¿Qué es GIT?

Es un software de control de versiones, su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Existe la posibilidad de trabajar de forma remota y una opción es **GitHub**.

¿Qué es GitHub?

Es una plataforma de desarrollo colaborativo para alojar proyectos (en la “nube”) utilizando el sistema de control de versiones Git.

Además cuenta con una herramienta muy útil que es GitHub Pages donde podemos publicar nuestros proyectos estáticos (HTML, CSS y JS) de forma gratuita.

GIT | Definición

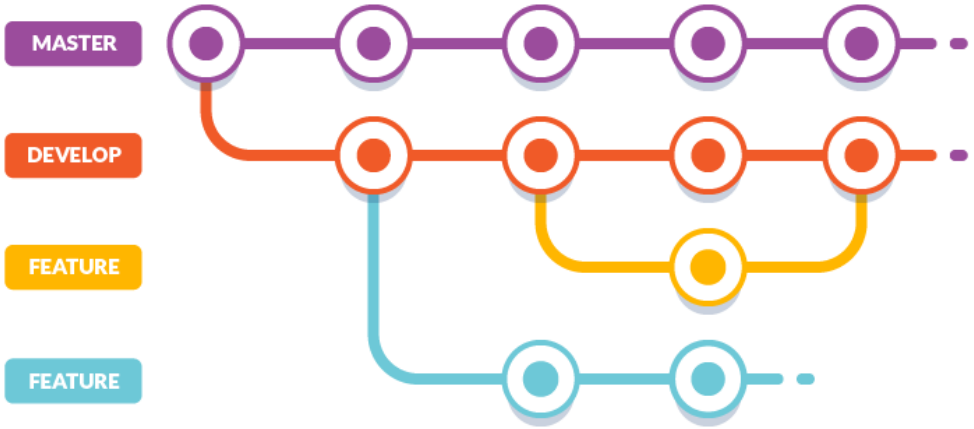
Un sistema que ayuda a organizar el código, el historial y su evolución, funciona como una máquina del tiempo que permite navegar a diferentes versiones del proyecto y si queremos agregar una funcionalidad nueva nos permite crear una rama (branch) para dejar intacta la versión estable y crear un ambiente de trabajo en el cual podemos trabajar en una nueva funcionalidad sin afectar la versión original.

Mediante el control de versiones distribuido permite:

- Manejar distintas versiones del proyecto.
- Guardar el historial o guardar todas las versiones de los archivos del proyecto.
- Trabajar simultáneamente sobre un mismo proyecto.

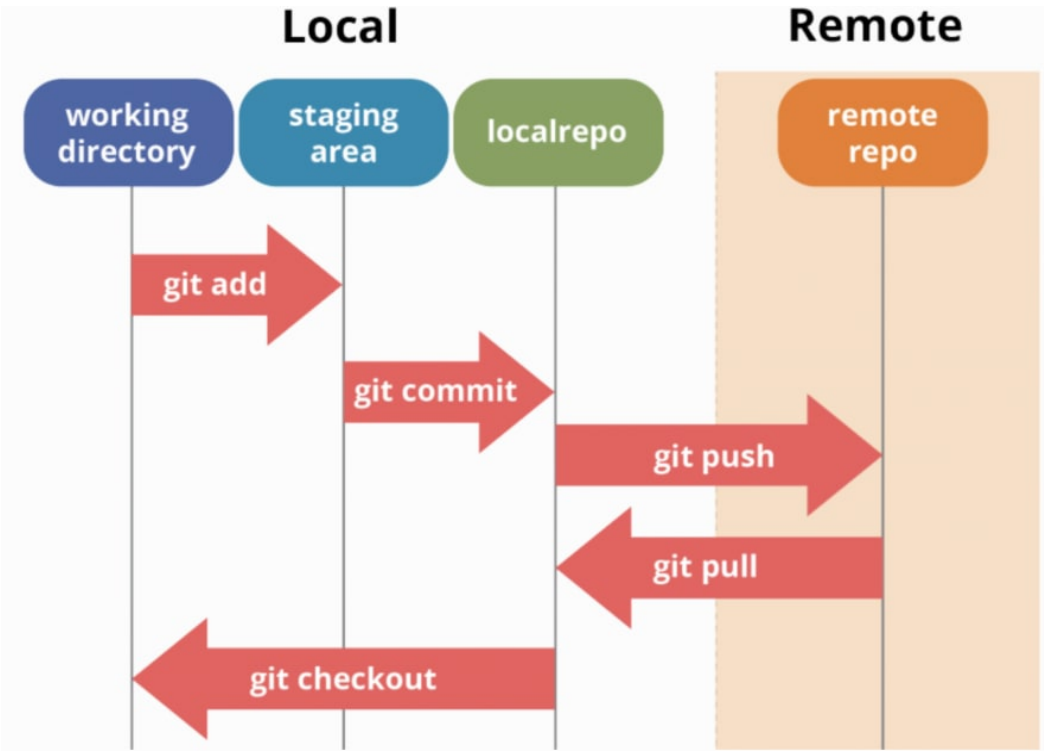
GIT | ¿Cómo funciona?

