Luis Gabriel Catalán Soto

999011084

¿Qué es GIT?

Git es un sistema de control de versiones de código abierto, desarrollado por Linus Torvalds en 2005. Se utiliza para realizar un seguimiento de los cambios en el código fuente durante el desarrollo de un proyecto de software.

Git permite a los desarrolladores trabajar en un proyecto de manera colaborativa y mantener un historial detallado de todos los cambios realizados en el código. Con Git, los desarrolladores pueden trabajar en diferentes versiones de un proyecto de manera simultánea, fusionar cambios de diferentes ramas y revertir cambios si es necesario.

Git es muy utilizado en la industria del software y es una herramienta fundamental para los desarrolladores que trabajan en proyectos en equipo. Git es compatible con una variedad de servicios de alojamiento de repositorios en línea, como GitHub, GitLab y Bitbucket, que permiten a los desarrolladores compartir y colaborar en sus proyectos.

Control de versiones con GIT

El control de versiones de Git es una herramienta que permite a los desarrolladores trabajar en equipo y realizar un seguimiento detallado de los cambios realizados en el código fuente de un proyecto, mediante los siguientes pasos:

- Crear un repositorio: se crea un repositorio de Git, que es el lugar donde se almacenará todo el código fuente del proyecto.
- Trabajar en el código: los desarrolladores trabajan en el código fuente del proyecto, realizando cambios y mejoras.
- Añadir cambios al área de preparación: los desarrolladores seleccionan los cambios que quieren incluir en la próxima versión del software y los añaden al área de preparación.
- Confirmar los cambios: una vez que los cambios están en el área de preparación, los desarrolladores pueden confirmarlos y guardarlos en el repositorio de Git. Cada confirmación se asocia con un mensaje descriptivo que explica los cambios realizados.
- Ramificar el código: si se desea trabajar en una característica o en una solución de problema específicos, se puede ramificar el código del repositorio en una nueva rama. De esta manera, los cambios realizados en la rama no afectan el código en la rama principal.
- Fusionar ramas: una vez que se han realizado los cambios en una rama, se pueden fusionar con la rama principal para integrarlos en el código principal.
- Gestionar conflictos: si se han realizado cambios en diferentes ramas en la misma sección del código, puede haber conflictos que deben ser resueltos antes de fusionar las ramas.
- Deshacer cambios: si se cometen errores o se necesitan revertir cambios previos, Git permite deshacer cambios en el código y volver a una versión anterior del proyecto.

Estados de un archivo en GIT

Los archivos pueden estar en diferentes estados según su estado de seguimiento y el estado de los cambios realizados en ellos. Los estados de un archivo en Git son los siguientes:

- Sin seguimiento (Untracked): un archivo se encuentra en este estado cuando se ha creado en el directorio de trabajo, pero aún no se ha agregado al repositorio de Git. Git no realiza un seguimiento de los cambios realizados en estos archivos.
- Modificado (Modified): un archivo se encuentra en este estado cuando se ha realizado algún cambio en el archivo después de su última confirmación. Git detecta automáticamente los cambios realizados en el archivo.
- Preparado (Staged): un archivo se encuentra en este estado cuando se ha agregado al área de preparación (también conocida como "index" o "staging area") para ser confirmado en la próxima versión del proyecto. Los cambios realizados en estos archivos serán incluidos en la próxima confirmación.
- Confirmado (Committed): un archivo se encuentra en este estado cuando los cambios realizados en el archivo se han confirmado en el repositorio de Git. Los cambios realizados en estos archivos son permanentes y forman parte del historial del proyecto.
- Ignorado (Ignored): los archivos se pueden especificar para que Git los ignore y no los incluya en el seguimiento de versiones. Estos archivos se encuentran en este estado.

Como se configura un repositorio

Para configurar un repositorio se deben seguir los siguientes pasos:

- Crear un directorio para el repositorio: en el sistema de archivos local, se debe crear un directorio que servirá como repositorio de Git. Se puede hacer desde la terminal utilizando el comando "mkdir" seguido del nombre del directorio.
- Inicializar el repositorio: una vez creado el directorio, se debe inicializar el repositorio de Git. Esto se realiza desde la terminal utilizando el comando "git init". Este comando creará un repositorio vacío en el directorio especificado y establecerá la estructura de archivos necesaria para el control de versiones.
- Añadir archivos al repositorio: para agregar archivos al repositorio, se utiliza el comando "git add". Este comando agrega los archivos especificados al área de preparación para ser confirmados en la próxima versión del proyecto. Por ejemplo, si se desea agregar un archivo llamado "index.html", se puede utilizar el comando "git add index.html".
- Confirmar los cambios: una vez que se han agregado los archivos al área de preparación, se pueden confirmar los cambios en el repositorio utilizando el comando "git commit". Este comando confirma los cambios realizados en los archivos y los guarda permanentemente en el historial de versiones del proyecto.
- Configurar el repositorio: se pueden establecer diferentes configuraciones en el repositorio de Git, como el nombre y correo electrónico del autor de los cambios, la configuración de la rama por defecto, entre otros. Esto se realiza utilizando el comando

"git config". Por ejemplo, para establecer el nombre del autor, se puede utilizar el comando "git config --global user.name "Nombre del Autor"".

