¿Por qué usar un sistema de control de versiones como Git? 1/43

RECURSOS MARCADORES

Un sistema de control de versiones como Git nos ayuda a guardar el historial de cambios y crecimiento de los archivos de nuestro proyecto.

En realidad, los cambios y diferencias entre las versiones de nuestros proyectos pueden tener similitudes, algunas veces los cambios pueden ser solo una palabra o una parte específica de un archivo específico. Git está optimizado para guardar todos estos cambios de forma atómica e incremental, o sea, aplicando cambios sobre los últimos cambios, estos sobre los cambios anteriores y así hasta el inicio de nuestro proyecto.

- El comando para iniciar nuestro repositorio, o sea, indicarle a Git que queremos usar su sistema de control de versiones en nuestro proyecto, es `git init` .
- El comando para que nuestro repositorio sepa de la existencia de un archivo o sus últimos cambios es `git add` . Este comando no almacena las actualizaciones de forma definitiva, únicamente las guarda en algo que conocemos como “Staging Area” (área de montaje o ensayo).
- El comando para almacenar definitivamente todos los cambios que por ahora viven en el staging area es `git commit` . También podemos guardar un mensaje para recordar muy bien qué cambios hicimos en este commit con el argumento `-m "Mensaje del commit"` .
- Por último, si queremos mandar nuestros commits a un servidor remoto, un lugar donde todos podamos conectar nuestros proyectos, usamos el comando `git push` .

Comandos básicos de git

- git init: inicializa un repositorio de GIT en la carpeta donde se ejecute el comando.
- git add: añade los archivos especificados al área de preparación (staging).
- git commit -m “commit description”: confirma los archivos que se encuentran en el área de preparación y los agrega al repositorio.
- git commit -am “commit description”: añade al staging area y hace un commit mediante un solo comando. (No funciona con archivos nuevos)
- git status: ofrece una descripción del estado de los archivos (untracked, ready to commit, nothing to commit).
- git rm (. -r, filename) (–cached): remueve los archivos del index.
- git config --global user.email tu@email.com: configura un email.
- git config --global user.name <Nombre como se verá en los commits>: configura un nombre.

- git config --list: lista las configuraciones.

Analizar cambios en los archivos de un proyecto Git

- git log: lista de manera descendente los commits realizados.
- git log --stat: además de listar los commits, muestra la cantidad de bytes añadidos y eliminados en cada uno de los archivos modificados.
- git log --all --graph --decorate --oneline: muestra de manera comprimida toda la historia del repositorio de manera gráfica y embellecida.
- git show filename: permite ver la historia de los cambios en un archivo.
- git diff <commit1> <commit2>: compara diferencias entre en cambios confirmados.

Volver en el tiempo con branches y checkout

- git reset <commit> --soft/hard: regresa al commit especificado, eliminando todos los cambios que se hicieron después de ese commit.
- git checkout <commit/branch> <filename>: permite regresar al estado en el cual se realizó un commit o branch especificado, pero no elimina lo que está en el staging area.
- git checkout – <filePath>: deshacer cambios en un archivo en estado modified (que ni fue agregado a staging)

git rm y git reset

git rm:

Este comando nos ayuda a eliminar archivos de Git sin eliminar su historial del sistema de versiones. Esto quiere decir que si necesitamos recuperar el archivo solo debemos “viajar en el tiempo” y recuperar el último commit antes de borrar el archivo en cuestión.

git rm no puede usarse por sí solo, así nomás. Se debe utilizar uno de los flags para indicar a Git cómo eliminar los archivos que ya no se necesitan en la última versión del proyecto:

- git rm --cached <archivo/s>: elimina los archivos del área de Staging y del próximo commit, pero los mantiene en nuestro disco duro.
- git rm --force <archivo/s>: elimina los archivos de Git y del disco duro. Git siempre guarda todo, por lo que podemos acceder al registro de la existencia de los archivos, de modo que podremos recuperarlos si es necesario (pero debemos aplicar comandos más avanzados).

git reset

Con git reset volvemos al pasado sin la posibilidad de volver al futuro. Borramos la historia y la debemos sobreescribir.

- git reset --soft: Vuelve el branch al estado del commit especificado, manteniendo los archivos en el directorio de trabajo y lo que haya en staging considerando todo como nuevos

cambios. Así podemos aplicar las últimas actualizaciones a un nuevo commit.

- git reset --hard: Borra absolutamente todo. Toda la información de los commits y del área de staging se borra del historial.
- git reset HEAD: No borra los archivos ni sus modificaciones, solo los saca del área de staging, de forma que los últimos cambios de estos archivos no se envíen al último commit. Si se cambia de opinión se los puede incluir nuevamente con git add.

Ramas o Branches en git

Al crear una nueva rama se copia el último commit en esta nueva rama. Todos los cambios hechos en esta rama no se reflejarán en la rama master hasta que hagamos un merge.

- git branch <new branch>: crea una nueva rama.
- git checkout <branch name>: se mueve a la rama especificada.
- git merge <branch name>: fusiona la rama actual con la rama especificada y produce un nuevo commit de esta fusión.
- git branch: lista las ramas generadas.