REPÚBLICA DOMINICANA

INSTITUTO TECNOLÓGICO DE LAS AMÉRICAS (ITLA)

DESARROLLO DE SOFTWARE

Programación III

Asignación: Tarea 3

Yulian Rafael De Los Santos

2022-0592

Group #02

Prof. Kelyn Tejada Belliard

Santo Domingo, República Dominicana, Distrito Nacional

Julio del 2023

Desarrolla el siguiente Cuestionario:

1-Que es Git?

Git es un sistema de control de versiones distribuido y de código abierto que se utiliza principalmente para el desarrollo de software. Fue creado por Linus Torvalds en 2005 y se ha convertido en uno de los sistemas de control de versiones más populares y ampliamente utilizados en el mundo del desarrollo de software.

Un sistema de control de versiones es una herramienta que permite a los desarrolladores rastrear y gestionar los cambios realizados en el código fuente y otros archivos de un proyecto a lo largo del tiempo. Esto facilita el trabajo en equipo, la colaboración, y ayuda a evitar conflictos cuando varias personas trabajan simultáneamente en el mismo proyecto.

Git es "distribuido", lo que significa que cada usuario que trabaja en el proyecto tiene una copia completa del repositorio (el lugar donde se almacenan los archivos y la historia de cambios). Esto permite que los desarrolladores trabajen de manera independiente y realicen cambios sin estar conectados a un servidor central todo el tiempo. Luego, pueden sincronizar sus cambios con el repositorio principal o con otros colaboradores cuando sea necesario.

Algunas características importantes de Git son:

- **Ramificación y fusión (Branching and Merging):** Permite a los desarrolladores crear ramas separadas para trabajar en nuevas características o correcciones de errores sin afectar el código principal. Luego, pueden combinar o fusionar estas ramas con la rama principal una vez que sus cambios han sido probados y revisados.
- **Historial completo de cambios (Commit History):** Git registra todos los cambios realizados en el repositorio a lo largo del tiempo, lo que facilita la visualización y el seguimiento de quién hizo qué cambios y cuándo.
- **Rastreo de cambios (Tracking Changes):** Los desarrolladores pueden ver las diferencias entre versiones anteriores y actuales de los archivos, lo que ayuda a identificar y resolver problemas.
- **Revertir cambios (Reverting):** Si un cambio introducido causa problemas, es posible revertir el código a una versión anterior y deshacer los cambios problemáticos.
- **Colaboración y trabajo en equipo (Collaboration and Teamwork):** Varios desarrolladores pueden colaborar en un proyecto al mismo tiempo, y Git facilita la combinación de sus cambios de manera ordenada y sin conflictos.
- **Integración con servicios de alojamiento (Integration with Hosting Services):** Git se integra con plataformas de alojamiento en línea, como GitHub, GitLab y Bitbucket, lo que proporciona un entorno colaborativo y seguro para compartir y gestionar proyectos.

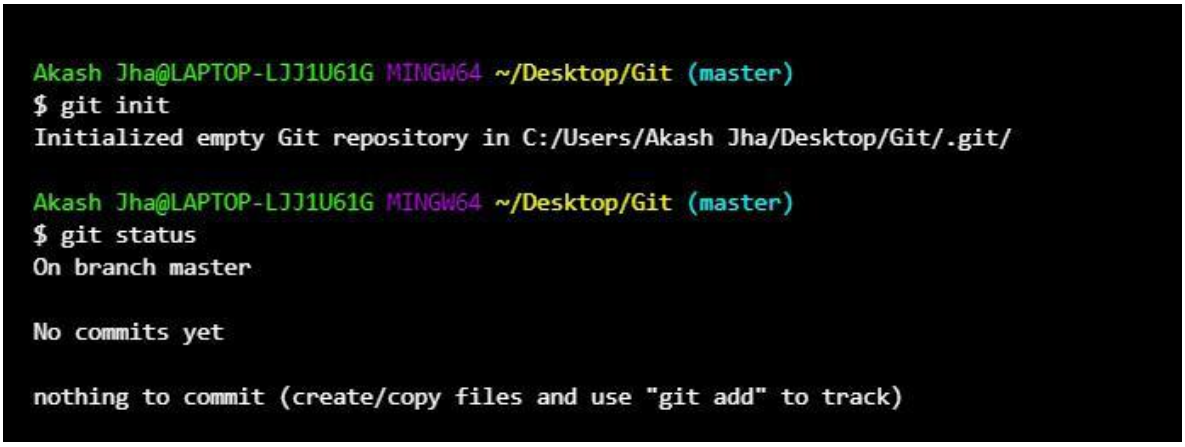
2-Para que funciona el comando Git init?

