Introducción

Git es un sistema de control de versiones distribuido, diseñado para gestionar proyectos de cualquier tamaño con rapidez y eficiencia. Se ha convertido en una herramienta fundamental para el desarrollo de software en todo tipo de proyectos.

Un sistema de control de versiones es una herramienta que permite gestionar cambios en el código fuente u otros archivos de un proyecto a lo largo del tiempo. Esto incluye la capacidad de rastrear cambios, revertir a versiones anteriores, y colaborar de manera efectiva en equipo.

Características principales de Git:

- Distribuido: Cada usuario tiene una copia completa del repositorio, lo que permite trabajar de forma independiente sin necesidad de una conexión constante a un servidor central.
- Rápido y eficiente: Git está optimizado para manejar grandes proyectos con rapidez, permitiendo realizar operaciones como la creación de ramas o fusiones en cuestión de segundos.
- Ramificación y fusión: Git facilita la creación de ramas para trabajar en paralelo en diferentes características o correcciones de errores, y luego fusionar esos cambios de manera limpia y sin conflictos.
- Seguimiento de cambios: Git registra cada cambio realizado en el repositorio, lo que permite ver quién hizo cada modificación, cuándo se realizó y qué se modificó.
- Colaboración: Git facilita la colaboración entre desarrolladores al permitir compartir cambios a través de repositorios remotos y gestionar conflictos de manera eficiente.

Comandos básicos

- git init: Inicia un nuevo repositorio Git en el directorio actual.
- git clone [url]: Clona un repositorio Git existente desde una URL remota a tu máquina local.
- git add [archivo(s)]: Agrega cambios al área de preparación para ser incluidos en el próximo commit. Puedes especificar archivos específicos o usar "." para agregar todos los cambios.
- git commit -m "[mensaje]": Crea un nuevo commit con los cambios que están en el área de preparación. El mensaje describe los cambios realizados en este commit.
- git status: Muestra el estado actual del repositorio, incluyendo los cambios sin agregar, los archivos en el área de preparación y los commits pendientes.
- git log: Muestra el historial de commits del repositorio, incluyendo el autor, la fecha y el mensaje de cada commit.
- git branch: Muestra una lista de ramas en el repositorio. La rama actual se resalta.
- git branch [nombre]: Crea una nueva rama con el nombre especificado.
- git checkout [nombre de la rama o commit]: Cambia a una rama diferente o restaura el repositorio al estado de un commit específico.
- git merge [rama]: Fusiona los cambios de una rama en la rama actual.
- git pull: Obtiene los cambios del repositorio remoto y los fusiona con el repositorio local.
- git push: Envía los commits locales al repositorio remoto.
- git remote -v: Muestra las URL de los repositorios remotos vinculados.
- git remote add [nombre] [url]: Agrega un nuevo repositorio remoto con el nombre especificado y la URL proporcionada.

GitHub

Trabajar con repositorios en GitHub implica interactuar con un servicio remoto basado en la web que utiliza Git como su sistema de control de versiones.

- Ramificaciones (Branches):
    - Las ramas en Git permiten trabajar en diferentes líneas de desarrollo de forma independiente. Esto es útil para desarrollar nuevas características, corregir errores o experimentar sin afectar la rama principal del proyecto. Cada rama tiene su propio conjunto de commits que representan cambios específicos realizados en esa línea de desarrollo.
- Fusión (Merge):
    - La fusión es el proceso de combinar los cambios de una rama en otra. Git intenta fusionar los cambios de forma automática, pero pueden surgir conflictos si los cambios son incompatibles entre sí. Cuando se fusionan ramas, Git analiza los cambios realizados en cada rama y las combina de manera inteligente. Si los cambios no se solapan, la fusión se realiza automáticamente. Sin embargo, si Git detecta que los cambios en ambas ramas afectan las mismas líneas de código, se produce un conflicto de fusión.
- Conflictos de Fusión:
    - Un conflicto de fusión ocurre cuando Git no puede fusionar los cambios automáticamente debido a diferencias entre las versiones de los archivos en las ramas que se están fusionando. Git marca los conflictos en los archivos afectados, indicando las secciones del código que causan conflicto.
- Resolución de Conflictos:
    - Los conflictos de fusión deben resolverse manualmente por el usuario. Esto implica editar los archivos afectados para elegir qué cambios mantener y cómo combinarlos. Después de resolver los conflictos, los archivos modificados deben agregarse al área de preparación y se completa la fusión con un commit que incluya los cambios resueltos. Una vez resueltos los conflictos y completada la fusión, la rama resultante contiene los cambios combinados de las ramas originales.
- Evitar Conflictos:
    - Para minimizar la posibilidad de conflictos, es útil mantener las ramas actualizadas con la rama principal y comunicarse con otros colaboradores para coordinar los cambios. Utilizar ramas cortas y fusionar con frecuencia puede reducir la probabilidad de conflictos graves. La gestión efectiva de ramas, fusiones y conflictos es esencial para un flujo de trabajo colaborativo y ordenado en Git. Responder adecuadamente a los conflictos garantiza que los cambios se fusionen de manera segura y que se mantenga la integridad del código del proyecto.

