GIT Y GIT Hub

GIT es el sistema y GitHub es la plataforma donde se comparten los repositorios.

GIT es para trabajar con los repositorios de manera LOCAL

Git Hub es donde los compartimos.

What does Git do?

* Manage projects with **Repositories**
* **Clone** a project to work on a local copy
* Control and track changes with **Staging** and **Committing**
* **Branch** and **Merge** to allow for work on different parts and versions of a project
* **Pull** the latest version of the project to a local copy
* **Push** local updates to the main project

Working with Git

* Initialize Git on a folder, making it a **Repository**
* Git now creates a hidden folder to keep track of changes in that folder
* When a file is changed, added or deleted, it is considered **modified**
* You select the modified files you want to **Stage**
* The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
* Git allows you to see the full history of every commit.
* You can revert back to any previous commit.
* Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

What is GitHub?

* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

**MI CONFIGURACION:**

* git config --global user.name "Eduardo Alvarez"
* git config --global user.email [opticoedu@gmail.com](mailto:opticoedu@gmail.com)

**CREATING A GIT FOLDER:**

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!

## 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.

EJEMPLO: YO ACA VOY AGREGANDO LOS FILES TIPO HTML, CSS, JS,FOTOS, ETC, O SEA VOY LLENANDO ESA CARPETA CON LOS FILES.

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

ls

index.html

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:

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 index.html. So we can add it to the Staging Environment:

git add index.html

## 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

git commit -m "First release of Hello World!"

[master (root-commit) 221ec6e] First release of Hello World!

3 files changed, 26 insertions(+)

create mode 100644 README.md

create mode 100644 bluestyle.css

create mode 100644 index.html

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!"

## Git Commit without Stage

Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The -a option will automatically stage every changed, already tracked file.

Let's add a small update to index.html:

And check the status of our repository. But this time, we will use the --short option to see the changes in a more compact way:

git status --short

M index.html

**Note:** Short status flags are:

* ?? - Untracked files
* A - Files added to stage
* M - Modified files
* D - Deleted files
* git commit -a -m "Updated index.html with a new line"
* [master 09f4acd] Updated index.html with a new line
* 1 file changed, 1 insertion(+)
* **Warning:** Skipping the Staging Environment is not generally recommended.
* Skipping the stage step can sometimes make you include unwanted changes.

## Git Commit Log

To view the history of commits for a repository, you can use the log command:

git log

commit 09f4acd3f8836b7f6fc44ad9e012f82faf861803 (HEAD -> master)

Author: w3schools-test Date: Fri Mar 26 09:35:54 2021 +0100 Updated index.html with a new line commit 221ec6e10aeedbfd02b85264087cd9adc18e4b26 Author: w3schools-test Date: Fri Mar 26 09:13:07 2021 +0100 First release of Hello World!

## Git Help

If you are having trouble remembering commands or options for commands, you can use Git help.

There are a couple of different ways you can use the help command in command line:

* git command -help -  See all the available options for the specific command
* git help --all -  See all possible commands

## Working with Git Branches

In Git, a branch is a new/separate version of the main repository.

Branches allow you to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects without them interfering with each other.

Branching in Git is very lightweight and fast!

## New Git Branch

Let add some new features to our index.html page.

We are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So we create a new branch:

git branch hello-world-images

## Emergency Branch

Now imagine that we are not yet done with hello-world-images, but we need to fix an error on master.

I don't want to mess with master directly, and I do not want to mess with hello-world-images, since it is not done yet.

So we create a new branch to deal with the emergency:

git checkout -b emergency-fix

Switched to a new branch 'emergency-fix'

LUEGO:

git add index.html

git commit -m "updated index.html with emergency fix"

[emergency-fix dfa79db] updated index.html with emergency fix

1 file changed, 1 insertion(+), 1 deletion(-)

Now we have a fix ready for master, and we need to merge the two branches.

## Merge Branches

We have the emergency fix ready, and so let's merge the master and emergency-fix branches.

First, we need to change to the master branch: