Git & GitHub pt. 3

Autor: José Guillermo Sandoval Huerta

Fecha: 31 octubre 2023

Índice

Comandos avanzados

Los comandos avanzados de Git son herramientas que permiten a los usuarios de Git realizar tareas más complejas y personalizadas en su flujo de trabajo de control de versiones. Estos comandos son útiles para tener mayor flexibilidad y control sobre el repositorio. A continuación, se presentan algunos de estos comandos.

Stash

Es una funcionalidad que permite guardar temporalmente cambios sin comprometerlos en una confirmación (commit) ni perderlos. El stash se puede entender como un área de almacenamiento temporal donde se puede guardar cambios en el directorio de trabajo que no estás listo para comprometer en un commit, pero que tampoco se desean perder. Sin embargo, es importante mencionar que el stash no es adecuado para almacenar cambios durante un largo período, ya que está diseñado para cambios temporales y transiciones rápidas entre tareas [1].

El stash es útil en situaciones en las que se está trabajando en una rama y es necesario cambiar de rama rápidamente sin confirmar los cambios en la rama actual. Para ello se ocupa el siguiene comando:

- **git stash save "<Mensaje descriptivo (opcional)>"**

Entre otros comandos de su funcionamiento tenemos los siguientes:

- **git stash pop** - recuperar cambios del stash.
- **git stash apply** - aplicar los cambios del stash sin eliminarlos del stash.
- **git stash list** - listar los cambios en el stash.
- **git stash drop stash@<n>** - eliminar cambios del stash. Donde <n> es el índice del stash que deseas eliminar.

Clean

El comando **git clean** en Git se utiliza para eliminar archivos no rastreados y directorios que no están bajo control de versiones [2]. Es útil cuando se desea eliminar archivos generados automáticamente, archivos temporales, o cualquier otro tipo de archivo no deseado que no debería formar parte de tu repositorio Git. El comando git clean es útil para mantener el directorio de trabajo limpio y eliminar elementos que no deben estar allí. Este comando emplea algunas opciones de filtrado como son:

- **git clean -n** - o **--dry-run**, muestra una lista de archivos que se eliminarán sin realmente eliminarlos. Es útil para ver qué archivos se eliminarán antes de ejecutar el comando.
- **git clean -f** - o **--force**, ejecuta el comando en modo forzado, lo que significa que los archivos se eliminarán sin necesidad de confirmación adicional.
- **git clean -i** - o **--interactive**, permite confirmar o rechazar cada archivo antes de eliminarlo.

Cabe mencionar que git clean es irreversible, por lo que es importante tener precaución al usarlo y asegurarse de que los archivos que se eliminarán no son necesarios en tu proyecto.

Cherry-pick

El comando **git cherry-pick** se utiliza en Git para aplicar commits específicos de una rama a otra. Esta operación permite seleccionar uno o varios commits individuales y copiarlos en una rama diferente. El cherry-pick toma los cambios introducidos por un commit y los aplica en la rama actual. Esto es útil cuando se desea traer cambios específicos de una rama a otra sin tener que fusionar la rama completa [3]. Para ejecutar este comando es por medio de la siguiente sentencia:

- **git cherry-pick <commit-hash>** - Siendo <commit-hash> el hash SHA-1 del commit que se desea aplicar.

El uso de este comando puede causar los siguientes procesos:

- **Conflictos**: Pueden ocurrir conflictos si los cambios del commit que estás aplicando entran en conflicto con los cambios existentes en la rama actual. Se deberán de resolver manualmente, similar a como se hace en una fusión (merge).

- **Orden de aplicación**: Los commits se aplican en el orden en que se especifiquen en el comando. Se puede aplicar varios commits en secuencia proporcionando múltiples hash SHA-1 en el comando.

- **Historia de commits**: Los commits copiados tendrán nuevos hash SHA-1 en la rama de destino, ya que se consideran commits nuevos con la misma funcionalidad.

Bisect

El comando **git bisect** se usa para encontrar un commit específico en la historia de un proyecto que introdujo un problema o un error. Se usa comúnmente cuando se está tratando de identificar exactamente cuál commit causó un fallo en el código. El comando git bisect realiza una búsqueda binaria (bisecting) para encontrar el commit problemático. Permitiendo ahorrar mucho tiempo y esfuerzo al no tener que revisar manualmente commits uno por uno en busca del culpable [4].

El proceso general de uso de git bisect implica los siguiente:

(1) Inicio del proceso: Inicialmente, se deberá decir a Git cuál fue el último commit en el que el proyecto funcionaba correctamente (un "commit bueno") y cuál fue el primer commit en el que se encontró el problema (un "commit malo"). Esto se hace usando los siguientes comandos:

- **git bisect start**

- **git bisect good <commit-bueno>**

- **git bisect bad <commit-malo>**

(2) Búsqueda binaria: Git comenzará a realizar una búsqueda binaria, seleccionando un commit intermedio entre el commit bueno y el commit malo. Luego, se deberá probar el proyecto para verificar si el error está presente en ese commit. Si el error está presente, se debe marcar el commit como "malo"; si no, como "bueno" usando:

- **git bisect good**
- **git bisect bad**

(3) Repetición del proceso: Git repetirá este proceso, seleccionando un nuevo commit intermedio, hasta que se encuentre el commit problemático. Una vez que Git haya encontrado el commit que causó el problema, proporcionará información sobre ese commit.

(4) Finalización del proceso: Para finalizar el proceso, se ejecuta **git bisect reset** para volver al estado normal de trabajo

Filter-Branch

El comando **git filter-branch** permite reescribir la historia de un repositorio, lo que implica aplicar filtros y transformaciones a los commits y su contenido [5]. Esta herramienta se utiliza en situaciones específicas en las que se necesita reestructurar o limpiar la historia de un repositorio. Es importante tener en cuenta que git filter-branch es una operación de reescritura de la historia y, por lo tanto, puede ser peligrosa si se utiliza incorrectamente. Además, una vez que se haya ejecutado git filter-branch, los commits reescritos tendrán nuevos hash SHA-1, lo que afectará a cualquier clon del repositorio existente y requerirá que todos los colaboradores actualicen sus copias locales del repositorio

Entre las principales funcionalidades están;

- **Eliminar archivos o carpetas específicos de todos los commits**: Permitiendo eliminar archivos o carpetas que se agregaron accidentalmente o que ya no deberían estar en el historial.
- **Cambiar la dirección de correo electrónico o nombre del autor**: Permitiendo cambiar la información del autor de los commits en caso de que se haya ingresado incorrectamente o se deba anonimizar.
- **Eliminar commits específicos**: Permitiendo eliminar commits enteros de la historia si, por ejemplo, contienen información sensible que no debe estar en el repositorio.
- **Fusionar múltiples commits en uno**: Permitiendo combinar varios commits en un solo commit para simplificar la historia.
- **Realizar conversiones de formato o renombrar archivos masivamente**: Permitiendo realizar tareas de transformación en los commits, como cambiar el formato de los archivos o cambiar nombres en toda la historia del proyecto.

En la mayoría de los casos, es preferible utilizar otras técnicas de Git, como git reset, git rebase o git commit --amend, para realizar cambios más simples en la historia del repositorio. git filter-branch se reserva para situaciones en las que se necesita realizar cambios más complejos y profundos en la historia del proyecto.

Reflog

Reflog es un registro que almacena un historial detallado de las referencias (ramas y etiquetas) en el repositorio. Contiene información sobre los cambios en las referencias, como los commits a los que apuntan, y permite recuperar información que de otro modo podría perderse. La palabra "reflog" es una abreviatura de "reference log" o "registro de referencias". El reflog es especialmente útil en situaciones en las que se ha perdido accidentalmente una referencia, como una rama o una etiqueta, o cuando se han realizado cambios que deseas deshacer [6]. Para ejecutar este proceso es por medio del siguiente comando:

- **git reflog**

Este comando mostrará una lista de eventos que involucran referencias en el repositorio, incluyendo información sobre los commits a los que apuntaban en cada evento. Es importante tener en cuenta que el reflog solo almacena un historial limitado de eventos y referencias. Con el tiempo, los eventos antiguos pueden ser eliminados del reflog para evitar que crezca demasiado. Por lo tanto, es útil utilizar el reflog con prontitud cuando se necesita recuperar información perdida o deshacer cambios.

Referencias

[1] J. Carrillo, «Git Stash Explicado: Cómo Almacenar Temporalmente los Cambios Locales en Git,» freecodecamp, 6 Febrero 2021. [En línea]. Available: https://www.freecodecamp.org/espanol/news/git-stash-explicado/. [Último acceso: 31 Octubre 2023].

[2] «Git Clean: limpiar tu proyecto de archivos no deseados,» Platzi, [En línea]. Available: https://platzi.com/clases/1557-git-github/19983-git-clean-limpiar-tu-proyecto-de-archivos-no-desea/. [Último acceso: 31 Octubre 2023].

[3] L. Burgos, «¿Cómo usar git cherry-pick sin morir en el intento?,» Medium, 13 Octubre 2020. [En línea]. Available: https://luisburgosv.medium.com/c%C3%B3mo-usar-git-cherry-pick-sin-morir-en-el-intento-fab92ba1ee7b. [Último acceso: 31 Octubre 2023].

[4] J. Holcombe, «Git Avanzado: Comandos Avanzados Además de los Básicos,» Kinsta, 21 Agosto 2023. [En línea]. Available: https://kinsta.com/es/blog/git-avanzado/. [Último acceso: 31 Octubre 2023].

[5] «Git Reset y Reflog: úsese en caso de emergencia,» Platzi, [En línea]. Available: https://platzi.com/clases/1557-git-github/19988-git-reset-y-reflog-usese-en-caso-de-emergencia/. [Último acceso: 31 Octubre 2023].

[6] Sebastian, «Rewriting history with git filter-branch,» sebastian-feldmann, 21 Noviembre 2019. [En línea]. Available: https://sebastian-feldmann.info/rewriting-history-with-git-filter-branch/. [Último acceso: 31 Octubre 2023].