Nombre:

Carlos Wilmer Arias Almanzar

Matricula:

2022-0021

Materia:

Programación 3

Sección:

2

Maestro:

Kelyn Tejada Belliard

Tarea 3

Desarrollar el siguiente Cuestionario:

Repositorio Git Hub: https://github.com/CarlosWAriasA/Tarea-3---Programacion-3.git

¿Qué es Git?

Git es un sistema de control de versiones que permite a los desarrolladores realizar un seguimiento de los cambios en el código fuente a lo largo del tiempo. Es distribuido, lo que significa que cada miembro del equipo de desarrollo tiene una copia completa del repositorio, incluidas todas las versiones históricas. Esto facilita el trabajo en equipo, la colaboración y la posibilidad de trabajar de forma independiente sin depender de un servidor centralizado.

Algunas de las características de git son:

Rápido y Eficiente: Git ha sido diseñado para ser rápido y eficiente, lo que es esencial en proyectos grandes con muchos archivos y colaboradores. Las operaciones como commit, branch y merge se ejecutan en local, lo que reduce la necesidad de una conexión constante con un servidor.

Branching y Merging: Una de las mayores aportaciones de Git es su sistema de branching y merging. Los desarrolladores pueden crear fácilmente ramas independientes para trabajar en nuevas funcionalidades o arreglar problemas sin afectar la rama principal. Luego, pueden combinar esas ramas una vez que se hayan probado y estén listas para integrarse en la rama principal.

Revisión del Código: Git facilita la revisión del código al permitir a los desarrolladores crear pull requests o solicitudes de extracción. Esto permite que otros miembros del equipo revisen el código, realicen comentarios y sugieran cambios antes de fusionar el código en el repositorio principal. Esta práctica promueve la calidad del código y evita errores costosos.

Historial Completo y Auditable: Git mantiene un historial completo y detallado de todos los cambios realizados en el repositorio. Cada commit contiene información sobre quién realizó el cambio, cuándo se realizó y qué archivos se modificaron. Esto proporciona un registro histórico del desarrollo del proyecto y es útil para fines de auditoría y resolución de problemas.

Descentralización: Al ser un sistema de control de versiones distribuido, Git permite a los desarrolladores trabajar sin conexión y realizar commits localmente. Esto es especialmente valioso para equipos distribuidos o para aquellos que trabajan en áreas con conectividad limitada.

¿Para que funciona el comando Git init?

El comando git init se utiliza para iniciar un nuevo repositorio de Git en un directorio vacío o existente. Cuando se ejecuta este comando en un directorio, Git crea una nueva estructura de control de versiones en ese directorio y comienza a realizar el seguimiento de los cambios en los archivos que se agregan a él.

Cuando ejecutamos git init, suceden varias cosas:

Se crea un subdirectorio oculto llamado “git” en el directorio actual. Este subdirectorio es donde Git almacena toda la información sobre el repositorio, incluidos los metadatos, configuraciones, referencias a commits y ramas, entre otros.

El repositorio se inicializa con una rama maestra (por defecto llamada "master"). Esta rama es la que contendrá el historial completo de cambios realizados en el repositorio.

Git configura algunas opciones predeterminadas para el repositorio, como el nombre y correo electrónico del autor de los commits.

¿Qué es una rama?

Una rama (también conocida como "branch") es una línea independiente de desarrollo que se crea a partir de un punto específico en la historia del repositorio. Una rama se puede imaginar como una línea de tiempo separada en la que se puede trabajar de forma aislada sin afectar directamente la rama principal.

Cada repositorio de Git comienza con una rama predeterminada llamada "master" o "main" (dependiendo de la configuración), que contiene el historial completo de desarrollo del proyecto hasta ese momento. Al crear una nueva rama, esta copia todos los commits de la rama principal en ese punto, lo que crea una bifurcación en el historial. A partir de ese momento, los cambios que se realicen en la nueva rama no afectarán la rama principal hasta que se fusionen nuevamente.

Las ramas son útiles porque permiten:

Desarrollo paralelo: Varios miembros del equipo pueden trabajar en diferentes funcionalidades o problemas en ramas separadas al mismo tiempo, sin interferir en el trabajo de los demás. Esto promueve la colaboración y la productividad.

Experimentación segura: Podemos crear una rama para probar nuevas ideas o cambios significativos sin afectar la estabilidad de la rama principal. Si los cambios en la rama experimental no funcionan según lo esperado, podemos descartar la rama sin afectar la rama principal.

