Guía de trabajo colaborativo en GitHub

Introducción

En este artículo vamos a abordar el flujo de trabajo recomendado para colaborar en un proyecto utilizando Git y GitHub.

Imaginemos un escenario donde trabajas en equipo con otro compañero y necesitas organizarte para trabajar en el mismo proyecto, evitando conflictos y asegurando que ambos estén al tanto de los cambios. Para este ejemplo, llamaremos a los dos compañeros Alumno A y Alumno B.

Paso 1: Creación del repositorio en GitHub

El alumno A crea un repositorio en GitHub. Este repositorio debe ser privado para que solo él y su compañero puedan verlo y trabajar en él.

Luego, Alumno A va a la sección Settings -> Collaborators en el repositorio e invita a Alumno B como colaborador. Alumno B debe aceptar la invitación que le llegará por correo o desde la propia página de GitHub.

Paso 2: Clonar el repositorio

Una vez que ambos tienen acceso al repositorio, cada uno debe clonar el repositorio en su ordenador con el comando `git clone <URL-del-repositorio>`.

Esto crea una copia exacta del proyecto en tu ordenador local, permitiéndote trabajar en el código sin interferir directamente en la versión en GitHub.

Paso 3: Crear una rama para cada tarea

Para evitar conflictos y trabajar de forma ordenada, cada tarea debe realizarse en una nueva rama. Las ramas permiten que cada uno trabaje en paralelo sin afectar el trabajo del otro.

Por ejemplo, si Alumno B va a trabajar en el diseño del pie de página (*footer*), debería crear una rama específica para ello: `git checkout -b footer-design`.

Este comando crea y cambia a una nueva rama llamada *footer-design*. Los cambios que Alumno B realice aquí no afectarán a la rama principal (*main*).

Paso 4: Trabajar en tu rama

Mientras trabajas en tu rama harás cambios y crearás commits. Incluso, podrías crear ramas nuevas ramas. Si necesitas hacer una pequeña tarea o prueba dentro de tu rama sin afectar directamente los cambios, puedes crear “subramas” dentro de tu rama de trabajo. Esto es útil para mantener orden si estás realizando una mejora o experimento en tu parte del proyecto y quieres probar algo antes de integrarlo en tu rama principal de trabajo.

Por ejemplo, si Alumno B está trabajando en la rama *footer-design* y necesita hacer pruebas con los colores, puede crear una la subrama *test-colors*. Esto creará la subrama *test-colors* a partir de *footer-design*. Ahora, Alumno B puede hacer cambios en esta subrama y, cuando termine, integrarlos en *footer-design* con un *merge*, cuyo proceso ya conoces.

1. Cambia a tu rama principal (*footer-design*): `git checkout footer-design`
2. Fusiona la subrama en tu rama principal de trabajo: `git merge test-colors`

Este proceso permite experimentar sin modificar directamente la rama de trabajo principal. Cuando termines y hayas fusionado la subrama, puedes borrarla con el comando: `git branch -d test-colors`.

Paso 5: Subir la Rama a GitHub

Cuando hayas terminado los cambios en tu rama, es hora de subirlos al repositorio de GitHub. Esto se hace con el comando: `git push origin footer-design`. Esto sube la rama *footer-design* al repositorio remoto en GitHub. Ahora, Alumno A puede ver esta nueva rama y revisar los cambios.

Paso 6: Crear una *pull request* (PR)

Cuando la rama está en GitHub, GitHub te sugerirá crear una *Pull Request* (PR) con el botón *Compare & Pull Request*. Este botón aparece cuando subes una rama por primera vez.

Una *Pull Request* es una solicitud para fusionar los cambios de una rama (en este caso, *footer-design*) en la rama principal (*main*). Al abrir una PR puedes escribir una descripción de los cambios realizados, lo que permite que tu compañero revise y comente.

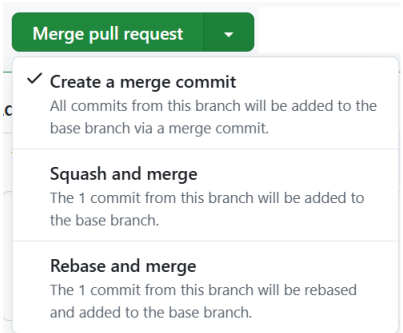
Cuando Alumno B hace clic en el botón *Compare & Pull Request*, GitHub mostrará una interfaz para comparar los cambios entre *footer-design* y *main*.

Los beneficios de hacer una *pull request* son varios:

- Permite a Alumno A revisar los cambios y asegurarse de que todo está correcto antes de fusionarlos en *main*.
- Da la oportunidad de hacer comentarios o pedir correcciones si es necesario.

Paso 7: Fusionar la rama

Cuando la *pull request* esté lista para fusionarse, verás un botón en GitHub que dice *Merge pull request*.



Al hacer clic en el botón, tienes tres opciones:

1. **Create a merge commit**: Esta opción crea un nuevo commit que fusiona toda la rama *footer-design* en *main*. Es útil porque mantiene todos los commits detallados en el historial.
2. **Squash and merge**: Combina todos los commits de la rama en un solo commit en *main*. Esto limpia el historial si se hicieron muchos commits pequeños (por ejemplo, pruebas o ajustes menores).
3. **Rebase and merge**: Coloca los commits de la rama directamente sobre *main*, reorganizando el historial sin crear un nuevo commit de fusión. Esto es útil para mantener un historial lineal, pero puede ser confuso para principiantes.

La primera opción, *Create a merge commit*, es la opción más sencilla y recomendada, ya que mantiene un registro claro de cada cambio.

Paso 8: Actualizar el repositorio local

Una vez que se ha fusionado la *pull request*, ambos alumnos deben actualizar su copia local del repositorio para asegurarse de tener los últimos cambios: `git pull origin main`

Resumen del flujo de trabajo

1. Crear el repositorio: Alumno A lo crea y añade a Alumno B como colaborador.
2. Clonar el repositorio: Ambos clonan el repositorio en sus máquinas locales.
3. Crear una rama por tarea: Cada tarea va en una nueva rama para evitar conflictos.
4. Trabajar en tu rama: Realizar commits regulares ayuda a organizar los cambios. Puedes crear subramas para organizar mejor tu trabajo.
5. Subir la rama a GitHub: Cuando termines una tarea, haz `push` de tu rama.
6. Abrir una *Pull Request*: Solicita la fusión de la rama en *main* mediante un PR.
7. Fusionar la rama: Elige la opción adecuada para fusionar los cambios en *main*.
8. Actualizar el repositorio local: Haz `pull` para mantener tu copia local actualizada.

Este flujo de trabajo ayuda a evitar errores y facilita la colaboración. Con el tiempo, estos pasos se volverán automáticos y verás que trabajar en equipo con Git y GitHub es eficiente y ordenado.