Git y Github son diferentes, primero hablaremos sobre el concepto de Git y después nos pondremos con el concepto de Github.
Git página oficial

- Sistema de control de versiones distribuido.
- Llevar todos los cambios que se realizan en un proyecto, documentarlos.
- Creado por Linus Torvalds

Instalación de Git

Si tenemos un sistema operativo como Mac o Linux, tenemos por defecto instalado Git, no es necesario realizar una instalación.

```
git --version o git -v
```

Para comprobar la versión que tenemos instalada en nuestro ordenador.

> **Note**
>
> Nota: Cuando se realice la instalación en Windows se instalara una terminal con bash para poder utilizar Git sin problemas, llamada Git-bash.

Realizada la instalación probaremos algunos comando de Git.

```
git --help o git -h
```

Todos los comandos son exactamente los mismos para cualquier sistema operativo, mientras se trabaje en una terminal que sea capaz de entender Git.

Comandos de la terminal

Comandos principales de una terminal:

```
ls
```

Listado de todos los directorios que tenemos.
Cómo nos desplazamos por sistemas de ficheros?

```
cd Nombre
```

Dónde nos encontramos?

```
pwd
```

Cómo podemos crear una carpeta usando la terminal?

```
mkdir "Nombre"
```

Una vez creada nuestra carpeta donde almacenaremos nuestro proyecto de código, usaremos nuestro editor de código preferido. (En este caso se usa VS Code)

```
code .
```

Configuración de Git

Git nos ayuda a que varias personas unan un código, si encontramos un error podemos identificar que persona ha actualizado cada parte del código.
Entonces tenemos que asociarnos a que persona esta trabajando.

Mínimo: Nombre de usuario y un email.

Para comenzar la configuración inicial, lo haremos a nivel global, esto significa que configuraremos Git a nivel del equipo/ordenador/pc.

```
git config --global
```

Configuremos entonces nuestro usuario para Git:

```
git config --global user.user "Username"

git config --global user.email "email"
```

Podemos revisar nuestro archivo gitconfig para revisar la configuración de nuestro user con los siguientes comandos

```
~/.gitconfig #Obtener la dirección donde se encuentra.

cat dirección/gitconfig #Abrir directamente en terminal
```

Git INIT

Para empezar a trabajar con Git, una vez hecha la correcta configuración es crear un fichero de código para empezar a entender los conceptos de manejo de control de versiones, puede ser el fichero que sea.
Podemos crearlo usado la consola

```
touch nombre.py #Fichero en python
```

Cómo indicamos que queremos trabajar en nuestro directorio con Git?

```
git init
```

Al correr este comando se creara un fichero llamado .git

> ⚠ **Warning**
>
> Es un fichero oculto, te asustes si no le ves.

Nos indica que de alguna forma hemos creado un repositorio de GIt. Este directorio trabaja con un control de versiones.

Ramas en Git

Una vez que se ejecuta el comando git init, por defecto se crea una rama llamada master/main, que es donde tendremos nuestro proyecto. Podemos revisar en que rama estamos usando

```
git status
```

Estamos entonces en la rama principal de nuestro proyecto

Git ADD y GIt COMMIT

Como podemos guardar nuestro código, para eso tenemos que añadir los ficheros que deseamos ir haciendo "fotografías".
Vamos a crear la primera "fotografía" de nuestro proyecto usando:

```
git add "ficheros"
```

Ahora que tenemos ese fichero preparado para hacerle una fotografía, cómo la hacemos? ---> COMMITS.
Vamos a realizar la fotografía con

```
git commit -m "comentario de que se realizó."
```

Si realizamos un commit sin comentarios, nos mandará directamente a un editor de texto para que lo realicemos, ya que no se puede hacer un commit sin comentarios.

Git LOG

Podemos revisar quien y cuando se han realizado commits con el siguiente comando:

```
git log
```

Mostrará el usuario, fecha y comentario del commit, así como la rama donde se realizó.

> **Note**
>
> Tiene más atributos que se pueden revisar en la documentación oficial para que el log sea más resumido.

Git CHECKOUT Y RESET

Imaginemos que no nos gustaron los cambios que realizamos en el código y queremos volver a una versión anterior, usaremos el siguiente comando

```
git checkout fichero
```

El cual nos ayuda a situarnos en un punto concreto ---> volver a una versión anterior.

Para revisar que archivos fueron modificados podemos usar

```
git reset
```

Git Alias

Podemos configurar un alias para no tener que escribir todo el tiempo comando largos con el siguiente comando:

```
git config --global alias.nombre "comando"
```

Esto guardara en nuestra configuración global ese alias y al momento de ejecutar un

```
git alias
```

Se ejecutará el comando que pusimos.

GITIGNORE

Ignorar ficheros, ficheros que no queremos nunca guardar.
Para ello crearemos un archivo nuevo, lo podemos crear desde terminal o no llamado de la siguiente manera 100% igual.

```
touch .gitignore
```

Los ficheros que añadamos dentro, son lo que no queremos tener en cuenta.

Git Diff

Con el siguiente comando podremos ver los cambios que se han realizado en los ficheros antes de realizar un commit .

```
git diff
```

TAG

Nos ayuda a crear identificadores para los commits

```
git tag nombre_del_tag
```

RAMAS

Podemos crear varias ramas para trabajar en cosas distintas, sirve más para equipos grandes o cuando más de una persona trabaja en el código.
Podemos crear una rama nueva con el comando

```
git branch nombre_rama
```

Para poder movernos entre ramas tendremos que usar el comando

```
git switch rama
```

También podríamos usar el comando checkout, pero es mejor práctica usar el comando switch, ya que al hacer un checkout descargamos los cambios, sirve para cuando estás trabajando en algo remoto.

Cómo podemos mantener coherencia entre ramas

```
git merge rama
```