GIT | Tips de Buenas prácticas

1. Nombre de las ramas.

Siempre que empecemos una nueva tarea, crearemos una rama con el siguiente formato:

<**tipo**>/<**nombre**>

- **Tipo**: feature, bugfix, hotfix o refactor

  feature: Cuando se trata de un desarrollo nuevo

  bugfix: Cuando se trata de la resolución de una incidencia sobre la rama principal

  hotfix: Cuando se trata de una depuración de código, por ejemplo, tras encontrar una incidencia en la rama principal.

  refactor: Cuando se trata de una refactorización del código. IMPORTANTE: Hay que recordar que una refactorización no debe incluir una funcionalidad nueva sobre lo que ya existe.

- **Nombre**: algo significativo que identifique qué estamos haciendo en nuestra rama. **NO** debe contener el nombre del desarrollador.

La nueva rama debe partir de la rama principal y no desde una subrama. Git permite la integración de diversas funcionalidades por lo que hay que aprovechar esto para separar las ramas ya que se podrían dar múltiples escenarios:

- Que una rama secundaria no llegue a promocionar, ya sea por cambios de alcance, mal planteamiento, etc.
- Que surjan conflictos durante un merge. Es más fácil y seguro resolver un conflicto por merge varias veces sobre la rama principal cuando, por cada merge, se está integrando una única funcionalidad que, resolver un conflicto una única vez, pero donde se incluyen varias funcionalidades una vez.
- Que se tenga que hacer un rollback (vuelta atrás) de una funcionalidad, pero mantener otras.

Nota: Espero que esto último no se le presente al lector porque aun siguiendo buenas prácticas es un dolor de cabeza. Generalmente se suele preferir crear una rama bugfix para solventar el error o incluso reprogramar la funcionalidad para dejarla en el estado anterior. Sin embargo, y reincidiendo en las recomendaciones, el haber separado la funcionalidad y el merge/pull request, puede ayudar volver a visitar esta solicitud para ver qué se cambió y rehacerlo.

2. Commit.

Cuando se lleva a cabo la realización de un commit, debe ir SIEMPRE con un comentario que responda a las preguntas “por qué” y “qué”. Hay que asumir que la persona que acceda al commit no va a saber qué desarrollo incluye dicho commit.

3. Merge.

Para facilitar la revisión de código, los merge request/pull request deben seguir estos principios:

1. Ser frecuentes y de tamaño reducido.
2. Abarcar una única funcionalidad.

Cuando sea posible, es buena práctica que la persona que revise el código, se pase a la rama en revisión y compruebe la funcionalidad.

4. Subida de código.

Cuando terminemos una tarea SIEMPRE debemos testearla. Una vez estemos seguros de que funciona correctamente, debemos crear un merge request o pull request, en el Git que trabajemos, de la siguiente manera:

1. Compilar el código justo antes de commitear.
2. Hacer "commit" en nuestra propia rama y luego "push".
3. Si lo hemos podido subir sin errores, debemos solicitar un “Merge Request” o “Pull Request”.
4. Algunos detalles a tener en cuenta:

    - Por lo general, la rama objetivo es *develop*.

    - Seguir el formato del texto de commit especificado en los apartados anteriores.

    - Asignar el merge request a una persona con permisos suficientes.

    - Antes de crearlo definitivamente, asegurarnos de que hemos subido los archivos adecuados.

5. Debemos estar atentos a posibles comentarios de nuestro responsable por si tenemos que corregir algo.

Referencias

https://www.freecodecamp.org/news/how-to-write-better-git-commit-messages/amp/