Git almacena instantáneas de un mini sistema de archivos, cada vez que confirmamos un cambio lo que Git hace es tomar una "foto" del aspecto del proyecto en ese momento y crea una referencia a esa instantánea, si un archivo no cambió Git no almacena el nuevo archivo sino que crea un enlace a la imagen anterior idéntica que ya tiene almacenada.



GIT | Flujo de trabajo

Tenemos nuestro directorio local (una carpeta en nuestra PC) con muchos archivos, Git nos irá registrando los cambios de archivos o códigos cuando nosotros le indiquemos, así podremos viajar en el tiempo retrocediendo cambios o restaurando versiones de código, ya sea en Local o de forma Remota (servidor externo).



GIT | Terminología

- **Repositorio:** es la carpeta principal donde se encuentran almacenados los archivos que componen el proyecto. El directorio contiene metadatos gestionados por Git, de manera que el proyecto es configurado como un repositorio local.
- **Commit:** un commit es el estado de un proyecto en un determinado momento de la historia del mismo, imaginemos esto como punto por punto cada uno de los cambios que van pasando. Depende de nosotros determinar cuántos y cuales archivos incluirá cada commit.
- **Rama (branch):** una rama es una línea alterna del tiempo, en la historia de nuestro repositorio. Funciona para crear features, arreglar bugs, experimentar, sin afectar la versión estable o principal del proyecto. La rama principal por defecto es **master**.
- **Pull Request:** en proyectos con un equipo de trabajo, cada persona puede trabajar en una rama distinta pero llegado el momento puede pasar que dicha rama se tenga que unir a la rama principal, para eso se crea un **pull request** donde comunicas el código que incluye tu cambio y usualmente revisa tu código, se agregan comentarios y por último lo

aprueban para darle **merge**. En el contexto de GIT, merge significa unir dos trabajos, en este caso tu **branch** con **master**.

GIT | Instalación

1) Descargar desde: https://git-scm.com/downloads.
2) Una vez descargado, se emplea la interfaz de línea de comando del sistema operativo para interactuar con GIT:
- En Windows: abrir la aplicación Git Bash que se instaló junto con GIT.
- En Mac: abrir la terminal mediante el finder.
- En Linux: abrir la consola bash.

3) Para verificar si está instalado, podemos ejecutar el comando: git --version.
- Si obtenemos respuesta nos indicará la versión de Git que tenemos instalada.
- En caso de que no poder, ver las instrucciones acá: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git

Meld | Instalación

1) Descargar desde meldmerge.org
2) Instalar el archivo descargado en la opción Windows, leer opciones para Linux, Mac OS.

GIT y GitHub | Material multimedia

- **Videos del Profesor Alejandro Zapata (Coordinador y Docente de Codo a Codo):** https://www.youtube.com/watch?v=ptXiQwE535s&list=PLoCpUTIZIYORkDzYwdunkVf-KIqGjyoot
- **GIT y GitHub (tutorial en español). Inicio Rápido para Principiantes:** https://www.youtube.com/watch?v=hWgIK8nWh60&list=PLPl81lqbj-4I8i-x2b5_MG58tZfgKmJls
- **¿Cómo trabajar con Git desde Visual Studio Code?:** https://youtu.be/AYbgqmyg7dk

Guía de comandos

1. Configuración inicial

- **Comandos útiles de terminal (crear y acceder a carpeta)**

```
mkdir carpeta
cd carpeta
```

- **Configuración de git y merge tool después de la descarga e instalación**

```
git config --global user.name "Example"
git config --global user.email "example@example.com"
git config --global merge.tool meld
git config --global mergetool.meld.path "C:\Program Files
(x86)\Meld\Meld.exe"
git config --global mergetool.keepBackup false
```

- **Versión actualmente instalada de git**

```
git version
```

- **Inicializar repositorio (en la carpeta actualmente seleccionada en el terminal)**

```
git init
```

- **Estado del repositorio - Estado del repositorio (versión simplificada)**

```
git status
git status --short
```

2. Staging Area

- **Añadir archivos al staging area (si se usa el punto se añaden todos los archivos modificados)**

```
git add index.html
git add .
```

- **Unstagear archivo**

```
git restore --staged index.html
```

- **Restaurar cambios a la última versión commiteada**

```
git restore index.html
```

