1. **GIT**

   Es un proyecto de código abierto con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005. Un asombroso número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Los desarrolladores que han trabajado con Git cuentan con una buena representación en la base de talentos disponibles para el desarrollo de software, y este sistema funciona a la perfección en una amplia variedad de sistemas operativos e IDE (entornos de desarrollo integrados).

   Git, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés). En lugar de tener un único espacio para todo el historial de versiones del software, como sucede de manera habitual en los sistemas de control de versiones antaño populares, como CVS o subversión (también conocido como SVN), en Git, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.

   Además de contar con una arquitectura distribuida, Git se ha diseñado teniendo en cuenta el rendimiento, la seguridad y la flexibilidad.

2. **Control de las versiones con GIT**

   El control de versiones, también conocido como "control de código fuente", es la práctica de rastrear y gestionar los cambios en el código de software. Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo. A medida que los entornos de desarrollo se aceleran, los sistemas de control de versiones ayudan a los equipos de software a trabajar de forma más rápida e inteligente. Son especialmente útiles para los equipos de DevOps, ya que les ayudan a reducir el tiempo de desarrollo y a aumentar las implementaciones exitosas.

   El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir hacia atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error, al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

   El control de versiones ayuda a los equipos a resolver este tipo de problemas al realizar un seguimiento de todos los cambios individuales de cada

colaborador y al contribuir a evitar que el trabajo concurrente entre en conflicto. Los cambios realizados en una parte del software pueden ser incompatibles con los que ha hecho otro desarrollador que está trabajando al mismo tiempo. Este problema debería detectarse y solucionarse de manera ordenada sin bloquear el trabajo del resto del equipo. Además, en todo el desarrollo de software, cualquier cambio puede introducir nuevos errores por sí mismo y el nuevo software no es fiable hasta que se prueba. De este modo, las pruebas y el desarrollo van de la mano hasta que está lista una nueva versión.

3. **Estados de un archivo en GIT**

   En Git, un archivo puede tener diferentes estados según su interacción con el repositorio. Los estados principales de un archivo en Git son los siguientes:

   - Untracked (No rastreado): Un archivo no rastreado es aquel que Git no está supervisando ni controlando cambios. Esto significa que Git no está al tanto de los cambios realizados en el archivo.

   - Modified (Modificado): Un archivo modificado es aquel que ha sufrido cambios después de que Git comenzó a rastrearlo. Estos cambios pueden ser nuevos contenidos, eliminaciones o modificaciones en el archivo.

   - Staged (Preparado): Un archivo preparado es aquel que ha sido marcado para ser incluido en el próximo commit. Es decir, los cambios realizados en el archivo han sido seleccionados para formar parte del historial del repositorio en el siguiente commit.

   - Committed (Confirmado): Un archivo confirmado es aquel que ha sido registrado en el historial del repositorio mediante un commit. En esta etapa, los cambios realizados en el archivo están seguros en el repositorio y se pueden recuperar en cualquier momento.

   Además de estos estados principales, hay algunos estados adicionales que pueden ser relevantes:

   - Ignored (Ignorado): Un archivo ignorado es aquel que está excluido de Git y no se rastrea en absoluto. Los archivos ignorados no se consideran para cambios, commits o actualizaciones en el repositorio.

4. **Como configurar un repositorio**
   Para configurar un repositorio Git, debemos seguir estos pasos:

   1. Instalar Git: Si aún no se cuenta con Git instalado en tu sistema, descarga, se puede obtener desde el sitio web oficial de Git (https://git-scm.com). Sigue las instrucciones específicas para tu sistema operativo.

   2. Configuración del nombre y dirección de correo electrónico: Abrimos una terminal o línea de comandos y ejecuta los siguientes comandos, reemplazando "Tu Nombre" y "tu@email.com" con tu información:

      git config –global user.name “Tu Nombre”
      git config –global user.email tu@gmail.com

      Estos comandos permiten establecer tu nombre y correo electrónico como los valores de configuración globales para Git.

   3. Inicializar un repositorio Git: Vamos al directorio del proyecto en el que deseas crear el repositorio y ejecuta el siguiente comando:

      git init

   4. Agrega archivos al repositorio: Coloca los archivos que deseas incluir en el repositorio en el directorio del proyecto. Luego, ejecuta el siguiente comando para agregar los archivos al área de preparación (staging area):

      git add nombre_archivo

      Se puede reemplazar “nombre_archiv” con el nombre de un archivo específico o usar “.” Para agregar todos los archivos nuevos o modificados en el directorio actual.

