

# Guía Práctica Paso a Paso: Uso de Git en forma local

Esta guía está pensada para que puedas practicar el uso de Git en tu computadora, sin necesidad de conectarte a GitHub. A medida que avances, irás dominando el flujo de trabajo local, los comandos esenciales, el manejo de ramas, la resolución de conflictos y la recuperación de versiones.



## 1. Crear un repositorio local

1. Abrió una terminal o consola de comandos. (*Permite ingresar instrucciones a la computadora usando texto*).
2. Creá una carpeta para tu proyecto: (*Vas a trabajar dentro de esta carpeta*)

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

3. Inicializá el repositorio: (*Transforma la carpeta en un repositorio Git*)

```
git init
```

4. Verificá el estado del repositorio vacío:

```
git status
```



## 2. Primer archivo y commit

1. Creá un archivo: (*Creamos un archivo README.md con un texto inicial*)

```
# En Bash o Git Bash:
```

```
echo "# Mi primer repo" > README.md
```

```
# En PowerShell:
```

```
echo "# Mi primer repo" > README.md
```

```
# En CMD:
```

```
echo # Mi primer repo > README.md
```

2. Consultá el estado: (*Verificamos si hay archivos nuevos o modificados*)

```
git status
```

3. Agregá el archivo al staging: *(Preparamos el archivo para ser guardado en el historial)*

```
git add README.md
```

4. Consultá nuevamente el estado para confirmar:

```
git status
```

5. Hacé el primer commit: *(Guardamos el archivo con un mensaje descriptivo)*

```
git commit -m "Agrega README inicial"
```

6. Verificá el historial:

```
git log --oneline
```



### 3. Agregar un nuevo archivo y hacer otro commit

1. Agregá otro archivo: *(Creamos un script simple en Python)*

```
# En Bash o Git Bash:
```

```
echo "print('Hola mundo')" > hola.py
```

```
# En PowerShell:
```

```
echo "print('Hola mundo')" > hola.py
```

```
# En CMD:
```

```
echo print('Hola mundo') > hola.py
```

2. Consultá el estado: *(Confirmamos que Git detecta el nuevo archivo)*

```
git status
```

3. Agregá y confirmá: *(Lo llevamos al historial con un nuevo commit)*

```
git add hola.py
```

```
git status
```

```
git commit -m "Agrega script hola mundo"
```

4. Consultá el historial actualizado:

```
git log --oneline
```

## ▲ 4. Crear y cambiar a una nueva rama

1. Crear rama: *(Creamos una nueva rama para desarrollar una funcionalidad)*

```
git branch nueva-funcion
```

2. Verificá las ramas disponibles:

```
git branch
```

3. Cambiar de rama: *(Nos movemos a esa nueva rama para trabajar de forma aislada)*

```
git checkout nueva-funcion
```



## 5. Hacer cambios en la nueva rama

1. Editá el archivo `hola.py`: *(Agregamos una nueva línea al script existente)*

# En Bash o Git Bash:

```
echo "print('Esta es una nueva funcionalidad')" >> hola.py
```

# En PowerShell:

```
echo "print('Esta es una nueva funcionalidad')" >> hola.py
```

# En CMD:

```
echo print('Esta es una nueva funcionalidad') >> hola.py
```

2. Verificá el estado:

```
git status
```

3. Guardá los cambios:

```
git add hola.py
```

4. Confirmá el cambio:

```
git commit -m "Agrega nueva funcionalidad"
```

5. Verificá el historial de esta rama:

```
git log --oneline
```



## 6. Volver a la rama principal y comparar

1. Cambiar a `main`: *(Volvemos a la rama principal del proyecto)*

```
git checkout main
```


2. Verificá el historial:

```
git log --oneline
```

3. Mostrá las diferencias:

```
git diff nueva-funcion
```

## 7. Hacer merge desde nueva rama

 Importante: para hacer un merge correctamente, debés estar posicionado en la rama donde querés integrar los cambios (en este caso, `main`).