El comando "git init" se utiliza para crear un nuevo repositorio de Git en un directorio específico. Cuando ejecutas "git init", Git inicializa un nuevo repositorio vacío en el directorio actual o en el directorio que especifiques como argumento.

En otras palabras, "git init" convierte un directorio normal en un repositorio de Git, lo que permite que dicho directorio y sus subdirectorios sean

rastreados y gestionados por Git para controlar las versiones de los archivos y la colaboración en el desarrollo de software u otros proyectos.

Es importante tener en cuenta que solo se debe ejecutar "git init" una vez en el directorio raíz del proyecto. Una vez que se inicializa el repositorio, Git estará activo y listo para rastrear y controlar los cambios en los archivos del proyecto. A partir de ese momento, se pueden utilizar los demás comandos de Git, como "git add", "git commit", "git branch", etc., para administrar el repositorio y trabajar con el control de versiones.



Algunas acciones que realiza "git init":

- **Crea la estructura del repositorio:** El comando "git init" crea una subcarpeta oculta llamada ".git" en el directorio raíz del proyecto. Esta carpeta contiene toda la información necesaria para que Git controle el historial de cambios y gestione los archivos.
- **Prepara el repositorio para seguimiento de cambios**: Después de ejecutar "git init", Git está listo para rastrear y controlar los cambios realizados en los archivos dentro del directorio y sus subdirectorios.
- **Inicia una rama principal (master/main):** Tras la inicialización, Git crea automáticamente una rama principal llamada "master" o "main", dependiendo de la configuración del sistema. Es en esta rama donde comenzará el desarrollo del proyecto.
- **Lista de archivos sin seguimiento:** Cuando se inicializa un repositorio, todos los archivos en el directorio aparecen como "sin seguimiento" hasta que se agreguen al área de preparación (staging area) con el comando "git add".

3-Que es una rama?

En el contexto de Git y otros sistemas de control de versiones, una rama (branch en inglés) es una línea independiente de desarrollo que se deriva del código fuente principal (rama principal) de un repositorio. Puedes pensar en una rama como una copia separada del proyecto en la que puedes hacer cambios sin afectar directamente la versión principal del proyecto. Cada rama tiene su propio historial de cambios, lo que permite que varios desarrolladores trabajen en diferentes características o correcciones de errores de forma paralela y aislada.

La rama principal del repositorio suele denominarse "master" o "main" (dependiendo de la configuración y las convenciones de nomenclatura). A medida que se desarrolla el proyecto, es común crear nuevas ramas para trabajar en funcionalidades específicas o para corregir problemas.

Cuando se crea una nueva rama, inicialmente será idéntica a la rama de la que se deriva, por lo general la rama principal. A medida que se realizan cambios en la nueva rama, estos cambios se almacenan de forma independiente en su historial de cambios. Esto permite que los cambios se mantengan separados del código en la rama principal hasta que estén listos para ser incorporados.

Una vez que se completan los cambios en una rama y se consideran estables y probados, se puede "fusionar" la rama con la rama principal para incorporar esos cambios en el proyecto principal. La fusión combina los cambios realizados en la rama secundaria con la rama principal, lo que permite que el proyecto principal se actualice con las últimas modificaciones.

Las ramas son útiles por varias razones:

Desarrollo aislado: Permite a los desarrolladores trabajar en nuevas funcionalidades o solucionar problemas sin afectar el estado estable del proyecto principal.

Colaboración: Varios desarrolladores pueden trabajar en diferentes ramas al mismo tiempo y luego fusionar sus cambios sin interferir entre sí.

Experimentación: Se pueden crear ramas para probar ideas sin afectar el flujo de trabajo principal.

