# Instituto Tecnológico de las Américas

**Práctica:**

Tarea 3

**Integrantes:**


- Roylin Rodriguez Jimenez 2022-0455


**Período Académico:**

2023-C3

**Profesor:**

Kelyn Tejada

**Fecha de Entrega:**

25/11/2023

## 1-Que es Git?

Git es, con diferencia, el sistema de control de versiones moderno más utilizado del mundo. Git es un proyecto de código abierto maduro y 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 Subversion (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.

### Rendimiento

Las características básicas de rendimiento de Git son muy sólidas en comparación con muchas otras alternativas. La confirmación de nuevos cambios, la ramificación, la fusión y la comparación de versiones anteriores se han optimizado en favor del rendimiento.

### Seguridad

Git se ha diseñado con la principal prioridad de conservar la integridad del código fuente gestionado. El contenido de los archivos y las verdaderas relaciones entre estos y los directorios, las versiones, las etiquetas y las confirmaciones, todos ellos objetos del repositorio de Git, están protegidos con un algoritmo de hash criptográficamente seguro llamado "SHA1".

## Flexibilidad

Uno de los objetivos clave de Git en cuanto al diseño es la flexibilidad. Git es flexible en varios aspectos: en la capacidad para varios tipos de flujos de trabajo de desarrollo no lineal, en su eficiencia en proyectos tanto grandes como pequeños y en su compatibilidad con numerosos sistemas y protocolos.

## 2-Para que funciona el comando Git init?

El comando **git init** crea un nuevo repositorio de Git. Puede utilizarse para convertir un proyecto existente y sin versión en un repositorio de Git, o para inicializar un nuevo repositorio vacío. La mayoría de los demás comandos de Git no se encuentran disponibles fuera de un repositorio inicializado, por lo que este suele ser el primer comando que se ejecuta en un proyecto nuevo.

Al ejecutar git init, se crea un subdirectorio de .git en el directorio de trabajo actual, que contiene todos los metadatos de Git necesarios para el nuevo repositorio. Estos metadatos incluyen subdirectorios de objetos, referencias y archivos de plantilla. También se genera un archivo HEAD que apunta a la confirmación actualmente extraída.

Aparte del directorio de .git, en el directorio raíz del proyecto, se conserva un proyecto existente sin modificar (a diferencia de SVN, Git no requiere un subdirectorio de .git en cada subdirectorio).

## 4-Que es una rama?

Las ramas en git son una división del estado del código, esto permite crear nuevos caminos a favor de la evolución del código. Normalmente la creación de ramas en otros sistemas de control de versiones puede tomar mucho tiempo y ocupar demasiado espacio en el almacenamiento. Por el contrario, las ramas en Git son parte diaria del desarrollo, son una guía instantánea para los cambios realizados.

Por ejemplo, imagina que quieres añadir una nueva función o tal vez arreglar un error, sin importar su tamaño, generas una nueva rama en la cual se alojan estos cambios que realizaste, al realizar esta acción va resultar más complicado que algún error o fallo del código inestable se incorpore al código base principal, dando la oportunidad de limpiar tu historial antes de fusionarlo todo con la rama principal, mejorando tu eficiencia de trabajo.

Para poder utilizar las ramas en Git es necesario el comando git branch y la implementación que estas realizan en Git es mucho más sencilla que la de otros modelos de sistemas de control de versiones.

## 3-Como saber es que rama estoy?

En Git, una "rama" es una línea de desarrollo independiente que permite trabajar en diferentes funcionalidades, correcciones o cambios sin afectar la rama principal del proyecto (generalmente denominada rama "master" o "main"). Para saber en qué rama te encuentras actualmente en Git, puedes utilizar el siguiente comando en la terminal o línea de comandos:

- **Git Branch**

Este comando te mostrará todas las ramas disponibles en tu repositorio local y resaltará la rama actual con un asterisco (*) junto a su nombre. La rama actual, en la que te encuentras trabajando, estará marcada con ese asterisco.

