**­­­­MÚLTIPLES ENTORNOS DE TRABAJO**

**Múltiples variantes del repositorio**

Git tiene la opción de poder trabajar en distintas ramas/bifurcaciones para poder tener distintas versiones de un proyecto sin que influya en la rama principal.

**Master** es la **rama principal**, en la que generalmente está situado el proyecto final, sin embargo podemos generar distintas ramas para agilizar el trabajo en equipo y no repercutir directamente en la rama principal.

Para crear una rama utilizaremos el comando **git branch**, tal que así: **git branch [nombre rama].**

* **git branch -l** -> Nos permite listar las ramas de nuestro repositorio.
* **git branch -d [nombre rama]** -> Nos permite eliminar una rama. Si en tu rama tienes cambios, git te pedirá que escribas **-D** en vez de -d para poder eliminar la rama.
* **git branch -m [nombre a renombrar] [nuevo nombre rama]** -> Podemos renombrar una rama con la propiedad -m y, especificando qué rama queremos cambiar y cuál es el nuevo nombre que queremos para la rama.

**Moviéndonos entre ramas y commits**

Para movernos entre ramas tenemos que utilizar el comando **git checkout**. Con este comando podemos movernos específicamente a la rama o commit indicado.

A diferencia de git reset, git checkout nos ofrece la potente herramienta de movernos en el tiempo sin eliminar absolutamente nada. Esto nos permite regresar a otro commit donde quizás introducimos algún error y, podemos crear una rama para poder trabajar sobre este error para no generar conflictos.

Para crear una rama al regresar en un commit específico podemos utilizar el comando **git checkout -b [nombre de la rama]**. Automáticamente nos generará una nueva rama con el nombre indicado y nos moverá allí con el mismo commit en el que estábamos.

**Mezclando ramas y resolviendo conflictos**

Para mezclar ramas y su contenido, git nos proporciona una herramienta, **git merge**. Con git merge podemos combinar el contenido de dos ramas sin tener que hacerlo nosotros manualmente.

Para hacer un merge tenemos que situarnos en la rama que queremos mezclar y hacer un git merge con la rama que pretendemos usar.

Ej: Estoy en la rama “master” y quiero hacer un merge de todo lo que tengo en la rama “responsive” dentro de “master”. git:(master) -> **git merge responsive**

Hay tres tipos de merge:

* **fast-forward**: Hace la mezcla automáticamente sin modificar datos.
* **recursive/auto-merging**: Hace la mezcla de forma recursiva. Te pide hacer un commit, guardas el archivo en vim y listo.
* **Merge manual (conflicto)**: Se han modificado las mismas líneas entre las ramas. Es necesario abrir el archivo y corregir manualmente el código indicado. (Si utilizamos atom tiene una interfaz muy intuitiva con botones para decidir con qué código quedarte). Después de ejecutar los cambios, hacemos un commit y listo.

**Reescribe la historia de tu proyecto**

Algunos desarrolladores no les gusta que un repositorio esté lleno de ramas y les gusta tener la línea del tiempo o estructura del repositorio como una sola línea en la rama principal. Git nos proporciona una herramienta para poder reescribir la historia del proyecto para dejar sin ninguna bifurcación el repositorio, estamos hablando de **git rebase**.

Generalmente el uso de git rebase y reescribir el historial de commits no es una buena práctica, sobretodo cuando trabajas con un equipo con más desarrolladores, porque puedes influir en los demás compañeros y en su trabajo. Si queremos dar uso de git rebase, se recomienda utilizarlo solo en local.

Tal y como hemos hecho antes con el merge, tenemos que situarnos en la rama con la que nos queremos quedar y especificar la rama que queremos unir.

* **git rebase [rama]**

Si añadimos el flag **-i** especificaremos a git rebase que ejecute un proceso interactivo para hacer el rebase, abriendo así el editor que hayamos definido en la configuración de git.

Si cambiamos pick por reword en el rebase interactivo, podremos cambiar el mensaje del commit.

**Guardando cambios temporalmente**

Si no estamos preparados para realizar un commit pero queremos guardar nuestros cambios, git tiene una herramienta: **git stash**.

Imaginemos por un casual que estamos en la rama “responsive” y tenemos algunos cambios en nuestro fichero. En ese preciso momento nos piden ocuparnos de un bug y tenemos que cambiar a la rama “hotfix”, pero si no hacemos un commit perderíamos el progreso de la rama “responsive”. Queremos guardar el progreso en esa rama pero no queremos hacer un commit todavía. Muy bien, con **git stash** podemos guardar temporalmente todos esos cambios y cambiar de rama sin perder el progreso.

Luego, cuando volvamos a la rama responsive, después de ocuparnos de la otra tarea en la otra rama, podemos listar los cambios que tenemos en el stash con **git stash list**, y recuperar el progreso guardado anteriormente.

Para recuperar y aplicar esos cambios vamos a dar uso al comando **git stash apply [stash id]**. Ej: **git stash apply stash@{3}**. (Si solo utilizamos el comando **git stash apply**, sin añadir el id, nos recuperará el último cambio guardado en stash, o sea el id 0).

Por cierto, al aplicar un stash y recuperar los cambios, ese mismo stash no se va a eliminar del stash list. Para eliminar algún stash realizado anteriormente podemos utilizar el comando **git stash drop [stash id]**. Ej: **git stash drop stash@{1}**.

Los archivos que no tengan modificaciones no podremos guardarlos en el stash a no ser que los movamos al Staging Area. Aun así, git stash tiene un flag que nos permite añadir archivos no trackeados (Working Directory): **git stash -u**.

PD: ¡Cuidado! Stash guarda el último cambio en stash en el id 0 [stash@{0}], no en el número entero más grande. Por lo tanto el último número de id será el primer cambio guardado en el stash.

**Eligiendo commits selectivamente**

Si estás trabajando en una rama, pero de repente te piden hacer un hotfix de un bug y en vez de hacer el cambio lo haces en la rama de la funcionalidad en la que estás trabajando ahora mismo. En vez de hacerlo en la rama del hotfix, deduces que vas a hacer el merge de esa funcionalidad pronto y no ves necesario solucionarlo en otra rama, pero, por lo visto vas a tardar más de lo que esperabas en desarrollar esa funcionalidad y el hotfix es muy urgente. En esas situaciones, podemos utilizar **cherry pick**; nos permitirá sacar cambios específicos de una rama y mezclarlos en otra.

Lo único que necesitaremos es tener el SHA-1 del commit que queremos cambiar a otra rama, crear o movernos a la otra rama y utilizar el comando: **git cherry-pick [sha-1]**. Con esto ya tendremos en la otra rama de master el contenido de ese commit que inicialmente debería haber ido en esta misma rama. Luego ya podemos hacer el merge sin ningún problema y tener el hotfix solucionado en master.