Comandos en GIT

A continuación, se describen algunos de los comandos más comunes en Git:

Configurar herramientas: Configura la información del usuario para todos los repositorios locales

- git config --global user.name "[name]" Establece el nombre que desea esté anexado a sus transacciones de commit
- git config --global user.email "[email address]" Establece el e-mail que desea esté anexado a sus transacciones de commit
- git config --global color.ui auto Habilita la útil colorización del producto de la línea de comando

Crear repositorios Inicia un nuevo repositorio u obtiene uno de una URL existente

- git init [project-name] Crea un nuevo repositorio local con el nombre especificado
- git clone [url] Descarga un proyecto y toda su historia de versión

Efectuar cambios Revisa las ediciones y elabora una transacción de commit

- git status Enumera todos los archivos nuevos o modificados que se deben confirmar
- git add [file] Toma una instantánea del archivo para preparar la versión
- git reset [file] Mueve el archivo del área de espera, pero preserva su contenido
- git diff Muestra las diferencias de archivos que no se han enviado aún al área de espera
- git diff --staged Muestra las diferencias del archivo entre el área de espera y la última versión del archivo
- git commit -m "[descriptive message]" Registra las instantáneas del archivo permanentemente en el historial de versión

Cambios grupales: Nombra una serie de commits y combina esfuerzos ya culminados

- git branch Enumera todas las ramas en el repositorio actual
- git branch [branch-name] Crea una nueva rama
- git checkout [branch-name] Cambia a la rama especificada y actualiza el directorio activo
- git merge [branch] Combina el historial de la rama especificada con la rama actual
- git branch -d [branch-name] Borra la rama especificada

Nombres del archivo de refactorización: Reubica y retira los archivos con versión

- git rm --cached [file] Retira el archivo del control de versiones, pero preserva el archivo a nivel local
- git rm [file] Borra el archivo del directorio activo y pone en el área de espera el archivo borrado
- git mv [file-original] [file-renamed] Cambia el nombre del archivo y lo prepara para commit

Suprimir tracking Excluye los archivos temporales y las rutas

- git ls-files --other --ignored --exclude-standard Enumera todos los archivos ignorados en este proyecto

Guardar fragmentos Almacena y restaura cambios incompletos

- git stash Almacena temporalmente todos los archivos tracked modificados
- git stash list Enumera todos los sets de cambios en guardado rápido
- git stash pop Restaura los archivos guardados más recientemente
- git stash drop Elimina el set de cambios en guardado rápido más reciente

Repasar historial: Navega e inspecciona la evolución de los archivos de proyecto

- git log Enumera el historial de la versión para la rama actual
- git log --follow [file] Enumera el historial de versión para el archivo, incluidos los cambios de nombre
- git diff [first-branch]...[second-branch] Muestra las diferencias de contenido entre dos ramas
- git show [commit] Produce metadatos y cambios de contenido del commit especificado

Rehacer commits Borra errores y elabora historial de reemplazo

- git reset [commit] Deshace todos los commits después de [commit], preservando los cambios localmente
- git reset --hard [commit] Desecha todo el historial y regresa al commit especificado

Sincronizar cambios Registrar un marcador de repositorio e intercambiar historial de versión

- git fetch [bookmark] Descarga todo el historial del marcador del repositorio
- git merge [bookmark]/[branch] Combina la rama del marcador con la rama local actual
- git push [alias] [branch] Carga todos los commits de la rama local al GitHub
- git pull Descarga el historial del marcador e incorpora cambios

Bibliografía

Bustamante, L. (2021). Control de versiones con Git. https://www.academia.edu/49331744/Control_de_versiones_con_Git

Git. (2023). Reference Manual. https://git-scm.com/doc

Github Training. (2022). Hoja de referencia. https://training.github.com/downloads/es_ES/github-git-cheat-sheet.pdf

Gómez, S. (2015). Introducción a Git y Github. https://www.uco.es/aulasoftwarelibre/wp-content/uploads/2015/11/git-cosfera-dia-1.pdf