



Uso de Git y Trabajo en Equipo

Objetivo: Aprender a usar Git para trabajar en equipo, conocer los comandos básicos y entender cómo se hace un merge hacia la rama `main` (o `master`), promoviendo buenas prácticas de colaboración en entornos de desarrollo de software.

Introducción

En el desarrollo de software moderno, el trabajo colaborativo y el control de versiones son fundamentales. Git se ha convertido en la herramienta más utilizada a nivel mundial para gestionar el historial de cambios en proyectos de programación. Gracias a Git, los equipos pueden trabajar en paralelo, fusionar sus aportes y llevar un registro detallado de cada modificación en el código fuente.

Este informe presenta una visión general de Git, su utilidad en el trabajo en equipo, los comandos esenciales y cómo manejar situaciones comunes como los conflictos al fusionar cambios.

Conceptos

¿Qué es Git?

Git es un sistema de control de versiones distribuido creado por Linus Torvalds en 2005. Permite a los desarrolladores llevar un historial detallado y ordenado de los cambios realizados en los archivos de un proyecto.



¿Para qué sirve Git?

- Guardar versiones del proyecto.
- Volver a versiones anteriores si ocurre un error.
- Trabajar en equipo sin sobrescribir el trabajo de otros.
- Dividir el desarrollo en ramas (features) y luego integrarlas al proyecto principal.
- Coordinar mejor los aportes de todos los desarrolladores.

¿Cuál es su importancia?

Git permite desarrollar software de forma **segura, ordenada y colaborativa**, evitando errores catastróficos como pérdida de código o sobrescritura accidental. También es clave para el trabajo remoto y la integración continua.

Comandos básicos de Git

A continuación, se listan los comandos más comunes con su explicación:

Comando	Descripción
git init	Inicializa un nuevo repositorio Git en una carpeta.
git status	Muestra el estado actual de los archivos (modificados, no versionados, listos para commit).
git add archivo.txt	Agrega un archivo al área de preparación (staging).
git add .	Agrega todos los archivos modificados.



git commit -m "mensaje"	Registra los cambios con un mensaje descriptivo.
git config --global user.name "Nombre"	Configura el nombre del usuario.
git config --global user.email "correo"	Configura el correo del usuario.
git log	Muestra el historial de commits.
git branch	Lista las ramas locales.
git checkout -b nueva-rama	Crea y cambia a una nueva rama.
git checkout nombre-rama	Cambia a otra rama existente.
git merge nombre-rama	Fusiona otra rama con la rama actual.
git remote add origin URL	Conecta un repositorio local con uno remoto (GitHub, GitLab, etc.).
git push origin rama	Sube los cambios al repositorio remoto.
git pull origin rama	Descarga e integra los cambios más recientes desde el repositorio remoto.

Flujo de trabajo recomendado con Git

1. Clonar el repositorio remoto.
2. Crear una nueva rama para cada funcionalidad.
3. Realizar cambios y confirmarlos (commit).
4. Subir la rama al repositorio remoto.



5. Crear un Pull Request (PR) para revisar y hacer merge con main/master.
6. Resolver conflictos si los hay.
7. Eliminar la rama una vez fusionada.

Conflictos al hacer Merge

Cuando dos ramas modifican las mismas líneas de un archivo, Git no puede decidir qué versión conservar. En ese caso, se genera un **conflicto de merge**, y el desarrollador debe:

- Editar el archivo para resolver el conflicto.
- Eliminar las marcas de conflicto <<<<<, =====, >>>>>.
- Hacer `git add` y luego `git commit` para finalizar la fusión.

Esto promueve la revisión manual del código y mejora la calidad del proyecto.

Git en el trabajo colaborativo

Git no solo guarda versiones del código, también permite a los equipos:

- Trabajar simultáneamente en múltiples funcionalidades.
- Revisar el trabajo de otros mediante Pull Requests.
- Establecer roles y reglas (ej: revisión obligatoria antes de hacer merge).



- Automatizar pruebas y despliegues al integrar con herramientas como GitHub Actions.