   5. Realiza un commit: Después de agregar los archivos al área de preparación, ejecuta el siguiente comando para realizar un commit y guardar los cambios en el repositorio:

      git commit -m “Mensaje del commit”

      Se reemplaza “Mensaje de commit” por una descripción breve y significatia de los cambios que se desee.

5. **Comandos en GIT**
    - Git add: Mueve los cambios del directorio de trabajo de área del entrono de ensayo. Así puedes preparar una instantánea antes de confirmar en el historial oficial.
    - Rama de git: Este comando es tu herramienta de administración de ramas de uso general. Permite crear entornos de desarrollo aislados en un solo repositorio.
    - Git checkout: Además de extraer las confirmaciones y las revisiones de archivos antiguas, git checkout también sirve para navegar por las ramas existentes. Combinado con los comandos básicos de Git, es una forma de trabajar en una línea de desarrollo concreta.
    - Git clean: Elimina los archivos sin seguimiento de tu directorio de trabajo. Es la contraparte lógica de git reset, que normalmente solo funciona en archivos con seguimiento.
    - Git clone: Crea una copia de un repositorio de Git existente. La clonación es la forma más habitual de que los desarrolladores obtengan una copia de trabajo de un repositorio central.
    - Git commit: Confirma la instantánea preparada en el historial del proyecto. En combinación con git add, define el flujo de trabajo básico de todos los usuarios de Git.
    - Git commint –amend: Pasar la marca --amend a git commit permite modificar la confirmación más reciente. Es muy práctico si olvidas preparar un archivo u omites información importante en el mensaje de confirmación.
    - Git config: Este comando va bien para establecer las opciones de configuración para instalar Git. Normalmente, solo es necesario usarlo inmediatamente después de instalar Git en un nuevo equipo de desarrollo.
    - Git fetch: Con este comando, se descarga una rama de otro repositorio junto con todas sus confirmaciones y archivos asociados. Sin embargo, no intenta integrar nada en el repositorio local. Esto te permite inspeccionar los cambios antes de fusionarlos en tu proyecto.
    - Git init: Inicializa un nuevo repositorio de Git. Si quieres poner un proyecto bajo un control de revisiones, este es el primer comando que debes aprender.
    - Git log: Permite explorar las revisiones anteriores de un proyecto. Proporciona varias opciones de formato para mostrar las instantáneas confirmadas.
    - Git merge: Es una forma eficaz de integrar los cambios de ramas divergentes. Después de bifurcar el historial del proyecto con git branch, git merge permite unirlo de nuevo.

- Git pull: Este comando es la versión automatizada de git fetch. Descarga una rama de un repositorio remoto e inmediatamente la fusiona en la rama actual. Este es el equivalente en Git de svn update.
- Git push: Enviar (push) es lo opuesto a recuperar (fetch), con algunas salvedades. Permite mover una o varias ramas a otro repositorio, lo que es una buena forma de publicar contribuciones. Es como svn commit, pero envía una serie de confirmaciones en lugar de un solo conjunto de cambios.
- Git rebase: Un cambio de base con git rebase permite mover las ramas, lo que ayuda a evitar confirmaciones de fusión innecesarias. El historial lineal resultante suele ser mucho más fácil de entender y explorar.
- Git rebase –i: La marca -i se usa para iniciar una sesión de cambio de base interactivo. Esto ofrece todas las ventajas de un cambio de base normal, pero te da la oportunidad de añadir, editar o eliminar confirmaciones sobre la marcha.
- Git reflog: Git realiza el seguimiento de las actualizaciones en el extremo de las ramas mediante un mecanismo llamado registro de referencia o reflog. Esto permite volver a los conjuntos de cambios, aunque no se haga referencia a ellos en ninguna rama o etiqueta.
- Git remote: Es un comando útil para administrar conexiones remotas. En lugar de pasar la URL completa a los comandos fetch, pull y push, permite usar un atajo más significativo.
- Git reset: Deshace los cambios en los archivos del directorio de trabajo. El restablecimiento permite limpiar o eliminar por completo los cambios que no se han enviado a un repositorio público.
- Git revert: Permite deshacer una instantánea confirmada. Si descubres una confirmación errónea, revertirla es una forma fácil y segura de eliminarla por completo del código base.
- Git status: Muestra el estado del directorio en el que estás trabajando y la instantánea preparada. Lo mejor es que lo ejecutes junto con git add y git commit para ver exactamente qué se va a incluir en la próxima instantánea.