Avance proyecto 1

**¿Qué es Git?**

Git es un sistema de control de versiones distribuido, creado por Linus Torvalds en 2005, que permite a los desarrolladores gestionar y registrar los cambios en el código fuente de sus proyectos. Es ampliamente utilizado en el desarrollo de software, tanto en proyectos de código abierto como comerciales, y se ha convertido en el estándar de facto en la industria.

**¿Qué es Github?**

GitHub es una plataforma de desarrollo colaborativo que permite a los desarrolladores alojar y gestionar proyectos utilizando el sistema de control de versiones Git. Fundada en 2008 y adquirida por Microsoft en 2018, GitHub se ha convertido en una de las herramientas más populares para la colaboración en proyectos de software, albergando más de 100 millones de repositorios, muchos de los cuales son de código abierto.

**¿Qué son los repositorios locales en Git?**

Los repositorios locales en Git son contenedores de archivos que se almacenan en el equipo de un usuario individual. A diferencia de los repositorios remotos, que se alojan en servidores y son accesibles para múltiples usuarios, los repositorios locales permiten a un desarrollador trabajar de manera independiente en su propio entorno.

**¿Qué son los repositorios remotos en Git?**

Los repositorios remotos en Git son versiones de un proyecto que se almacenan en un servidor o en la nube, permitiendo a varios usuarios colaborar en el mismo código desde diferentes ubicaciones. Estos repositorios son fundamentales para el trabajo en equipo, ya que facilitan la sincronización de cambios y el acceso compartido al código.

**¿Cómo se crean los repositorios remotos en Github?**

Para crear repositorios remotos en GitHub, sigue estos pasos:

**Crear un Repositorio en GitHub:**

* Inicia sesión en tu cuenta de GitHub.
* Haz clic en el botón "New" o "Nuevo" para crear un nuevo repositorio.
* Asigna un nombre a tu repositorio y selecciona si deseas que sea público o privado. También puedes añadir un archivo README si lo deseas.
* Haz clic en "Create repository" para finalizar la creación.

**Obtener la URL del Repositorio:**

* Una vez creado el repositorio, GitHub te proporcionará una URL para acceder a él. Esta URL puede ser en formato HTTPS o SSH. Por ejemplo:
* HTTPS: https://github.com/usuario/nombre-repo.git
* SSH: git@github.com:usuario/nombre-repo.git

**Configurar el Repositorio Remoto en tu Repositorio Local:**

* Abre la terminal y navega hasta tu repositorio local (donde ya tienes tu código).
* Usa el siguiente comando para añadir la URL del repositorio remoto. Generalmente, se le asigna el nombre origin:

*git remote add origin <URL\_DEL\_REPOSITORIO>*

*git remote add origin https://github.com/usuario/nombre-repo.git*

**Subir Cambios al Repositorio Remoto:**

* Para enviar tus cambios al repositorio remoto por primera vez, utiliza el siguiente comando:

*git push -u origin master*

* Esto subirá la rama master y establecerá el seguimiento entre tu rama local y la remota.

**Sincronizar Cambios Posteriores:**

* Después de realizar nuevos commits, puedes usar simplemente:

*git push*

* Esto enviará los nuevos cambios al repositorio remoto sin necesidad de especificar la rama nuevamente, siempre que hayas configurado el seguimiento correctamente en el paso anterior.

¿Cómo se crean los repositorios locales en Git?

**Instalar Git**

Antes de comenzar, asegúrate de que Git esté instalado en tu sistema. Puedes verificarlo abriendo una terminal y ejecutando:

*git –version*

**Crear un Nuevo Directorio (Opcional)**

Si deseas crear un nuevo proyecto, primero crea un directorio para él. Puedes hacerlo con el siguiente comando:

*mkdir nombre-del-proyecto*

*cd nombre-del-proyecto*

**Inicializar el Repositorio**

Para convertir el directorio en un repositorio Git, utiliza el siguiente comando:

*git init*

Este comando crea un nuevo subdirectorio llamado .git, que contiene todos los archivos necesarios para el control de versiones. Ahora tu directorio es un repositorio Git vacío.

