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:

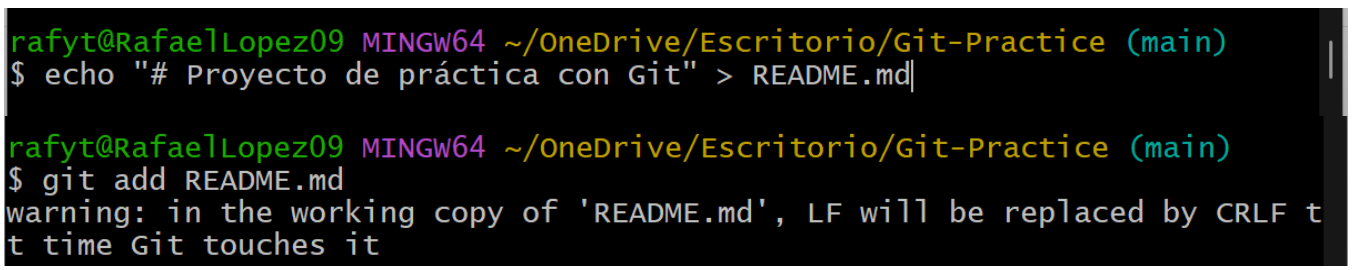
```
echo "# Proyecto de práctica con Git" > README.md
```

Agrega el archivo al *staging area*:

```
git add README.md
```

Guarda el cambio en el repositorio (commit):

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



```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ git commit -m "Primer commit: agregar README"
[main (root-commit) 583169d] Primer commit: agregar README
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

4. Revisar historial y estado

Ver estado actual:

```
git status
```

Ver historial de commits:

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

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ git status
On branch main
nothing to commit, working tree clean

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ git log --oneline --graph
* 583169d (HEAD -> main, origin/main) Primer commit: agregar README
```

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"
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ echo "print('Hola Git!')" > app.py
bash: !': event not found

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ echo "print('Hola Git')" > app.py

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (ma
in)
$ git add app.py
warning: in the working copy of 'app.py', LF will be replaced by C
RLF the next time Git touches it

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (ma
in)
$ git commit -mn "Agregar archivo app.py con saludo"
```

Edita un archivo:

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

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (ma
in)
$ echo "print('Modificado')" >> app.py

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (ma
in)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   app.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working direc
tory)
        modified:   app.py
```

Luego guarda el cambio:

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

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (ma
in)
$ git add app.py
warning: in the working copy of 'app.py', LF will be replaced by C
RLF the next time Git touches it

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (ma
in)
$ git commit -m "Modificar app.py para imprimir mensaje nuevo"
[main 56134f0] Modificar app.py para imprimir mensaje nuevo
 1 file changed, 2 insertions(+)
 create mode 100644 app.py
```

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
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (ma
in)
$ git checkout -- app.py
```

7. Crear y cambiar de ramas

Crear una rama:

```
git branch nueva-funcionalidad
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ git branch nueva-funcionalidad
```

Cambiar a esa rama:

```
git checkout nueva-funcionalidad
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ git checkout nueva-funcionalidad
Switched to branch '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"
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (nueva-funcionalidad)
$ echo "print('Nueva funcionalidad')" >> app.py

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (nueva-funcionalidad)
$ git add app.py
warning: in the working copy of 'app.py', LF will be replaced by CRLF the next time Git touches it

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (nueva-funcionalidad)
$ git commit -m "Agregar nueva funcionalidad en app.py"
[nueva-funcionalidad 04e5984] Agregar nueva funcionalidad en app.py
 1 file changed, 1 insertion(+)
```

8. Fusionar ramas (merge)

Regresa a `main`:

```
git checkout main
```

Fusiona:

```
git merge nueva-funcionalidad
```

Si no hay conflictos, se unirá automáticamente.

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (nueva-funcionalidad)
$ git checkout main
Switched to branch 'main'

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ git merge nueva-funcionalidad
Updating 56134f0..04e5984
Fast-forward
 app.py | 1 +
 1 file changed, 1 insertion(+)
```

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
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (master)
$ git remote add origin https://github.com/PolyNomius09/Git-Practice.git

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (master)
$ ģit branch main
bash: ģit: command not found

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (master)
$ git branch -M main
```

10. Actualizar y colaborar

- Para traer cambios remotos:

```
git pull origin main
```

- Para subir tus commits:

```
git push origin main
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice (main)
$ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 660 bytes | 660.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/PolyNomius09/Git-Practice.git
   583169d..04e5984  main -> 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
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos (main)
$ mkdir git-conflictos

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos (main)
$ cd git-conflictos

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (main)
$ git init
Initialized empty Git repository in C:/Users/rafyt/OneDrive/Escritorio
/Git-Practice/git-conflictos/git-conflictos/.git/
```