Practica 1: Escritura colaborativa de un cuento

Requerimiento funcional:

Escribir un cuento corto titulado “El Bosque Encantado”, dividido así:

- Introducción
- Desarrollo
- Final

Equipo de trabajo

- Ana: Encargada de la introducción.
- Carlos: Encargado del desarrollo.
- Lucía: Encargada del final.

Todos colaboran en el archivo **cuento.txt**

Ramas y distribución

Cada persona trabaja en su propia rama:



- intro-ana
- desarrollo-carlos
- final-lucía

Todos parten desde una rama main vacía con solo el título.

Archivo base (cuento.txt en main)

tendrá la siguiente información: Título: El Bosque Encantado

Nota: El líder del grupo debe crear el repositorio, crear el archivo base y subirlo a la rama principal (main o master). Posteriormente deben enviar a sus compañeros la invitación como colaboradores del repositorio y ellos deben aceptar la invitación.

Avance individual (todos deben trabajar al tiempo; todos excepto Ana, deben clonar el repositorio e ingresar a la carpeta, dentro de dicha carpeta encontrarás la carpeta .git y el archivo txt con el cual vas a trabajar) y realizar lo que se dice a continuación:

- Ana (rama **intro-ana**) escribe:

Título: El Bosque Encantado

Había una vez un bosque donde los árboles susurraban secretos al viento...

- Carlos (rama **desarrollo-carlos**)

edita el mismo archivo y lo deja así:

Título: El Bosque Encantado

Todo cambió cuando un niño curioso entró al bosque y encontró una llave brillante...



Recuerda que debes subir este archivo en tu rama.

Nota: Ambos han editado el mismo archivo y la misma línea siguiente al título, sin saber que el otro también lo hizo.

hacer merge con main...

1. Ana hace merge sin problema:

```
git checkout main
```

```
git merge intro-ana
```

```
git push origin main
```

2. Luego Carlos intenta hacer su merge:

```
git checkout main
```

```
git pull origin main
```

```
git merge desarrollo-carlos
```

3. Git muestra un conflicto en cuento.txt

archivo con conflicto:

Título: El Bosque Encantado

<<<<< HEAD



Había una vez un bosque donde los árboles susurraban secretos al viento...

=====

Todo cambió cuando un niño curioso entró al bosque y encontró una llave brillante...

>>>>> desarrollo-carlos

Resolución del conflicto

Carlos:

- 1 . Se da cuenta del conflicto al leer el mensaje de Git.
- 2 . Llama a Ana y Lucía para revisar el archivo juntas.
- 3 . Deciden conservar ambas partes, ordenadas por estructura narrativa.

Carlos edita el archivo así:

Título: El Bosque Encantado

Había una vez un bosque donde los árboles susurraban secretos al viento...

Todo cambió cuando un niño curioso entró al bosque y encontró una llave brillante...



Elimina las marcas <<<<<, =====, >>>>>.

Luego ejecuta:

```
git add cuento.txt
```

```
git commit -m "Conflicto resuelto: introducción y desarrollo combinados"
```

```
git push origin main
```

Resultado final en main

Título: El Bosque Encantado

Había una vez un bosque donde los árboles susurraban secretos al viento...

Todo cambió cuando un niño curioso entró al bosque y encontró una llave brillante...

Reflexión del equipo

- Ahora Lucía, al escribir el final, primero hace git pull origin main para traer todo lo que ya se ha fusionado.
- El equipo decide dividir el cuento en tres archivos (intro.txt, desarrollo.txt, final.txt) y luego unirlos al final para evitar conflictos.



- También proponen tener un archivo guion.md con la estructura y responsables.

Practica 2: Proyecto colaborativo de un cuento ilustrado

Proyecto: El Secreto del Faro

Estructura del repositorio (carpeta del proyecto):

```
cuento-faro/
├── portada.png      # Ilustración de portada
├── cuento.txt       # Texto del cuento completo
├── diseño-portada.psd # Archivo editable de diseño
└── README.md         # Instrucciones del proyecto
```

Equipo de trabajo

- Laura: Escritora principal, edita cuento.txt.
- David: Diseñador gráfico, trabaja en portada.png y diseño-portada.psd.
- Sofía: Coordinadora, estructura el proyecto y hace revisión final.

