FLUJO COLABORATIVO EN GITHUB DESDE LA TERMINAL

1 Resumen del funcionamiento de las ramas

En un proyecto colaborativo con Git, las ramas (*branches*) son entornos de trabajo separados que permiten a los desarrolladores trabajar en diferentes funcionalidades o tareas sin interferir con el trabajo de los demás. El flujo típico implica tener una rama principal (por ejemplo, `main`) donde solo se fusiona código completamente funcional y una rama de desarrollo (por ejemplo, `develop`) donde se integran nuevas características antes de pasar a producción. Cada desarrollador trabaja en una rama propia, la cual se deriva de `develop`.

2 Diccionario de términos

Clone Obtener una copia local del repositorio desde GitHub. Se hace solo una vez al inicio.

Branch Ramas, o líneas de desarrollo, que permiten trabajar en nuevas funcionalidades sin afectar el código principal.

Checkout Cambiar entre ramas en el repositorio local.

Pull Obtener los últimos cambios de una rama en GitHub hacia tu repositorio local.

Push Subir tus cambios locales a GitHub.

Commit Guardar tus cambios localmente en el historial de Git.

Merge Fusionar los cambios de una rama en otra.

Rebase Reorganizar los cambios de una rama sobre otra para mantener un historial más limpio.

Pull Request (PR) Solicitud para fusionar tu rama con otra en GitHub, generalmente usada para integrar tu trabajo con `develop` o `main`.

Stash Guardar cambios no confirmados temporalmente para poder cambiar de rama sin perder el trabajo.

Fetch Descargar las referencias de las ramas remotas sin aplicarlas en tu rama actual.

Upstream Repositorio original de donde se derivó un fork.

Origin El repositorio remoto por defecto que se asocia con tu repositorio local.

3 Crear un entorno virtual y gestionar dependencias

Es recomendable trabajar con un entorno virtual en Python para aislar las dependencias de tu proyecto. Esto se asegura de que las bibliotecas instaladas para un proyecto no entren en conflicto con las de otro proyecto. Para crear un entorno virtual:

```
# Crear un entorno virtual
python -m venv nombre-del-entorno

# Activar el entorno en Windows
nombre-del-entorno\Scripts\activate
```

```
# Instalar las dependencias desde un archivo requirements.txt
pip install -r requirements.txt
```

4 Archivo .gitignore

El archivo `.gitignore` es necesario en proyectos colaborativos, ya que permite excluir archivos que no deben ser incluidos en el control de versiones, como archivos específicos de entorno o grandes archivos binarios. Un ejemplo típico de un `.gitignore` para un proyecto de Python podría ser:

```
# Ignorar entornos virtuales
venv/
env/

# Ignorar archivos de sistema
.DS_Store
Thumbs.db

# Ignorar archivos de configuración de editor
.vscode/
.idea/

# Ignorar archivos de dependencias
__pycache__/

# Ignorar archivos grandes no deseados
*.log
*.sqlite3
*.bak
```

Es fundamental mantener actualizado el archivo `.gitignore` para evitar que archivos innecesarios se suban al repositorio, lo que puede afectar el rendimiento y la organización del proyecto.

5 Flujo de trabajo

5.1 Clonar el repositorio

Cada desarrollador clona el repositorio solo una vez al inicio del proyecto. Esto crea una copia local completa del repositorio en tu máquina:

```
git clone https://github.com/tu-usuario/nombre-del-repo.git
cd nombre-del-repo
```

5.2 Crear o moverte a tu rama

- Si tu rama aún no existe y quieres crearla desde `develop`:

```
git checkout develop
git pull origin develop  # Asegurarse de tener la última versión
git checkout -b nombre-rama-de-feature  # Crear y cambiar a la nueva rama
git push -u origin nombre-rama-de-feature  # Subir la nueva rama a GitHub
```

5.3 Trabajar en la rama (USO DIARIO)

Si la rama ya fue creada previamente (por ti o por otro colaborador), simplemente muévete a ella:

```
git checkout nombre-rama-de-feature
git pull origin nombre-rama-de-feature  # Obtener la última versión de la rama
```