Crea un archivo base:

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

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master)
$ echo "print('Hola desde main')" > app.py

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master)
$ git add app.py
warning: in the working copy of 'app.py', LF will be replaced by CRLF
the next time Git touches it

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master)
$ git commit -m "Commit inicial con app.py"
[master (root-commit) e7c634d] Commit inicial con app.py
 1 file changed, 1 insertion(+)
 create mode 100644 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"
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master)
$ git checkout feature-a
Switched to branch 'feature-a'
M       app.py

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (feature-a)
$ echo "print('Cambio desde feature A')" >> app.py

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (feature-a)
$ git add app.py
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (feature-a)
$ git commit -m "Feature A: agregar mensaje en app.py"
[feature-a 171cca6] Feature A: agregar mensaje en app.py
 1 file changed, 2 insertions(+)
```

3. Crear y trabajar en `feature-b`

Vuelve a `main` y crea otra rama:

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

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (feature-a)
$ git checkout master
Switched to branch 'master'

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master)
$ git checkout -b feature-b
Switched to a new branch '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"
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (feature-b)
$ echo "print('Cambio desde feature B')" >> app.py

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (feature-b)
$ git add app.py
warning: in the working copy of 'app.py', LF will be replaced by CRLF
the next time Git touches it

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (feature-b)
$ git commit -m "Feature B: agregar mensaje en app.py"
[feature-b 3025605] Feature B: agregar mensaje en app.py
 1 file changed, 1 insertion(+)
```

4. Simular conflicto

Ahora intentemos fusionar ambas ramas en `main`.

Primero merge de `feature-a`:

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

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (feature-b)
$ git checkout master
Switched to branch 'master'

rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master)
$ git merge feature-a
Updating e7c634d..171cca6
Fast-forward
 app.py | 2 ++
 1 file changed, 2 insertions(+)
```

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`.

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master)
$ git merge feature-b
Auto-merging app.py
CONFLICT (content): Merge conflict in app.py
Automatic merge failed; fix conflicts and then commit the result.
```

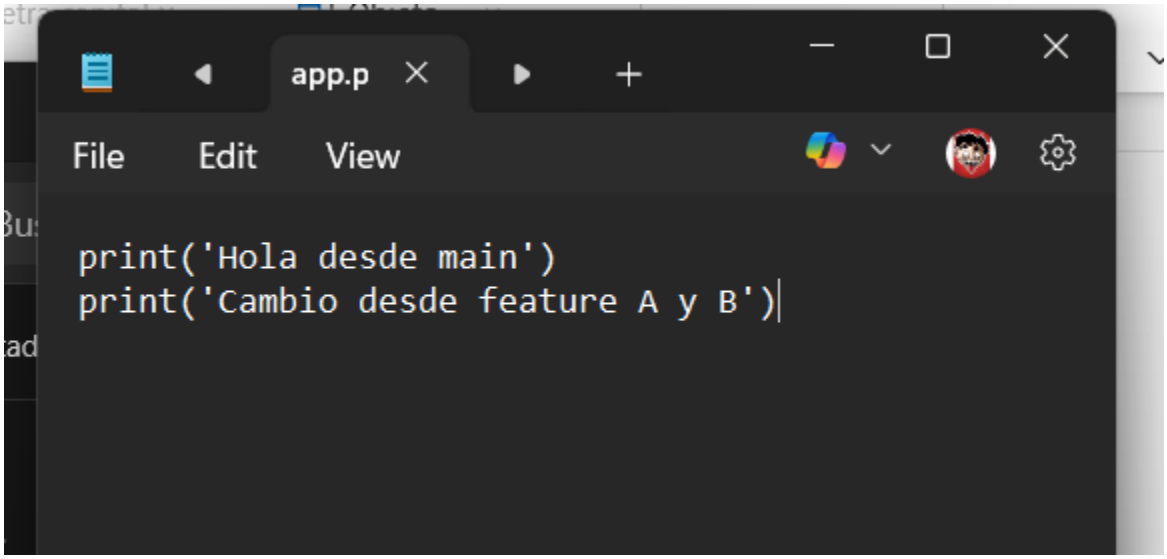
5. Resolver conflicto

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"
```

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master|MERGING)
$ git commit -m "Resolver conflicto entre feature A y feature B"
[master 00ec39d] 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`.

```
rafyt@RafaelLopez09 MINGW64 ~/OneDrive/Escritorio/Git-Practice/git-con
flictos/git-conflictos (master)
$ git log --oneline --graph --all
*   00ec39d (HEAD -> master) Resolver conflicto entre feature A y feat
ure B
|\
| * 3025605 (feature-b) Feature B: agregar mensaje en app.py
* | 171cca6 (feature-a) Feature A: agregar mensaje en app.py
|/
* e7c634d Commit inicial con app.py
```

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