Todos trabajan desde el repositorio remoto github.com/equipo/cuento-faro.



FLUJO DEL PROYECTO: 5 COMMITS

Commit 1 – Inicialización (hecho por Sofía)

git init

git add cuento.txt

git commit -m "Inicio del proyecto con estructura base y archivo cuento.txt"

Contenido de cuento.txt:

Título: El Secreto del Faro

[Escribe aquí tu cuento...]

Commit 2 – Laura escribe la introducción

git add cuento.txt

git commit -m "Laura agregó la introducción del cuento"

Nuevo contenido:

Título: El Secreto del Faro

Había una vez un faro abandonado al borde de un acantilado. Nadie sabía su historia...

Commit 3 – David agrega el diseño de portada (correcto)

git add portada.png diseño-portada.psd

git commit -m "David agregó la portada original y su archivo editable"



Archivos agregados:

- portada.png (una imagen bonita del faro)
- diseño-portada.psd (Photoshop editable)

Commit 4 – Laura finaliza el cuento

```
git add cuento.txt
```

```
git commit -m "Laura finalizó el cuento completo"
```

Contenido final de cuento.txt:

Título: El Secreto del Faro

Había una vez un faro abandonado al borde de un acantilado. Nadie sabía su historia...

Un día, el faro volvió a encenderse, guiando a un niño perdido hacia un nuevo destino.

Commit 5 – David accidentalmente daña la portada



David sin querer reemplaza portada.png con otra imagen (borrosa o corrupta), y también guarda mal el .psd.

```
git add portada.png diseño-portada.psd
```

```
git commit -m "Ajustes finales a la portada (sin saber que dañó el archivo)"
```

Problema: Al revisar el repositorio, Lucía nota que la portada está dañada.

SOLUCIÓN: Volver a la versión estable (Commit 4)

Sofía identifica que el último commit fue el problema. Así que ejecuta:

```
git log --oneline
```

Resultado:

e5e5e5e (HEAD -> main) Ajustes finales a la portada ← COMMIT MALO

d4d4d4d Laura finalizó el cuento completo ← COMMIT BUENO

c3c3c3c David agregó la portada original

b2b2b2b Laura agregó la introducción

a1a1a1a Inicio del proyecto

OPCIÓN 1: Revertir el commit dañado

Para mantener el historial pero anular el daño:



```
git revert e5e5e5e
```

Esto genera un nuevo commit que deshace los cambios del anterior sin borrar el historial.

Luego:

```
git push origin main
```

OPCIÓN 2: Volver al commit bueno

Solo si están seguros de descartar lo posterior:

```
git reset --hard d4d4d4d
```

```
git push origin main --force
```

Esto borra el historial después del commit d4d4d4d para todos. Usar con precaución.

Resultado Final

- El repositorio vuelve al estado correcto: portada buena, cuento completo.
- El equipo aprende la importancia de:



- Revisar los archivos antes de hacer commit.
- Etiquetar versiones estables con git tag.
- No sobrescribir archivos importantes sin respaldo.

Consejo adicional: Usar una etiqueta antes del daño

En el commit bueno (d4d4d4d), pudieron haber creado una etiqueta:

```
git tag -a v1.0 -m "Versión estable con portada y cuento terminado"
```

```
git push origin v1.0
```

Y para volver:

```
git checkout v1.0
```

*Análisis y Desarrollo de software -
ADSO
Instructora Isaura Suarez
2025*



Conclusión

Git es una herramienta indispensable en el desarrollo moderno de software. Su dominio permite a los desarrolladores trabajar de forma organizada, prevenir errores, colaborar con efectividad y mantener la trazabilidad del proyecto. Enseñar Git desde el inicio de un proceso formativo asegura que los aprendices adquieran competencias clave para su vida profesional.