Comandos

- Crear un repositorio en GitHub:
    - Ve a la página principal de GitHub y haz clic en el botón "New" para crear un nuevo repositorio.
    - Completa la información requerida, como el nombre del repositorio, la descripción y la visibilidad (público o privado).
    - Opcionalmente, puedes inicializar el repositorio con un archivo README.md, un archivo .gitignore y/o una licencia.
- Clonar un repositorio de GitHub:
        - git clone <URL_del_repositorio>
    - Esta acción descarga una copia del repositorio desde GitHub a tu máquina local.
- Agregar cambios y hacer commits:
    - Realiza tus cambios en los archivos locales.
    - Utiliza git add para agregar los cambios al área de preparación.
    - Utiliza git commit para confirmar los cambios y crear un nuevo commit.
        - git add .

        - git commit -m "Mensaje del commit"
- Subir cambios a GitHub:
        - git push origin <nombre_de_rama>
    - Esto envía tus commits locales a GitHub, específicamente a la rama especificada.
- Obtener cambios desde GitHub:
        - git pull origin <nombre_de_rama>
    - Esto obtiene los últimos cambios del repositorio remoto en GitHub y los fusiona con tu rama local.
- Crear y fusionar ramas:
    - Para crear una nueva rama:
        - git branch <nombre_de_rama>
    - Para cambiar a una rama existente:
        - git checkout <nombre_de_rama>
    - Para fusionar una rama en la rama actual:
        - git merge <nombre_de_rama>
- Configurar repositorios remotos:
    - Para ver los repositorios remotos asociados con tu repositorio local:
        - git remote -v
    - Para agregar un nuevo repositorio remoto:
        - git remote add <nombre_remoto> <URL_del_repositorio>
- Actualizar y eliminar repositorios remotos:
    - Para actualizar la URL de un repositorio remoto:
        - git remote set-url <nombre_remoto> <nueva_URL>
    - Para eliminar un repositorio remoto:
        - git remote remove <nombre_remoto>

Conceptos y comandos avanzados

- Pull Request (Solicitud de extracción):
    - Una solicitud de extracción es una forma de proponer cambios en un repositorio y solicitar que sean fusionados con otra rama. Se utiliza comúnmente en entornos de desarrollo colaborativo, donde los colaboradores pueden revisar los cambios propuestos y dejar comentarios antes de fusionarlos. Los pull requests permiten una revisión y discusión transparente de los cambios antes de que se incorporen al proyecto principal.
        - hub pull-request -b <rama_base> -h <rama_origen> -m "Mensaje de la pull request"
- Fork:
    - Un fork es una copia independiente de un repositorio en GitHub, creada por otro usuario. Permite trabajar en cambios sin afectar el repositorio original. Los cambios realizados en un fork pueden ser propuestos mediante una solicitud de extracción. Los forks son comunes en proyectos de código abierto, donde cualquiera puede contribuir con cambios sin necesidad de permisos especiales.
- Rebase:
    - Rebase es una operación que mueve o reescribe la historia de una rama sobre otra. Se utiliza para integrar cambios de una rama en otra de manera más limpia y ordenada que una fusión estándar. Puede ser útil para mantener la historia del repositorio más lineal y fácil de entender, pero debe usarse con precaución para evitar problemas de historial.

- Stash:
    - Stash es una función que permite guardar temporalmente cambios no comprometidos en una pila de cambios. Se utiliza cuando se necesita cambiar de rama o aplicar cambios en otro lugar sin realizar un commit incompleto. Los cambios almacenados en el stash pueden ser recuperados más tarde y aplicados en cualquier momento.
        - git stash save "Mensaje de descripción"
- Clean:
    - Clean es un comando que se utiliza para eliminar archivos no rastreados y no deseados del directorio de trabajo. Puede ser útil para limpiar el directorio de trabajo de archivos generados automáticamente o ignorados que ya no son necesarios.
        - git clean -i
- Cherry-pick:
    - Cherry-pick es una operación que permite seleccionar un commit específico de una rama y aplicarlo en otra. Se utiliza cuando se desea incorporar un único cambio o conjunto de cambios de una rama a otra sin fusionar toda la historia de la rama.
        - git cherry-pick <commit>