**Añadir Archivos al Repositorio**

Agrega los archivos que deseas gestionar en el repositorio. Puedes crear archivos manualmente o copiar archivos existentes en el directorio. Una vez que tengas archivos en el directorio, usa el siguiente comando para añadirlos al área de preparación (staging area):

*git add*

El punto (.) indica que deseas añadir todos los archivos en el directorio actual. También puedes añadir archivos específicos reemplazando el punto por el nombre del archivo.

**Realizar un Commit**

Después de añadir los archivos, realiza un commit para guardar los cambios en el historial del repositorio. Usa el siguiente comando:

*bash*

*git commit -m "Mensaje descriptivo del commit"*

Reemplaza "Mensaje descriptivo del commit" con un mensaje que explique los cambios realizados.

**Verificar el Estado del Repositorio**

Puedes verificar el estado de tu repositorio en cualquier momento utilizando:

*bash*

*git status*

Este comando te mostrará los archivos que están en el área de preparación, los que no están rastreados y otros detalles sobre el estado del repositorio.

¿Qué son las ramas (branches) en Git?

Las ramas (branches) en Git son una característica fundamental que permite a los desarrolladores trabajar en diferentes líneas de desarrollo dentro del mismo proyecto. Una rama es esencialmente un puntero que apunta a una de las confirmaciones (commits) en el historial del repositorio, permitiendo realizar cambios de manera aislada sin afectar la rama principal o cualquier otra rama.

Indique para que se utilizan cada uno de los siguientes comandos en Git:

**1. git clone**

Uso:

Este comando se utiliza para crear una copia local de un repositorio remoto.

Propósito:

* **Clonar Repositorios:** Permite a los desarrolladores obtener una copia completa de un repositorio, incluyendo todo su historial de commits, ramas y archivos. Es comúnmente utilizado para comenzar a trabajar en un proyecto existente que se encuentra en un servidor remoto como GitHub.

Ejemplo:

*bash*

*git clone https://github.com/usuario/nombre-repo.git*

**2. git checkout**

Uso:

Se utiliza para cambiar entre ramas o restaurar archivos en el directorio de trabajo.

Propósito:

* **Cambiar de Rama:** Permite al usuario cambiar a una rama diferente en el repositorio.
* **Restaurar Archivos:** También se puede usar para restaurar archivos a un estado específico en el historial del repositorio.

Ejemplo:

*bash*

*git checkout nombre-de-la-rama*

**3. git pull**

Uso:

Este comando se utiliza para actualizar la rama local con los cambios del repositorio remoto.

Propósito:

* **Sincronización:**git pull combina dos operaciones: git fetch (descargar los cambios del remoto) y git merge (fusionar esos cambios en la rama actual). Es útil para asegurarse de que tu copia local esté actualizada con los últimos cambios realizados por otros colaboradores.

Ejemplo:

*bash*

*git pull origin nombre-de-la-rama*

**4. git commit**

Uso:

Se utiliza para guardar los cambios en el repositorio local.

Propósito:

* **Registrar Cambios:** git commit toma los cambios que han sido añadidos al área de preparación (staging area) y los guarda en el historial del repositorio. Cada commit debe incluir un mensaje descriptivo que explique los cambios realizados.

Ejemplo:

*bash*

*git commit -m "Descripción de los cambios realizados"*

**5. git push**

Uso:

Este comando se utiliza para enviar los cambios locales a un repositorio remoto.

Propósito:

**Actualizar el Remoto:** git push permite a los desarrolladores subir sus commits locales a un repositorio remoto, haciendo que sus cambios estén disponibles para otros colaboradores. Es fundamental para compartir el trabajo en equipo.

Ejemplo:

*bash*

*git push origin nombre-de-la-rama*

**6. git stash**

Uso:

Se utiliza para guardar temporalmente los cambios no confirmados en el directorio de trabajo.

Propósito:

* **Guardar Cambios Temporales:** git stash permite a los desarrolladores guardar su trabajo en progreso sin hacer un commit. Esto es útil cuando necesitas cambiar de rama o realizar otras tareas sin perder los cambios actuales.

Ejemplo:

*bash*

*git stash*

Para recuperar los cambios guardados en el stash, puedes usar:

*bash*

*git stash pop*