git version X.Y

**Configure Git**

Now let Git know who you are. This is important for version control systems, as each Git commit uses this information:

Exa:

git config --global user.name "w3schools-test"

git config --global user.email "test@w3schools.com"

Change the user name and e-mail address to your own. You will probably also want to use this when registering to GitHub later on.

**Note:** Use global to set the username and e-mail for **every repository** on your computer.

If you want to set the username/e-mail for just the current repo, you can remove global

**Creating Git Folder**

Now, let's create a new folder for our project:

Exa:

mkdir myproject

cd myproject

mkdir **make**s a **new directory**.

cd **changes** the **current working directory**.

Now that we are in the correct directory. We can start by initializing Git!

**Note:** If you already have a folder/directory you would like to use for Git:

Navigate to it in command line, or open it in your file explorer, right-click and select "Git Bash here"

# Initialize Git

Once you have navigated to the correct folder, you can initialize Git on that folder:

git init

Initialized empty Git repository in /Users/user/myproject/.git/

You just created your first Git Repository!

**Note:** Git now knows that it should watch the folder you initiated it on.

Git creates a hidden folder to keep track of changes.

# Git Adding New Files

You just created your first local Git repo. But it is empty.

So let's add some files, or create a new file using your favourite text editor. Then save or move it to the folder you just created.

And save it to our directory as test.py.

Let's go back to the terminal and list the files in our current working directory:

Exa:

Ls

Test.py

ls will **list** the files in the directory. We can see that index.html is there.

Then we check the Git status and see if it is a part of our repo:

Exa:

git status

On branch master

No commits yet

Untracked files:

  (use "git add ..." to include in what will be committed)     index.html nothing added to commit but untracked files present (use "git add" to track)

Now Git is **aware** of the file, but has not **added** it to our repository!

Files in your Git repository folder can be in one of 2 states:

* Tracked - files that Git knows about and are added to the repository
* Untracked - files that are in your working directory, but not added to the repository

 When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

# Git Staging Environment

One of the core functions of Git is the concepts of the Staging Environment, and the Commit.

As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

**Staged** files are files that are ready to be **committed** to the repository you are working on. You will learn more about commit shortly.

For now, we are done working with test.py. So we can add it to the Staging Environment:

Syntax:

git add test.py

after that use command below:

git status

u should see:

On branch master

No commits yet

Changes to be committed:

  (use "git rm --cached ..." to unstage)     new file: index.html

# Add multiple files

add all files in the current directory to the Staging Environment:

syntax:

git add –all

Using --all instead of individual filenames will stage all changes (new, modified, and deleted) files.

**Note:** The shorthand command for git add --all is git add -A

# Git Commit

Since we have finished our work, we are ready move from stage to commit for our repo.

Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should **always** include a **message**.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

Exa:

git commit -m “first release of hello world”

The commit command performs a commit, and the -m "message" adds a message.

The Staging Environment has been committed to our repo, with the message:  
"First release of Hello World!"