1. Asegurate de estar en `main`:

```
git checkout main
```

2. Verificá el estado actual del repositorio:

```
git status
```

3. Traé los cambios desde la rama `nueva-funcion`:

```
git merge nueva-funcion
```

4. Confirmá que el historial refleje la fusión:

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

## 8. Generar un conflicto y resolverlo

1. Cambiá una línea en `hola.py` desde `main`, hacé un commit. *(Simulamos que alguien más hizo un cambio en el mismo archivo)*
2. Cambiá a la rama `nueva-funcion`:

```
git checkout nueva-funcion
```

3. Cambiá la misma línea desde `nueva-funcion`, hacé un commit. *(Creamos un cambio incompatible en paralelo)*
4. Cambiá a la rama `main` para intentar hacer el merge:

```
git checkout main
```

5. Intentá un merge:

```
git merge nueva-funcion
```

6. Git marcará un conflicto. Abrió el archivo y verás marcas como estas:

```
<<<<<<< HEAD
print("Cambios en main")
=====
print("Cambios en nueva-funcion")
>>>>>>> nueva-funcion
```

Editá el archivo para resolverlo (dejá solo la versión correcta y eliminá las marcas <<<<<<, =====, >>>>>>). 7. Una vez resuelto:

```
git add hola.py
git status
```

8. Confirmá la resolución del conflicto:

```
git commit -m "Resuelve conflicto entre main y nueva-funcion"
```

9. Verificá que el historial muestre la resolución:

```
git log --oneline
```

## 9. Revertir cambios

### 9.1 Revertir archivo antes del commit

Si modificaste un archivo y aún no hiciste commit: (*Deshacé los cambios y volvé a la última versión guardada*)

```
git restore hola.py
```

### 9.2 Volver a una versión anterior

1. Consultá el historial: (*Lista los commits hechos hasta ahora*)

```
git log --oneline
```

2. Copiá el ID del commit anterior. (*Identificás a qué estado querés volver*)
3. Volvé temporalmente a esa versión: (*Esto no borra el historial, solo te deja ver el proyecto como estaba*)

```
git checkout <id_commit>
```

4. Para volver a la rama: (*Retomás la rama principal*)

```
git checkout main
```

## 10. Conectar tu proyecto local con GitHub

Una vez que ya practicás localmente, podés subir tu trabajo a GitHub para guardarlo en la nube y colaborar con otros.

### Configurar tu nombre de usuario y correo electrónico

Antes de trabajar con Git, es importante que configures tu identidad. Esto se hace una sola vez por computadora (o por repositorio si querés usar diferentes identidades).

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

Podés verificar que se hayan guardado correctamente con:

```
git config --global --list
```

Esto asegura que cada commit que hagas esté correctamente etiquetado con tu nombre y correo.

### Paso 1: Crear un repositorio en GitHub

1. Ingresá a <https://github.com> y creá un nuevo repositorio.
2. Copiá la URL del repositorio (por ejemplo:  
`https://github.com/usuario/mi-proyecto.git`)

### Paso 2: Vincular tu proyecto local con GitHub

Desde tu carpeta del proyecto:

```
git remote add origin https://github.com/usuario/mi-proyecto.git
```

Esto le dice a Git que "origin" será el repositorio remoto.

### Paso 3: Enviar tu trabajo local a GitHub

1. Si tu rama se llama `master`, es buena práctica renombrarla a `main`:

```
git branch -M main
```

2. Subí tus cambios a GitHub:

```
git push -u origin main
```

### Paso 4: Confirmar en GitHub

Ingresa a tu repositorio en GitHub y actualizá la página: vas a ver los archivos que subiste.

## 11. ¿Qué es un fork y cómo trabajar con él?

Un **fork** es una copia completa de un repositorio de GitHub que se crea dentro de tu cuenta. Te permite:

- Probar cambios sin afectar el proyecto original.
- Proponer mejoras enviando los cambios mediante un pull request.
- Trabajar con repositorios a los que no tenés permisos directos.

### ¿Cuándo usar un fork?

Cuando querés contribuir a un proyecto externo (por ejemplo, de código abierto), primero hacés un fork, clonás tu fork, realizás cambios, y luego solicitás que tus cambios sean integrados.

### Cómo hacer un fork de un repositorio

1. Ingresa a un repositorio público de GitHub (por ejemplo, <https://github.com/otro-usuario/proyecto>).
2. Hacé clic en el botón **Fork** (esquina superior derecha).
3. Elegí tu cuenta para crear una copia del proyecto en tu propio GitHub.

### Clonar tu fork en tu PC

Una vez que tengas tu fork:

1. Copiá la URL de tu fork (por ejemplo: `https://github.com/tu-usuario/proyecto.git`).
2. En la terminal, ejecutá:

```
git clone https://github.com/tu-usuario/proyecto.git
```

3. Entrá al directorio:

```
cd proyecto
```

Ahora podés trabajar como si fuera tu propio repositorio. Una vez que hayas hecho cambios y los hayas confirmado (`git add + git commit`), podés subirlos con:

```
git push origin main
```

Y luego enviar un `pull request` desde GitHub para que los responsables del proyecto original revisen y, si lo desean, integren tus cambios.

