Git Workflow

Guía de Flujo de Trabajo y Colaboración con Git

GF

Grupo Flores

Versión 1.0 - Fernando Flores Marzo 2026

1. Introducción

Este documento describe el flujo de trabajo con Git utilizado en los proyectos de Grupo Flores o Alternativos. Su objetivo es asegurar que todos los desarrolladores trabajen de forma organizada, evitando conflictos y manteniendo un historial de cambios claro.

Las reglas aquí descritas deben seguirse al trabajar en cualquier repositorio de la empresa.

2. Conceptos básicos

Repositorio: Proyecto que contiene el código y su historial de cambios.

Rama (branch): Permite trabajar en una funcionalidad sin afectar la rama principal.

Commit: Registro de cambios realizados en el código.

Pull Request (PR): Solicitud para integrar cambios de una rama hacia otra.

Produccion: Codigo funcionando para usuarios reales

3. Flujo de trabajo

Cuando un proyecto comienza, normalmente se crea un repositorio con una rama principal llamada **main/master**. Esta rama contiene el estado actual del proyecto y representa la base del desarrollo.

Antes de comenzar a trabajar es importante asegurarse de tener la versión más reciente del proyecto.

```
git checkout main
git pull origin main
```

Con esto cambiamos a la rama principal y descargamos los cambios más recientes del repositorio remoto.

Una vez actualizado el proyecto, el siguiente paso es crear una nueva rama para trabajar en la funcionalidad o cambio que se desea desarrollar.

```
git checkout -b feature/login-system
```

Este comando crea una nueva rama llamada **feature/login-system** y automáticamente cambia el entorno de trabajo a esa rama. A partir de este momento todos los cambios se realizan dentro de esta rama y no afectan directamente a **main**.

Mientras se desarrolla la funcionalidad, se realizan cambios en el código y se registran mediante commits.

```
git add .
git commit -m "feat: add login validation"
```

El comando **git add .** prepara los archivos modificados y el commit registra los cambios en el historial del proyecto.

Cuando la rama ya tiene avances, es necesario subirla al repositorio remoto para que esté disponible para el resto del equipo.

```
git push origin feature/login-system
```

Esto crea la rama en el repositorio remoto y permite posteriormente abrir un Pull Request.

Mientras se trabaja en una rama, es posible que otros desarrolladores estén realizando cambios en la rama principal del proyecto. Para mantener la rama actualizada y evitar conflictos más adelante, se recomienda traer periódicamente los cambios más recientes de **main**.

```
git pull origin main
```

Este comando descarga los cambios de la rama principal y los integra en la rama actual en la que se está trabajando.

Una vez que la funcionalidad está terminada y el código ha sido probado, el último paso es crear un Pull Request desde la rama de trabajo hacia **main**. Esto permite que los cambios sean revisados antes de integrarse al proyecto principal.

Resumen del flujo de trabajo

```
git checkout main
git pull origin main

git checkout -b feature/new-feature

git add .
git commit -m "feat: add new feature"

git push origin feature/new-feature
```

4. Errores comunes trabajando con Git

Cuando se trabaja en equipo con Git es común cometer algunos errores que pueden generar conflictos en el repositorio o afectar el trabajo de otros desarrolladores. A continuación se describen algunas situaciones frecuentes y cómo evitarlas.

4.1 Hacer cambios directamente en main

Uno de los errores más comunes es trabajar directamente en la rama principal del proyecto.

```
git checkout main
```

Realizar cambios directamente en esta rama puede afectar el estado estable del proyecto y generar conflictos cuando otros desarrolladores intenten integrar sus cambios.

Por esta razón, todas las nuevas funcionalidades o correcciones deben desarrollarse en una rama independiente.

Ejemplo correcto:

```
git checkout -b feature/login-system
```

4.2 No actualizar el repositorio antes de trabajar

Si se comienza a trabajar sin traer los cambios más recientes del repositorio remoto, es posible que se generen conflictos más adelante al intentar integrar el código.
Antes de empezar a trabajar siempre se debe actualizar la rama principal.

```
git checkout main
git pull origin main
```

4.3 Hacer commits demasiado grandes

Otro error común es acumular muchos cambios y hacer un solo commit grande.

Esto dificulta entender el historial del proyecto y hace más complicado revisar el código.

Un commit debe representar un cambio específico y fácil de entender.

Ejemplo de commit claro:

```
git commit -m "feat: add login validation"
```

4.4 Usar mensajes de commit poco claros

Mensajes como estos no ayudan a entender qué cambió en el proyecto.

```
git commit -m "changes"
git commit -m "fix"
git commit -m "update"
```

Un commit debe describir claramente el cambio realizado.

Ejemplo correcto:

```
git commit -m "fix: resolve password
validation error"
```

4.5 No subir la rama al repositorio

A veces un desarrollador trabaja en una rama local pero olvida subirla al repositorio remoto. Esto impide que el resto del equipo vea el progreso o revise los cambios.

Después de realizar commits es importante subir la rama.

```
git push origin feature/login-system
```

Esto hace que la rama esté disponible para el equipo y permite abrir un Pull Request.

4.6 No traer cambios de main mientras se trabaja

Si una rama se mantiene mucho tiempo sin actualizarse con la rama principal, es probable que se generen conflictos grandes al momento de integrar los cambios.

Para evitar esto se recomienda traer periódicamente los cambios de **main**.

```
git pull origin main
```

Esto mantiene la rama sincronizada con el proyecto principal.