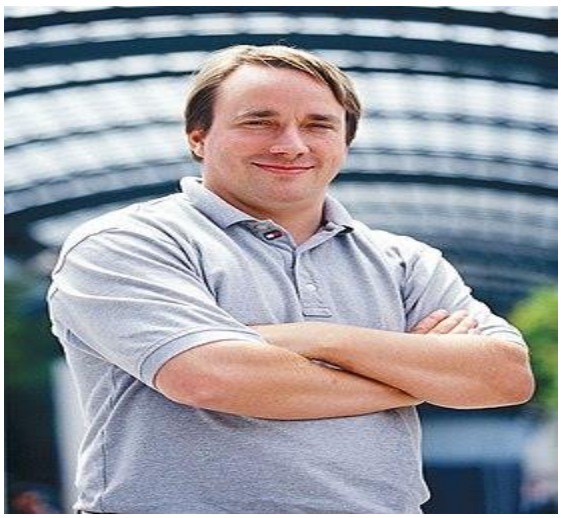
Versionado de características: Cada rama puede representar una característica específica en desarrollo, lo que facilita el seguimiento y la gestión de versiones.

4-Como saber en qué rama estoy?

Según investigue para saber en qué rama me encuentro en un repositorio Git, puedo utilizar el comando “git branch” y con este comando se me mostrará todas las ramas existentes en el repositorio y resaltará con un asterisco (*) la rama en la que me encuentre actualmente.

Además del comando git branch, también se puede utilizar “git status” para obtener información sobre el estado actual del repositorio, incluida la rama en la que me encuentre y se mostrará información sobre los cambios realizados en los archivos y la rama actual también.

5-Quien creo git?



Git fue creado por Linus Torvalds, el mismo desarrollador que creó el kernel (núcleo) del sistema operativo Linux. Linus desarrolló Git en 2005 para abordar las deficiencias de otros sistemas de control de versiones disponibles en ese momento. Su objetivo era crear una herramienta de control de versiones rápida, eficiente, confiable y fácil de usar para el desarrollo colaborativo del kernel de Linux y otros proyectos de código abierto.

La creación de Git fue impulsada por la necesidad de un sistema de control de versiones distribuido que pudiera manejar grandes proyectos de código abierto y mantener el rendimiento a medida que crecía el número de contribuyentes. Antes de Git, otros sistemas de control de versiones centralizados y basados en servidor, como CVS y Subversion, eran comunes, pero presentaban limitaciones en cuanto a escalabilidad y capacidad de trabajo desconectado.

6-Cuales son los comandos más esenciales de Git?

Los comandos más esenciales de Git son aquellos que nos permiten realizar las tareas básicas de control de versiones y colaborar en proyectos de desarrollo de software y son los siguientes:

- **Git init:** Inicializa un nuevo repositorio de Git en un directorio, convirtiéndolo en un repositorio local.
- **Git clone [URL]:** Clona un repositorio remoto existente en tu máquina local, creando una copia completa del proyecto.
- **Git add [archivo(s)]:** Agrega los cambios realizados en uno o varios archivos al área de preparación (staging area) para que se incluyan en el próximo commit.
- **Git commit -m "mensaje del commit":** Crea un nuevo commit con los cambios que se encuentran en el área de preparación. El mensaje del commit debe describir brevemente los cambios realizados.
- **Git status:** Muestra el estado actual del repositorio, incluyendo los archivos modificados, los cambios sin preparar y la rama actual.
- **Git-log:** Muestra el historial de commits realizados en el repositorio, incluyendo información sobre el autor, fecha y mensaje de cada commit.
- **Git pull:** Trae los cambios desde el repositorio remoto y los fusiona con la rama actual en tu repositorio local.
- **Git push:** Envía los cambios locales a un repositorio remoto, actualizando la rama correspondiente en el repositorio remoto.
- **Git branch:** Muestra una lista de todas las ramas del repositorio y resalta con un asterisco la rama en la que te encuentras actualmente.
- **Git checkout [rama]:** Cambia a la rama especificada, permitiéndote trabajar en una línea de desarrollo diferente.
- **Git merge [rama]:** Fusiona los cambios de la rama especificada en la rama actual. Puede generar conflictos que deben ser resueltos manualmente.
- **Git remote:** Muestra una lista de los repositorios remotos vinculados al repositorio local.
- **Git remote add [nombre] [URL]:** Agrega un nuevo repositorio remoto y lo asocia con un nombre específico para facilitar las operaciones de pull y push.