3. Local Repo

- **Realizar commit**

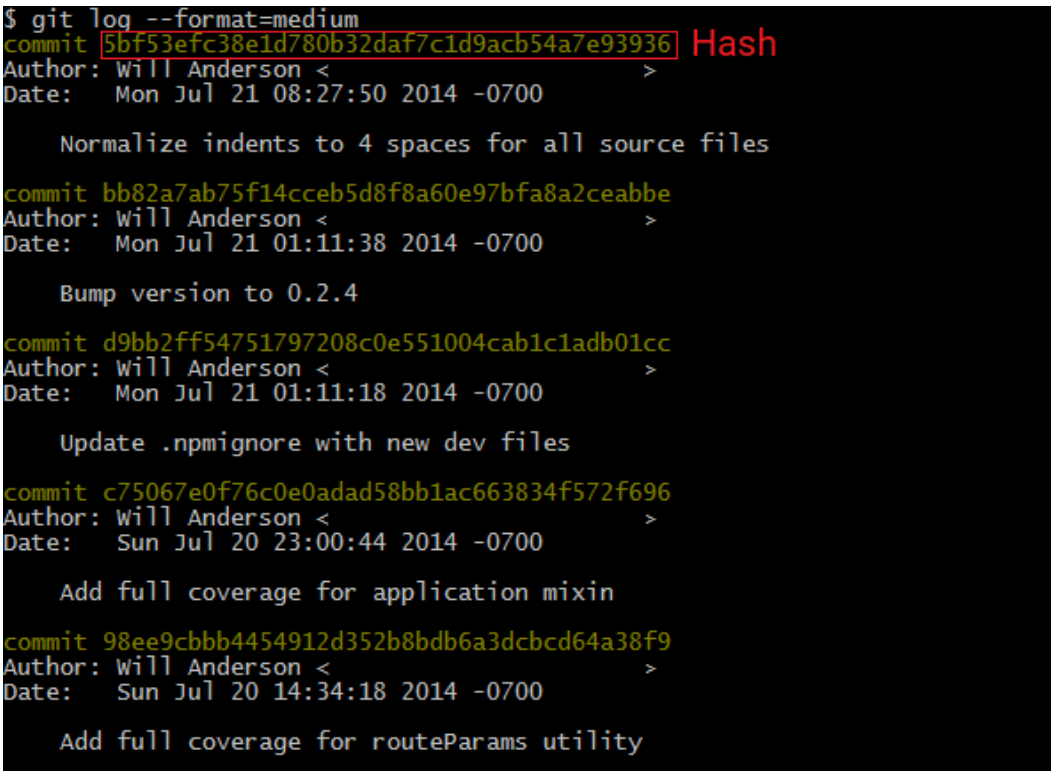
```
git commit -m "Mensaje"
```

- **Log de commits**

```
git log
```

- **Ir a un commit específico (copiar hash de commit del git log)**

```
git checkout 5bf53efc38e1d780b32daf7c1d9acb54a7e93936
```



4. Branches

- **Crear branch nueva**

```
git branch example
```

- **Ver branches**

```
git branch
```

- **Switchear entre branches**

```
git checkout example
git checkout master
```

- **Borrar branches**

```
git branch -d example
```

- **Mergear branches**

```
git merge example (desde master)
```

- **En caso de haber conflicto - abrir mergetool (Meld)**

```
git mergetool  (hacer un commit de los cambios después de esta línea)
```

5. GitHub

Iniciar sesión en GitHub y crear repositorio, una vez creado copiar la URL que el sitio brinda
Asegurarse de que sea con el mismo mail configurado anteriormente.

- **Configurar repositorio remoto (desde VS Code)**

Es posible que pida loguearse con las credenciales de **GitHub** al ejecutar los siguientes comandos

```
git remote add origin https://github.com/example/gittest.git
git branch -M main
git push -u origin main
```

- **Traer cambios realizados en GitHub al workspace local**

```
git pull
```

- **Clonar repositorio remoto localmente**

El repositorio aparecerá en la carpeta que tenemos seleccionada en el terminal

```
git clone https://github.com/w3schools-test/w3schools-test.github.io.git
```

6. .gitignore

Fuente: https://www.freecodecamp.org

El archivo .gitignore, es un archivo de texto que le dice a Git qué archivos o carpetas ignorar en un proyecto.

Un archivo local .gitignore generalmente se coloca en el directorio raíz de un proyecto. También puedes crear un archivo global .gitignore, y cualquier entrada en ese archivo se ignorará en todos tus repositorios de Git.

Para crear un archivo .gitignore local, crea un archivo de texto y asígnale el nombre ".gitignore" (recuerda incluir el . al principio). Luego, edita este archivo según sea necesario. Cada nueva línea debe incluir un archivo o carpeta adicional que quieras que Git lo ignore.

Este es un ejemplo de cómo puede lucir el archivo .gitignore :

```
# Ignora archivos del sistema Mac
.DS_store

# Ignora la carpeta node_modules
node_modules

# Ignora todos los archivos de texto
*.txt

# Ignora los archivos relacionados a API keys
.env

# Ignora archivos de configuración SASS
.sass-cache
```