APUNTES GIT

Cuando se queda atrapado en la terminal tienes que presionar q

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

Git is the industry-standard version control system for web developers

- Use Git commands to help keep track of changes made to a project:
  - `git init` creates a new Git repository
  - `git status` inspects the contents of the working directory and staging area
  - `git add` adds files from the working directory to the staging area
  - `git diff` shows the difference between the working directory and the staging area
  - `git commit` permanently stores file changes from the staging area in the repository
  - `git log` shows a list of all previous commits

git add .

Para agregar todos los archivos (sin tener que hacer un git add para cada uno)

PARA VOLVER ATRAS ANTES DE LOS CAMBIOS

```
git checkout HEAD filename
```

`git checkout -- filename`

will restore the file in your working directory to look exactly as it did when you last made a commit.

`git reset HEAD filename`

This command resets the file in the staging area to be the same as the `HEAD` commit. It does not discard file changes from the working directory, it just removes them from the staging area.

Before reset:

- `HEAD` is at the most recent commit

After resetting:

- `HEAD` goes to a previously made commit of your choice
- The gray commits are no longer part of your project
- You have in essence rewound the project's history

`git reset commit_SHA`: Resets to a previous commit in your commit history.

git branch nombre_de_la_branch_nueva

git branch

`git checkout branch_name`

para cambiar de rama

`git merge branch_name`

Your goal is to update `master` with changes you made to `fencing`.

Notice the output: The merge is a "fast forward" because Git recognizes that `fencing` contains the most recent commit.

Git *fast forwards* `master` to be up to date with `fencing`.

WHEN A CONFLICT SHOWS UP

We must fix the merge conflict.

In the code editor, look at **resume.txt**. Git uses markings to indicate the `HEAD` (master) version of the file and the `fencing`version of the file, like this:

```
<<<<<<< HEAD
master version of line
=======
fencing version of line
>>>>>>> fencing
```

Git asks us which version of the file to keep: the version on `master` or the version on `fencing`. You decide you want the `fencing` version.

From the code editor:

Delete the content of the line as it appears in the `master`branch

Delete **all of Git's special markings** including the words `HEAD`and `fencing`. If any of Git's markings remain, for example,`>>>>>>>` and `=======`, the conflict remains.

`git branch -d branch_name`

si no está fully merged -D (tienes q poner D mayus)

The following commands are useful in the Git branch workflow.

- `git branch`: Lists all a Git project's branches.
- `git branch branch_name`: Creates a new branch.
- `git checkout branch_name`: Used to switch from one branch to another.
- `git merge branch_name`: Used to join file changes from one branch to another.
- `git branch -d branch_name`: Deletes the branch specified.

You can accomplish all of this by using remotes. A remote is a shared Git repository that allows multiple collaborators to work on the same Git project

from different locations. Collaborators work on the project independently, and merge changes together when they are ready to do so.

```
git clone remote_location clone_name
```

In this command:

- `remote_location` tells Git where to go to find the remote. This could be a web address, or a filepath, such as:

```
/Users/teachers/Documents/some-remote
```

- `clone_name` is the name you give to the directory in which Git will clone the repository.

The remote is listed twice: once for `(fetch)` and once for `(push)`. We'll learn about these later in the lesson.

```
git fetch
```

This command will not merge changes from the remote into your local repository. It brings those changes onto what's called a remote branch. Learn more about how this works below.

The workflow for Git collaborations typically follows this order:

1. Fetch and merge changes from the remote
2. Create a branch to work on a new project feature
3. Develop the feature on your branch and commit your work
4. Fetch and merge from the remote again (in case new commits were made while you were working)
5. Push your branch up to the remote for review

The command:

```
git push origin your_branch_name
```

will push your branch up to the remote, `origin`

A remote is a Git repository that lives outside your Git project folder. Remotes can live on the web, on a shared network or even in a separate folder on your local computer.

- The Git Collaborative Workflow are steps that enable smooth project development when multiple collaborators are working on the same Git project.

We also learned the following commands

- `git clone`: Creates a local copy of a remote.
- `git remote -v`: Lists a Git project's remotes.
- `git fetch`: Fetches work from the remote into the local copy.
- `git merge origin/master`: Merges `origin/master` into your local branch.
- `git push origin <branch_name>`: Pushes a local branch to the `origin`remote.