

Práctica de Git en la consola

1. Configuración inicial

Primero, asegúrate de tener Git instalado (`git --version`).

Configura tu nombre y correo (solo la primera vez):

```
git config --global user.name "Tu Nombre"  
git config --global user.email "tuemail@example.com"
```

2. Crear un repositorio nuevo

Crea una carpeta de práctica y conviértela en un repositorio Git:

```
mkdir git-practica  
cd git-practica  
git init
```

Esto crea un repositorio vacío (`.git/`).

3. Primer archivo y primer commit

Crea un archivo:

```
echo "# Proyecto de práctica con Git" > README.md
```

Agrega el archivo al *staging area*:

```
git add README.md
```

Guarda el cambio en el repositorio (commit):

```
git commit -m "Primer commit: agregar README"
```

4. Revisar historial y estado

Ver estado actual:

```
git status
```

Ver historial de commits:

```
git log --oneline --graph
```

5. Crear y modificar archivos

Agrega un archivo nuevo:

```
echo "print('Hola Git!') > app.py
git add app.py
git commit -m "Agregar archivo app.py con saludo"
```

Edita un archivo:

```
echo "print('Modificado') >> app.py
git status
```

Luego guarda el cambio:

```
git add app.py
git commit -m "Modificar app.py para imprimir mensaje nuevo"
```

6. Deshacer cambios

- Si cambiaste un archivo pero no lo agregaste al *staging area*:

```
git checkout -- app.py
```

- Si lo agregaste pero no hiciste commit:

```
git reset app.py
```

7. Crear y cambiar de ramas

Crear una rama:

```
git branch nueva-funcionalidad
```

Cambiar a esa rama:

```
git checkout nueva-funcionalidad
```

(Alternativa más corta:)

```
git checkout -b nueva-funcionalidad
```

Agrega un cambio en esa rama:

```
echo "print('Nueva funcionalidad')" >> app.py
git add app.py
git commit -m "Agregar nueva funcionalidad en app.py"
```

8. Fusionar ramas (merge)

Regresa a main:

```
git checkout main
```

Fusiona:

```
git merge nueva-funcionalidad
```

Si no hay conflictos, se unirá automáticamente.

9. Conectar con un repositorio remoto

En GitHub (o GitLab/Bitbucket), crea un repo vacío y copia la URL.

Luego:

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

10. Actualizar y colaborar

- Para traer cambios remotos:

```
git pull origin main
```

- Para subir tus commits:

```
git push origin main
```

11. Buenas prácticas

- Haz commits pequeños y descriptivos.
- Usa ramas para nuevas funcionalidades.
- Revisa siempre `git status`.
- Antes de subir, sincroniza con `git pull`.

Ejercicio Guiado de Git

Escenario

Imagina que trabajas en un proyecto con otro desarrollador.

Tendremos una rama principal (`main`) y crearemos dos ramas de trabajo (`feature-a` y `feature-b`).

En cada una se harán cambios **en la misma parte de un archivo** para generar un conflicto, y luego lo resolveremos.

1. Preparar entorno

```
mkdir git-conflictos
cd git-conflictos
git init
```

Crea un archivo base:

```
echo "print('Hola desde main')" > app.py
git add app.py
git commit -m "Commit inicial con app.py"
```

2. Crear y trabajar en `feature-a`

```
git checkout -b feature-a
```

Modifica `app.py`:

```
echo "print('Cambio desde feature A')" >> app.py
```

Guarda el cambio:

```
git add app.py
git commit -m "Feature A: agregar mensaje en app.py"
```

3. Crear y trabajar en `feature-b`

Vuelve a `main` y crea otra rama:

```
git checkout main
git checkout -b feature-b
```

Edita `app.py` de forma distinta (mismo lugar):

```
echo "print('Cambio desde feature B')" >> app.py
```

Guarda el cambio:

```
git add app.py
git commit -m "Feature B: agregar mensaje en app.py"
```

4. Simular conflicto

Ahora intentemos fusionar ambas ramas en `main`.

Primero merge de `feature-a`:

```
git checkout main
git merge feature-a
```

Este debería funcionar sin problemas.

Luego merge de `feature-b`:

```
git merge feature-b
```

Aquí se genera un **conflicto de merge**, porque `feature-a` y `feature-b` modificaron la misma parte de `app.py`.

5. Resolver conflicto

Abre `app.py` y verás algo así:

```
print('Hola desde main')
<<<<< HEAD
print('Cambio desde feature A')
=====
print('Cambio desde feature B')
>>>>> feature-b
```

Decide cómo resolverlo.

Ejemplo de solución final:

```
print('Hola desde main')
print('Cambio desde feature A y B')
```

Marca el conflicto como resuelto:

```
git add app.py
git commit -m "Resolver conflicto entre feature A y feature B"
```

6. Ver historial gráfico

```
git log --oneline --graph --all
```

Verás cómo las ramas se unieron en `main`.

7. Subir a remoto (opcional)

Si quieres practicar con GitHub:

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

Con esto has practicado:

- Crear ramas
- Hacer commits
- Fusionar cambios
- Generar y resolver conflictos