

Hola devs! Les debo este tutorial para hacer la conexión desde la terminal hacía github y vamos a repasar algunos puntos básicos. Antes de comenzar asegúrate de tener instalado GITBASH y VSCODE o tu editor de texto preferido.

Primero, ¿qué es GIT?

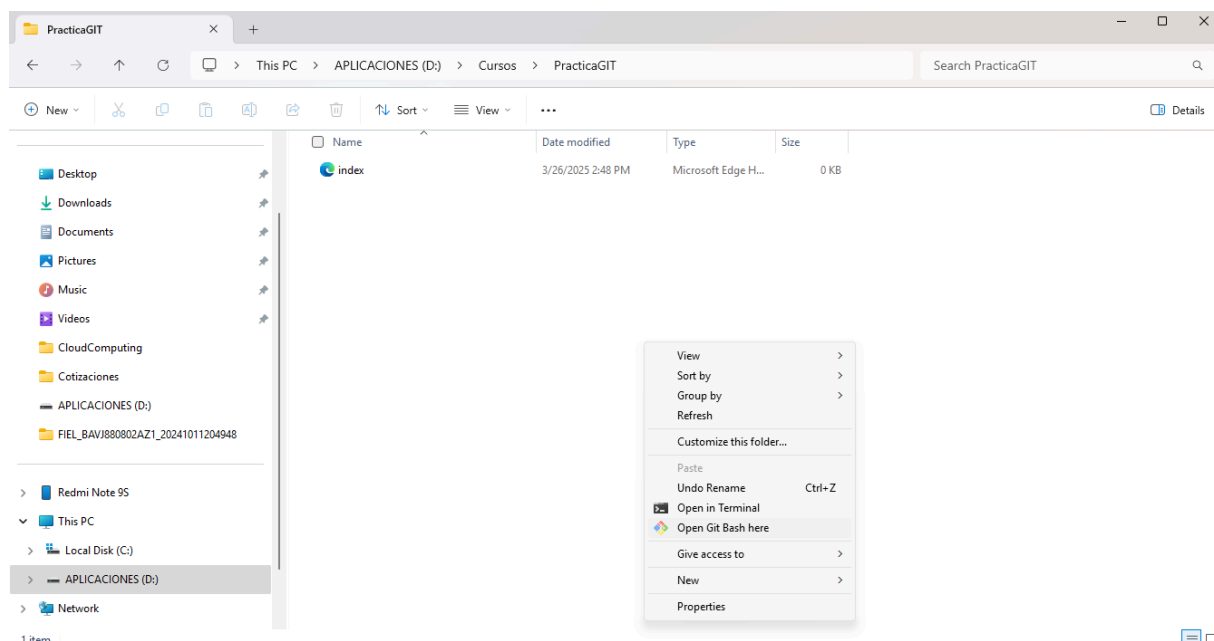
Git es un **sistema de control de versiones distribuido** que permite gestionar el historial de cambios en archivos y coordinar el trabajo entre múltiples desarrolladores. Es ampliamente utilizado en el desarrollo de software para rastrear cambios en el código fuente, colaborar en proyectos y facilitar la integración de nuevas características sin perder versiones anteriores.

Características principales:

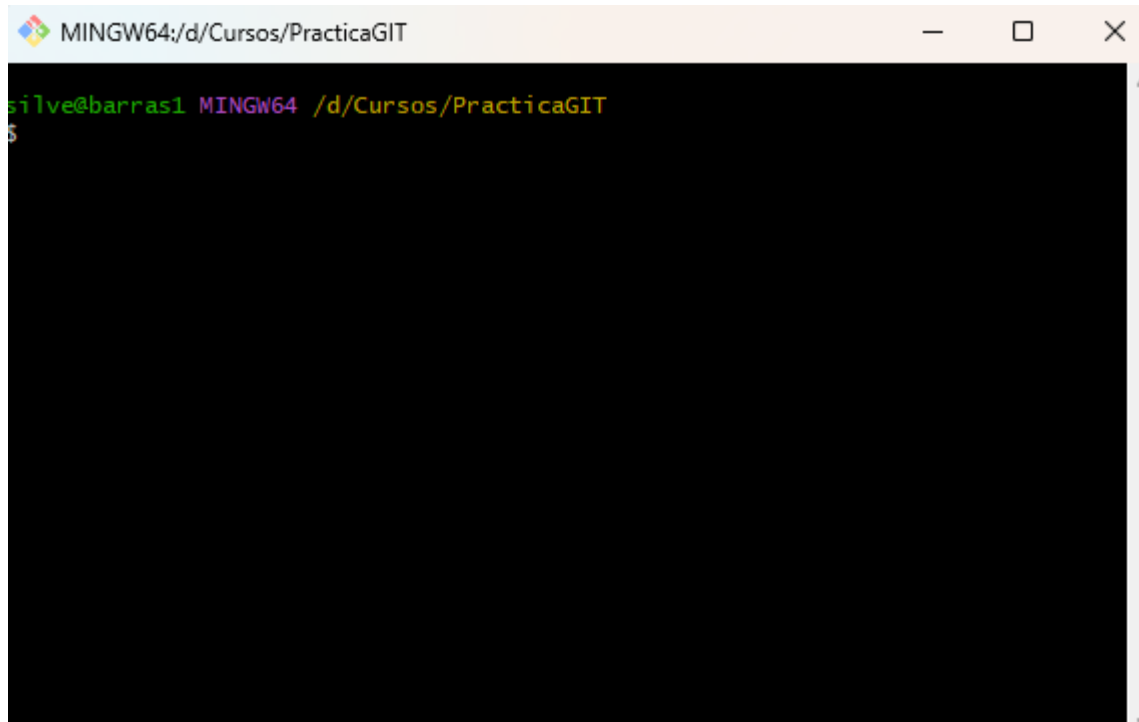
- **Distribuido:** Cada copia del repositorio contiene todo el historial, lo que permite trabajar sin conexión y realizar copias de seguridad fácilmente.
- **Control de versiones:** Permite registrar cambios en los archivos a lo largo del tiempo y volver a versiones anteriores si es necesario.
- **Ramas (Branches):** Facilita la creación de nuevas funcionalidades sin afectar el código principal y permite la integración segura mediante "merges".
- **Colaboración:** Es la base de plataformas como **GitHub, GitLab y Bitbucket**, que facilitan el trabajo en equipo.

