Que es Git?

esa una herramienta confiable que a ayuda a trabajar de manera colaborativa en proyectos de desarrollo de software. Permite realizar cambios de manera segura, trabajar en ramas separadas y fusionar las contribuciones de manera ordenada.

Git permite realizar cambios a un proyecto y estos cambios no afecten el trabajo de los demás. Git permite crear una "rama" separada donde se pueden hacer cambios sin interferir con el trabajo principal.

Cuando se termine de hacer cambios a los proyectos GIT permite unificar estos cambios al proyecto principal, lo que significa que permite que las contribuciones se combinen a los demás cambios hechos por otros colaboradores.

Control de versiones con GIT

Es una herramienta que permite administrar y verificar los cambios realizados. Es una forma de mantener un historial detallado de las modificaciones realizadas.

En GIT cada cambio realizado se guarda en “commit” que es indicador de cambios realizados. Estos commits se almacenan históricamente lo que permite regresar a versiones previas del proyecto

El control de versiones en Git ofrece varias ventajas importantes:

Registro completo, Colaboración eficiente, Ramificación y fusión, Seguimiento de cambios

Estado de un archivo en GIT

En Git, los archivos los siguientes estados:

Untracked (No rastreado): Cuando un archivo está en estado "No rastreado", significa que Git no está siguiendo los cambios que se le realizan

Modified (Modificado): Cuando un archivo se encuentra en estado "Modificado", significa que ha sufrido cambios desde el último commit

Committed (Confirmado): Cuando un archivo está en estado "Confirmado", significa que los cambios realizados en ese archivo se han registrado en el repositorio Git.

Staged (En preparación): en Git se refiere a la etapa intermedia donde los cambios realizados en un archivo se han seleccionado para ser incluidos en el próximo commit

Ignored (Ignorado): en Git se refiere a los archivos o patrones de archivos especificados en el archivo ".gitignore", que se excluyen del seguimiento de versiones. Git no rastrea los cambios en estos archivos ni los incluye en los commits, lo que ayuda a mantener un repositorio limpio y evitar la inclusión de archivos innecesarios o sensibles.

Como se configura un Repositorio

1. **Inicializar un repositorio**: En el directorio principal de tu proyecto en el sistema de archivos. Luego, en la línea de comandos, ejecuta el comando git init. Esto creará un repositorio vacío en ese directorio.

2. **Agregar archivos**: Coloca los archivos y carpetas de tu proyecto dentro del directorio raíz del repositorio. Puedes utilizar el comando git add seguido de los nombres de los archivos o utilizar git add . para agregar todos los archivos y carpetas del directorio actual al área de preparación.

3. **Realizar un commit inicial:** Una vez que los archivos están en el área de preparación, puedes hacer un commit para confirmar los cambios. Usa el comando git commit -m "Mensaje del commit" para crear un commit con un mensaje que describa los cambios realizados.

4. **Configurar el nombre y correo electrónico del autor**: Es recomendable configurar tu nombre y correo electrónico como autor de los commits. Puedes usar los comandos git config --global user.name "Tu nombre" y git config --global user.email "tu@email.com" para establecer esta información.

Comandos en GIT

Aquí tienes una lista detallada de los comandos de Git y una descripción de lo que hace cada uno de ellos:

1. **“git init**”: Inicializa un nuevo repositorio Git vacío en un directorio local.

2. “git clone [url]”: Clona un repositorio Git remoto existente en un nuevo directorio local.

3. “git add [archivo]”: Agrega un archivo específico al área de preparación (staging area) para ser incluido en el próximo commit. También se puede utilizar “git add .” para agregar todos los archivos modificados al área de preparación.

4. “git commit -m "mensaje"“: Crea un nuevo commit con los cambios realizados en los archivos en el área de preparación. El mensaje proporciona una descripción breve de los cambios realizados.

5. “git status”: Muestra el estado actual del repositorio, incluyendo los archivos modificados, agregados y eliminados.

6. “git diff”: Muestra las diferencias entre los cambios realizados y el último commit.

7. “git log”: Muestra el historial de commits realizados en el repositorio, incluyendo información como el autor, la fecha y el mensaje del commit.

8. “git branch”: Lista todas las ramas (branches) en el repositorio. También se puede utilizar “git branch [nombre]” para crear una nueva rama o “git branch -d [nombre]” para eliminar una rama.

9. “git checkout [rama]”: Cambia a la rama especificada, actualizando el directorio de trabajo con los archivos de esa rama.

10. “git merge [rama]”: Combina los cambios de una rama con la rama actual.

11. “git pull”: Obtiene los últimos cambios de un repositorio remoto y los fusiona con la rama actual.

12. “git push”: Envía los commits locales a un repositorio remoto.

13. “git remote”: Muestra la lista de repositorios remotos vinculados al repositorio local. También se puede utilizar “git remote add [nombre] [url]” para agregar un nuevo repositorio remoto.

14. “git fetch”: Descarga los últimos cambios de un repositorio remoto, pero no los fusiona con la rama actual.

15. “git revert [commit]”: Crea un nuevo commit que deshace los cambios realizados en un commit específico.

16. “git reset [commit]”: Restablece el estado del repositorio a un commit específico, eliminando los commits posteriores.

17. “git tag”: Administra etiquetas (tags) para marcar puntos específicos en la historia del repositorio. Se puede utilizar “git tag [nombre]” para crear una etiqueta y “git tag -l” para listar las etiquetas existentes.

18. “git remote -v”: Muestra los detalles de los repositorios remotos vinculados, incluyendo las URL de lectura y escritura.

19. “git config”: Configura opciones y variables de Git. Se puede utilizar “git config --global [opción] [valor]” para establecer opciones globales.

20. “git stash”: Guarda los cambios locales en una pila temporal y limpia el directorio de trabajo. Puede utilizarse para cambiar temporalmente de rama sin confirmar los cambios.

21. “git fetch origin”: Descarga los últimos cambios del repositorio remoto llamado "origin" sin fusionarlos con la rama actual.