Una vez que estés en tu rama, puedes comenzar a trabajar en los archivos del proyecto. Dependiendo de los cambios que hayas realizado, puedes optar por añadir todos los archivos modificados o solo algunos específicos antes de hacer un `commit`. Los comandos principales a utilizar son:

```
# Añadir todos los archivos modificados al índice
git add .

# O bien, añadir archivos específicos al índice
git add archivo1.py archivo2.txt

# Guardar los cambios con un mensaje descriptivo
git commit -m "Descripción de los cambios realizados"

# Subir los cambios locales a GitHub
git push origin nombre-rama-de-feature
```

Es recomendable hacer commits frecuentes y con mensajes claros para mantener un buen historial de cambios.
Si solo quieres añadir algunos archivos específicos, utiliza `git add` seguido del nombre de cada archivo, en lugar de `git add .`, que añade todos los archivos modificados. Esto puede ser útil si aún no quieres subir algunos cambios o si necesitas dividir el trabajo en varios `commits` con descripciones detalladas.

5.4 Sincronizar con la rama `develop`

Es importante asegurarse de que la rama en la que estás trabajando esté sincronizada con `develop`. Esto se hace periódicamente para evitar conflictos en el momento de fusionar las ramas.

- Para fusionar los cambios recientes de `develop` en tu rama:

```
git checkout develop
git pull origin develop  # Obtener los últimos cambios de develop
git checkout nombre-rama-de-feature
git merge develop  # Integrar los cambios de develop en tu rama
```

- Si prefieres rebase en lugar de merge, puedes hacerlo así:

```
git checkout nombre-rama-de-feature
git pull origin develop --rebase  # Rebase sobre develop
```

5.5 Crear un Pull Request (PR)

Una vez que hayas terminado tu funcionalidad o tarea, y después de verificar que tu rama está actualizada con `develop`, debes crear un Pull Request. Este paso es esencial para la revisión de código por parte del equipo y para integrar tu trabajo en `develop` o `main`. Para subir tus últimos cambios a GitHub:

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

El *Pull Request* no se crea automáticamente al hacer un `push`, sino que debes hacerlo manualmente desde la interfaz de GitHub. Una vez que el código está en GitHub, accede a la página del repositorio y selecciona tu rama. Luego, busca la opción de `Pull Request` y sigue las instrucciones.

Antes de crear un Pull Request, asegúrate de que tu código haya pasado las pruebas y no rompa la funcionalidad existente.

5.6 Resolver conflictos de merge

En algunos casos, cuando trates de fusionar cambios, pueden aparecer conflictos si diferentes desarrolladores modificaron las mismas líneas de código en sus ramas. Git te avisará si esto ocurre y deberás resolver los conflictos manualmente.

Después de resolver los conflictos, sigue estos pasos:

```
git add archivos-conflitivos  # Después de resolver el conflicto
git rebase --continue  # O si estás haciendo un merge:
git commit  # Completa el commit después de resolver los conflictos
```

Es muy importante probar el código después de resolver conflictos para asegurarse de que todo funcione correctamente.

5.7 Actualizar tu repositorio local

Si hay nuevos cambios en `develop` o `main`, no es necesario clonar de nuevo el repositorio. Simplemente obtén los últimos cambios con:

```
git checkout develop  # Moverse a la rama develop
git pull origin develop  # Obtener los últimos cambios
```

Asegúrate de mantener tu rama sincronizada periódicamente para evitar grandes conflictos al final del proceso de desarrollo.

6 Manejo de cambios en la rama `main`

Si la rama `main` ha sido actualizada con cambios nuevos y tienes cambios locales en tu rama, sigue estos pasos para incorporar los cambios:

1. **Trae los últimos cambios de `main`** a tu repositorio local:

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

2. **Cambia a tu rama de trabajo** y fusiona los cambios de `main` en tu rama:

```
git checkout nombre-rama-de-feature
git merge main  # O bien puedes hacer rebase si prefieres mantener un historial lineal:
git rebase main
```

Si has hecho cambios en tu rama y también han cambiado cosas en `main`, es posible que surjan conflictos. Resuélvelos como se indicó en la sección de conflictos.