Como ya lo hemos discutido en sesión, veremos cómo trabajar con la interfaz CLI (terminal) de gitbash, esto para que aprendas los comandos básicos.

Lo primero es abrir nuestra terminal, es importante abrirla en la misma carpeta que el proyecto en el que vas a trabajar:



Un poco sobre la interfaz de la terminal, te muestra el usuario, dividido por el nombre del equipo, después la estructura y al final la ruta donde estamos colocados.

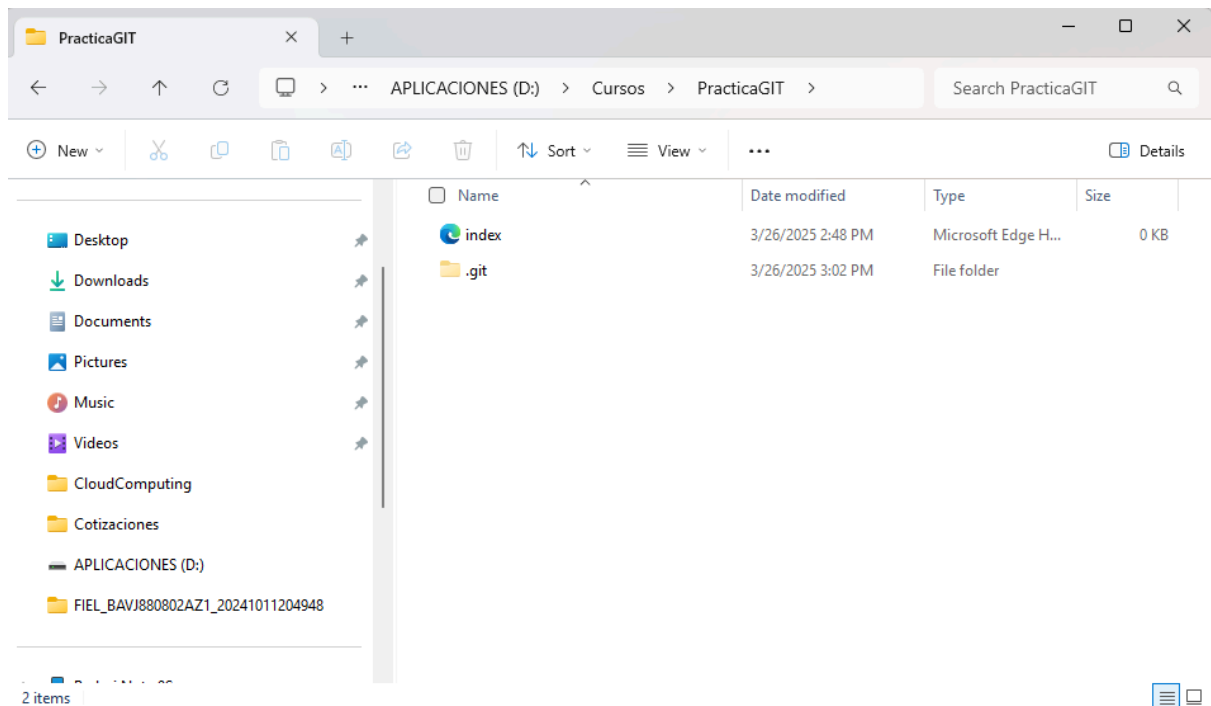


```
MINGW64:/d/Cursos/PracticaGIT  
silve@barras1 MINGW64 /d/Cursos/PracticaGIT  
$
```

para inicializar GIT en nuestro repositorio corremos el siguiente comando:

```
git init
```

Esto va a iniciar GIT en el proyecto creando una carpeta llamada .git, está carpeta guarda todo el historial, si la borras perderás toda la información.



¿Cómo funciona GIT?

