

Flujo de Trabajo para Colaboradores (Iteración Diaria/Por Característica):

1. Mantener tu Rama Principal (local) Actualizada:

- **Propósito:** Antes de empezar cualquier trabajo nuevo, asegúrate de que tu copia local de la rama principal (`main` o `master`) esté sincronizada con la última versión del repositorio remoto en GitHub. Esto es vital para trabajar con la base de código más reciente y minimizar conflictos.
- **Comandos:**

Bash



```
git checkout main # o master (cambia a tu rama principal)
git pull origin main # o master (descarga los últimos cambios del remoto)
```

`git pull` es un atajo para `git fetch` (descargar cambios) seguido de `git merge` (fusionar los cambios descargados).

2. Crear una Nueva Rama para tu Trabajo (Feature Branch):

- **Propósito:** Nunca trabajes directamente en `main` (o `master`). Crea una rama nueva y descriptiva para cada nueva característica, corrección de bug, o tarea. Esto aísla tus cambios y permite una revisión independiente sin afectar el código principal.
- **Comando:**

Bash



```
git checkout -b nombre-de-tu-rama # Ej: feature/añadir-pagina-perfil, fix/c
```

Esto crea la rama y te cambia a ella inmediatamente.

3. Desarrollar y Realizar Cambios:

- **Propósito:** Escribe tu código, modifica archivos, agrega nuevas funcionalidades o corrige errores dentro de esta nueva rama.
- **Acción:** Trabaja en tu editor de código (VS Code, etc.).

4. Guardar tus Cambios Localmente (Commit):

- **Propósito:** Una vez que hayas completado una unidad lógica de trabajo (no esperes a terminar toda la característica, haz commits pequeños y frecuentes), guarda esos cambios en el historial de tu rama local.
- **Comandos:**

Bash



```
git status # Para ver qué archivos han sido modificados o son nuevos
git add . # O git add nombre-del-archivo (para agregar los cambios al área
git commit -m "Mensaje descriptivo del commit" # Ej: "feat: Añadido formula
```

Consejo: Los mensajes de commit claros y concisos son muy importantes.

5. Subir tus Cambios a tu Rama Remota (Push):

- **Propósito:** Sube los commits que has hecho en tu rama local a tu repositorio en GitHub. Esto hace que tus cambios sean visibles para otros y los respalda.
- **Comando:**

Bash



```
git push origin nombre-de-tu-rama
```

Si es la primera vez que subes esa rama, Git podría sugerirte `git push --set-upstream origin nombre-de-tu-rama` (o `-u` para abreviar). Sigue la sugerencia.

6. Crear un "Pull Request" (PR) o "Merge Request":

- **Propósito:** Notifica a los mantenedores del proyecto y a otros colaboradores que tus cambios están listos para ser revisados e integrados en la rama principal (`main` o `master`).

- **Acción:**

- Ve a la página del repositorio en GitHub.
- Generalmente, GitHub detectará que has subido una nueva rama y te ofrecerá un botón para "Compare & pull request" o "Create pull request".
- Haz clic en él.
- **Proporciona un título y una descripción clara de tu PR.** Explica qué problema resuelve, qué funcionalidad añade, y cómo se puede probar. Si tu proyecto tiene un sistema de seguimiento de incidencias, enlaza el PR a la incidencia correspondiente.

7. Participar en la Revisión de Código:

- **Propósito:** Otros miembros del equipo revisarán tu código, harán comentarios, sugerirán mejoras o pedirán aclaraciones. Este es un paso fundamental para la calidad del código y el aprendizaje mutuo.
- **Acción:** Responde a los comentarios. Si se solicitan cambios:

- Vuelve a tu rama local (`git checkout nombre-de-tu-rama`).
- Realiza los cambios.
- `git add .`
- `git commit -m "fix: Implementado feedback de revisión de PR"` (o un mensaje similar).
- `git push origin nombre-de-tu-rama`
- Los nuevos commits se añadirán automáticamente a tu PR abierto en GitHub.

8. Resolver Conflictos (si los hay):

- **Propósito:** Si mientras trabajabas en tu rama, otros colaboradores fusionaron cambios a `main` (o `master`) que afectan los mismos archivos que tú modificaste, surgirán "conflictos de fusión".

- **Acción:**

- **Antes de que tu PR sea fusionado (o si quieres integrar cambios recientes de `main` en tu rama):**

Bash



```
git checkout nombre-de-tu-rama
git pull origin main # Esto intentará fusionar los cambios de main en tu rama
```

Si hay conflictos, Git te lo notificará. Tendrás que abrir los archivos con conflictos, resolverlos manualmente (buscando los marcadores `<<<<<<`, `=====`, `>>>>>>`), guardar los archivos, y luego:

Bash



```
git add . # Para marcar los conflictos como resueltos
git commit -m "Merge main into feature/nombre-de-tu-rama" # Un mensaje de commit
git push origin nombre-de-tu-rama # Subir los conflictos resueltos
```

- **Alternativa a `pull` (usando `rebase`):** Algunos equipos prefieren `rebase` para mantener un historial más lineal, pero es más avanzado.

Bash



```
git checkout nombre-de-tu-rama
git rebase main # Reaplicar tus commits sobre la punta de main
# Si hay conflictos, resolverlos, luego git add . y git rebase --continue
git push --force-with-lease origin nombre-de-tu-rama # IMPORTANTE: rebase
```

¡Usa `rebase` con precaución y solo si entiendes sus implicaciones!

9. Fusión del "Pull Request" (Merge):

- **Propósito:** Una vez que el código ha sido revisado y aprobado, un mantenedor del repositorio (o tú mismo si tienes los permisos y está permitido) fusionará tu rama con la rama `main` (o `master`).
- **Acción:** Se hace desde la interfaz de GitHub (botón "Merge pull request").

10. Limpiar Ramas (Opcional, pero recomendado):

- **Propósito:** Después de que un PR ha sido fusionado, la rama de característica ya no es necesaria. Borrarla mantiene tu lista de ramas limpia.
- **Acción:**
 - En GitHub, después de fusionar, hay un botón para "Delete branch".
 - Localmente:

Bash



```
git checkout main # o master
git pull origin main # o master (para asegurarte de que tu rama principal
git branch -d nombre-de-tu-rama # Elimina la rama local (el -d solo per
# Si quieres borrar la rama remota que aún puede existir en GitHub (si
# git push origin --delete nombre-de-tu-rama
```

Este ciclo se repite cada vez que un colaborador trabaja en una nueva característica o corrección. Este proceso, aunque parece largo, se vuelve una segunda naturaleza con la práctica y es esencial para un desarrollo colaborativo eficiente y sin problemas.