## 5-Quien creo git?

Git fue creado por Linus Torvalds, el mismo creador del kernel de Linux. Linus creó Git en 2005 como una herramienta de control de versiones distribuido y de código abierto para ayudar en el desarrollo del kernel de Linux.

Inicialmente, Linus creó Git debido a las limitaciones y dificultades que enfrentaba con otros sistemas de control de versiones disponibles en ese momento para administrar el desarrollo colaborativo del kernel de Linux. Quería una herramienta que fuera rápida, eficiente, capaz de manejar grandes volúmenes de código y que permitiera a los desarrolladores trabajar de manera descentralizada, lo que llevó a la creación de Git. Desde entonces, Git se ha convertido en una de las herramientas de control de versiones más utilizadas en la industria del desarrollo de software.

**6-Cuales son los comandos más esenciales de Git?**

Existen varios comandos esenciales en Git que son fundamentales para comenzar a trabajar con este sistema de control de versiones.

1. **git init:** Inicia un nuevo repositorio Git en un directorio local.
2. **git clone:** Clona un repositorio Git existente desde una ubicación remota a tu sistema local.
3. **git add:** Agrega cambios al área de preparación (staging) para ser incluidos en el próximo commit.
4. **git commit:** Confirma los cambios preparados en el área de preparación y los guarda en el historial del repositorio.
5. **git status:** Muestra el estado actual de los archivos en el directorio de trabajo y en el área de preparación.
6. **git log:** Muestra el historial de commits del repositorio.
7. **git branch:** Lista, crea o elimina ramas. También se usa para cambiar entre ramas.
8. **git checkout**: Se utiliza para cambiar de rama o para restaurar archivos de un commit anterior.
9. **git pull:** Descarga los cambios desde un repositorio remoto y los fusiona con tu rama actual.
10. **git push:** Envía los commits locales al repositorio remoto.
11. **git merge:** Fusiona una rama con otra rama activa.
12. **git remote:** Gestiona las conexiones remotas a repositorios.

**7-Que es git Flow?**

GitFlow se define como un sistema de branching o ramificación o modelo de manejo de ramas en Git, en el que se usan las ramas principales y la feature. De modo que la rama feature la crean los desarrolladores para fusionarla con la rama principal, únicamente cuando cumpla con sus labores.

Este sistema se caracteriza, además, por incluir múltiples ramas de mayor duración y más commit o confirmaciones.

**8-Que es trunk based development?**

Trunk Based Development es una metodología de desarrollo de software que se centra en trabajar directamente sobre la rama principal (conocida como trunk, main o master) del repositorio de código. Esta metodología promueve la colaboración continua y la integración frecuente de cambios en la rama principal del repositorio, en contraste con modelos de ramificación prolongada como GitFlow o modelos de desarrollo basados en ramas largas.

En el Trunk Based Development:

1. **Desarrollo en la rama principal:** Los desarrolladores realizan cambios directamente en la rama principal del repositorio de código. Esto implica que todos los cambios se realizan en la misma línea de desarrollo, lo que facilita una mayor visibilidad y colaboración entre los equipos.
2. **Integración continua:** Se fomenta la integración constante de los cambios en la rama principal varias veces al día. Los desarrolladores se comprometen a mantener el código en un estado funcional y listo para producción.
3. **Pequeños y frecuentes commits:** Se promueve la realización de cambios pequeños y frecuentes en lugar de cambios masivos y complejos. Esto ayuda a reducir el riesgo de conflictos y problemas de integración.
4. **Pruebas automáticas: El** Trunk Based Development generalmente está asociado con prácticas de pruebas automáticas intensivas. Esto garantiza que los cambios integrados en la rama principal no rompan la funcionalidad existente y mantengan la estabilidad del sistema.
5. **Rápida resolución de problemas:** Al mantener una sola rama principal y realizar integraciones frecuentes, cualquier problema o conflicto puede identificarse y solucionarse rápidamente, lo que reduce la cantidad de trabajo necesario para resolver conflictos de integración más adelante.

## Ejercicio Práctico

https://github.com/ElRoy14/Tarea3-Programacion3.git