Git funciona como un **sistema de control de versiones distribuido**, lo que significa que cada usuario tiene una copia completa del historial del repositorio en su máquina. Se basa en la creación de "snapshots" (instantáneas) del estado de los archivos en un momento dado, permitiendo registrar cambios, trabajar en paralelo y fusionar modificaciones sin perder el historial.

Vamos a realizar un cambio en el proyecto, puede ser cualquiera. Crearemos una estructura básica de html.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

Una vez salvado el documento, corremos el siguiente comando:

```
git status
```

La salida de la consola nos despliega lo siguiente:

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files present (use "git add" to track)
```

GIT ha registrado un cambio reciente en el archivo index.html, pero aún no lo agregamos al área de cambios (Staging area), si no hacemos esto no podremos regresar a ese punto más adelante de querer hacerlo. Para realizarlo corremos:

```
git add .
```

Este comando agrega TODOS los archivos al área de cambios, ahora git ya está registrando los cambios, si volvemos a correr un git status, podremos comprobarlo:

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html
```

index.html ya está en el área de cambios, conforme yo vaya agregando más cambios al código puedo repetir el mismo comando e irlos agregando.

Una vez termine de agregar los cambios que necesito, es hora de tomar la fotografía (snapshot) para que esos cambios se queden guardados en el registro, para hacer eso necesito hacer un COMMIT

```
git commit -m "primer commit"
```

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (master)
$ git commit -m "primer commit"
[master (root-commit) 14c814f] primer commit
1 file changed, 11 insertions(+)
create mode 100644 index.html
```

¡Felicidades! has salvado tu primer commit en GIT, es un punto en la historia de tu proyecto al cual más adelante podrás regresar en caso de así necesitarlo. Si analizamos un poco la respuesta de la consola nos dice que hay un archivo con cambios y 11 inserciones, se refiere a las 11 líneas del código de HTML, también nos entrega un código hexadecimal el cual es nuestro ID de la snapshot.

Puedo revisar la información corriendo un:

```
git log
```

```
commit 14c814f2a7edd0caf6be3f36d92a07f33b016240 (HEAD -> master)
Author: jobarv <jbarrera@motiontech.com.mx>
Date:   Wed Mar 26 15:40:59 2025 -0600

    primer commit
```

Podemos sintetizar más la información haciendo un:

```
git log --oneline
```

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (master)
$ git log --oneline
14c814f (HEAD -> master) primer commit
```

sigamos haciendo cambios, puedes ver que la rama en la que trabajamos se llama "master", esta es nuestra rama principal, cuando subamos la información a GITHUB esto va a generar un detalle, la rama principal en línea se llama "main". Para que todo concuerde cambiemos el nombre:

```
git branch -M main
```

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (main)  
$
```

Hemos cambiado el nombre de la rama.

Agreguemos la rama de ambiente de desarrollo, le llamaremos "test"

```
git branch test
```

también puedes hacer un:

```
git checkout -b test
```

Esto creará la rama y te colocará en ella

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (test)  
$ |
```

si haces un:

```
git branch
```

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (test)  
$ git branch  
  main  
* test
```

Te muestra las ramas del proyecto.

Recuerda que la buena práctica es trabajar en el ambiente de pruebas o de desarrollo, muy rara vez te tocará hacer cambios directamente en producción (main).

Una vez en la rama TEST, hagamos otro cambio, agreguemos más elementos a nuestro archivo HTML y agreguemos un archivo de estilos CSS.

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <header></header>
  <main></main>
  <footer></footer>
</body>
</html>
```

CSS

```
html{
  background-color: blue;
}
```

si hacemos un git status de nuevo:

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (test)
$ git status
On branch test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    styles.css

no changes added to commit (use "git add" and/or "git commit -a")
```

GIT ha detectado nuevos cambios, pero aún no los registra por qué no los hemos salvado al área de cambios, corremos "git add ." y luego un "git status" y vemos:

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (test)
$ git status
On branch test
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   index.html
    new file:   styles.css
```

Tomemos la segunda snapshot con un segundo commit:

```
git commit -m "se agregan cambios y css"
```

Recuerda hacer las descripciones de los commits lo más intuitivas posibles.

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (test)
$ git commit -m "se agregan cambios y css"
[test ceee617] se agregan cambios y css
2 files changed, 7 insertions(+), 1 deletion(-)
create mode 100644 styles.css
```

Ahora vemos dos archivos y 7 inserciones.

Si revisamos el log nuevamente vemos:

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (test)
$ git log --oneline
ceee617 (HEAD -> test) se agregan cambios y css
14c814f (main) primer commit
```

Ahora tengo dos puntos en mi línea de tiempo, esto en la rama test, si hago un

```
git checkout main
```

y me coloco en main, después hago un git status nuevamente vemos:

```
silve@barras1 MINGW64 /d/Cursos/PracticaGIT (main)
$ git log --oneline
14c814f (HEAD -> main) primer commit
```

en main no se ha registrado el segundo cambio, y es lo que queremos, por qué trabajamos en test mejorando o implementando nuevos cambios, en cuanto estos cambios estén listos los agregamos a la rama de producción.