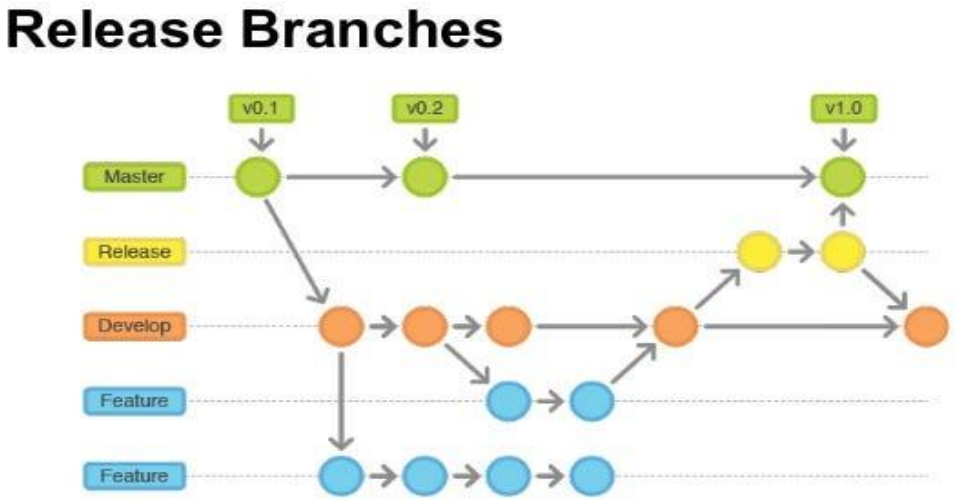
- **Git stash:** Guarda temporalmente los cambios sin confirmar en un lugar temporal (stash) para que puedas cambiar de rama o realizar otras acciones sin comprometer los cambios actuales.
- **Git diff:** Muestra las diferencias entre el directorio de trabajo y el área de preparación (staging area).

7-Que es git Flow?

Git Flow es un conjunto de pautas y flujos de trabajo propuesto por Vincent Driessen en 2010 para el uso de Git en proyectos de desarrollo de software. Proporciona una estructura y un conjunto de reglas para organizar las ramas y el flujo de desarrollo en un repositorio Git, con el objetivo de facilitar la colaboración entre equipos y gestionar el desarrollo de características y versiones de forma ordenada.

El flujo de trabajo de Git Flow ayuda a mantener una estructura clara para el desarrollo y la liberación de software. Facilita la colaboración entre equipos, permite trabajar en paralelo en múltiples características y corrige errores de manera controlada.

Es importante tener en cuenta que, si bien Git Flow es una metodología popular, existen otros flujos de trabajo y prácticas de ramificación que se adaptan a diferentes contextos y necesidades de desarrollo. Algunas comunidades y equipos han optado por flujos de trabajo más simplificados o adaptados a sus propios procesos.



El flujo de trabajo de Git Flow se basa en el uso de diferentes tipos de ramas con propósitos específicos. Las ramas principales que utiliza Git Flow son:

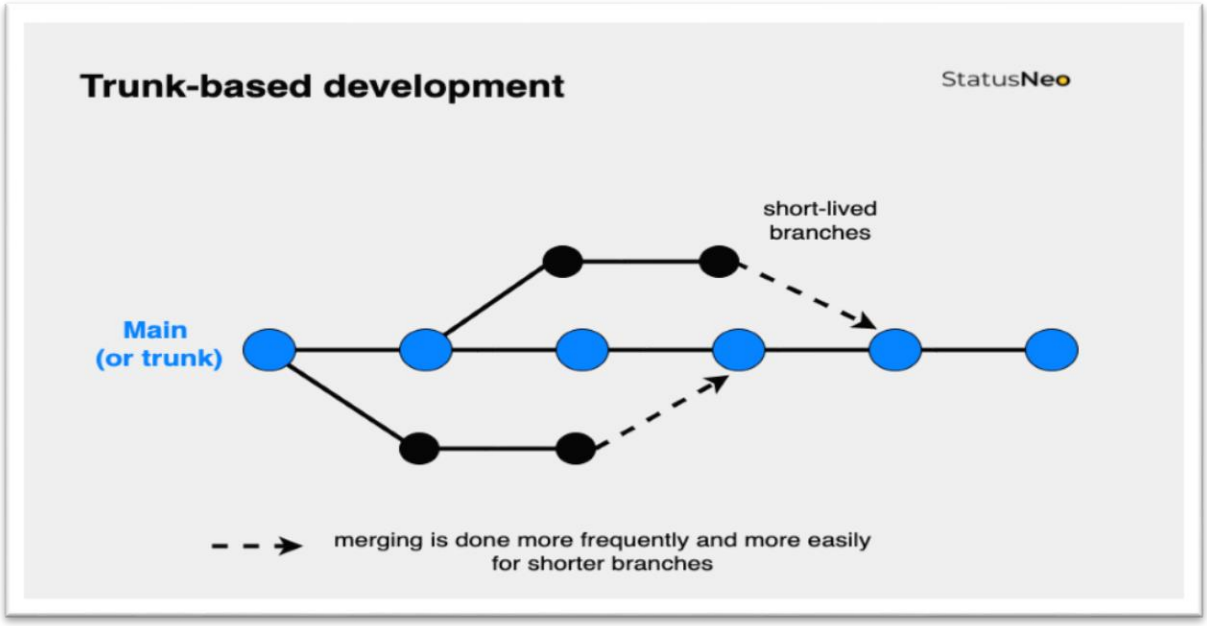
- **Master (main):** Esta es la rama principal del repositorio y contiene el código que se considera estable y listo para producción. Las versiones de lanzamiento se etiquetan desde esta rama.
- **Develop:** La rama "develop" es donde se integran las características desarrolladas por los diferentes miembros del equipo. Aquí se encuentra el código en desarrollo que se considera lo suficientemente estable como para compartir con otros desarrolladores.

Además de estas dos ramas principales, Git Flow utiliza ramas auxiliares para diferentes propósitos:

- **Feature branches (ramas de características):** Se crean a partir de la rama "develop" y se utilizan para desarrollar nuevas características o funcionalidades. Una vez que una característica está completa, se fusiona de nuevo en la rama "develop".
- **Release branches (ramas de lanzamiento):** Se crean a partir de la rama "develop" y se utilizan para preparar una versión para su lanzamiento. En esta rama, se realizan correcciones de errores y ajustes finales antes de la versión final. Una vez que la versión está lista, se fusiona en ambas ramas "master" y "develop", y se etiqueta con un número de versión.
- **Hotfix branches (ramas de correcciones rápidas):** Se crean a partir de la rama "master" (o "main") y se utilizan para corregir rápidamente problemas críticos en la producción. Una vez que se completa la corrección, se fusiona tanto en la rama "master" como en "develop".

8-Que es trunk based development?

El Trunk Based Development (TBD) es un enfoque de desarrollo de software que promueve mantener una rama principal (trunk) única y estable como el centro del desarrollo en un repositorio Git. En este modelo, los desarrolladores trabajan directamente en la rama principal (por lo general llamada "main" o "master") en lugar de mantener ramas separadas para características o correcciones de errores.



El Trunk Based Development es adecuado para equipos ágiles que valoran la velocidad, la entrega continua y la colaboración directa. Al permitir que los desarrolladores trabajen en la rama principal y hagan pequeñas iteraciones, el TBD ayuda a reducir la complejidad y los conflictos de fusión, lo que a su vez mejora la productividad y la calidad del software.

Es importante mencionar que el Trunk Based Development puede no ser adecuado para todos los escenarios. En proyectos muy grandes o con muchos equipos trabajando en paralelo, puede resultar más complejo mantener una rama principal completamente estable y pueden surgir problemas de integración más difíciles de manejar. En esos casos, otros flujos de trabajo como Git Flow pueden ser más apropiados. La elección del flujo de trabajo dependerá de las necesidades y características específicas de cada equipo y proyecto.