Versiones estables: La rama principal (master o main) generalmente contiene una versión estable y funcional del proyecto. Al crear nuevas ramas para trabajar en características o correcciones, la rama principal permanece en un estado estable y utilizable.

Control de versiones: Git facilita el seguimiento de las diferentes ramas y permite fusionar los cambios de una rama en otra cuando estén listos para integrarse.

¿Como saber es que rama estoy?

Para saber en qué rama nos encontramos en un repositorio de Git, podemos utilizar el comando: “git Branch”.

Ejecutando este comando en la línea de comandos o en la terminal, Git mostrará una lista de todas las ramas existentes en el repositorio y resaltará con un asterisco (*) la rama en la que nos encontramos actualmente. La rama activa es aquella en la que realizaremos los nuevos commits, a menos que cambiemos a otra rama.

Por ejemplo, si el resultado del comando es algo como:

--

* main

development

feature/new-feature

--

Significa que nos encontramos en la rama principal (main) y el asterisco (*) indica la rama activa.

¿Quién creo git?

Git fue creado por Linus Torvalds, el mismo desarrollador que creó el kernel del sistema operativo Linux. Linus comenzó el desarrollo de Git en abril de 2005 debido a sus frustraciones con otros sistemas de control de versiones existentes en ese momento. Quería una herramienta de control de versiones que fuera rápida, eficiente y que funcionara bien con proyectos de código abierto grandes y complejos como el kernel de Linux.

Así, Linus desarrolló Git y lo lanzó como un proyecto de código abierto en el que otros desarrolladores pudieran contribuir y mejorar la herramienta. Con el tiempo, Git se convirtió en uno de los sistemas de control de versiones más populares y ampliamente utilizados en la comunidad de desarrollo de software, y ha sido fundamental en la colaboración y gestión del código fuente en proyectos de todas las escalas.

¿Cuáles son los comandos más esenciales de Git?

Los comandos más esenciales de Git son aquellos que permiten realizar las tareas básicas de control de versiones en el repositorio. Algunos de los comandos más importantes son:

git init: Inicializa un nuevo repositorio de Git en un directorio vacío o existente.

git clone [URL]: Clona un repositorio existente desde una URL remota a tu sistema local.

git add [archivos]: Agrega archivos al área de preparación (staging area) para ser incluidos en el próximo commit.

git commit -m "mensaje": Realiza un commit con los cambios en el área de preparación y agrega un mensaje descriptivo.

git status: Muestra el estado actual del repositorio, incluyendo los archivos modificados, en el área de preparación y sin seguimiento.

git log: Muestra el historial de commits realizados en la rama actual, mostrando detalles como el autor, fecha y mensaje del commit.

git pull: Descarga los cambios desde el repositorio remoto y los fusiona con tu rama local.

git push: Sube los cambios locales al repositorio remoto.

git branch: Muestra una lista de las ramas existentes y resalta la rama activa.

git checkout [rama]: Cambia a la rama especificada.

git merge [rama]: Fusiona la rama especificada con la rama actual.

git remote -v: Muestra las URL remotas configuradas para el repositorio.

¿Qué es git Flow?

GitFlow es un modelo de flujo de trabajo o una metodología para el control de versiones en Git, diseñada para facilitar el desarrollo colaborativo de software y la gestión de ramas en proyectos más complejos. Fue popularizado por Vincent Driessen en 2010 y ha sido ampliamente adoptado en la comunidad de desarrollo de software.

El flujo de trabajo GitFlow se basa en el uso de diferentes tipos de ramas para organizar el desarrollo y las versiones del software. Las ramas principales que se utilizan en GitFlow son las siguientes:

Master/Main: La rama principal del proyecto, donde se encuentra el código estable y probado. En esta rama solo se encuentran versiones que han sido liberadas y están listas para producción.

Develop: La rama de desarrollo, donde se integran todas las características y cambios que están en desarrollo. Es una rama intermedia entre la rama principal y las ramas de funcionalidades.

Feature branches: Son ramas creadas desde la rama de desarrollo para implementar nuevas características o funcionalidades. Una vez que se completa el desarrollo de una característica, se fusiona de nuevo con la rama de desarrollo.

Release branches: Son ramas creadas desde la rama de desarrollo para preparar una nueva versión para su lanzamiento. Se utilizan para realizar pruebas finales, correcciones de errores y preparar la versión final antes de ser fusionada con la rama principal.

Hotfix branches: Son ramas creadas desde la rama principal con el propósito de solucionar problemas críticos en producción. Una vez que se corrige el error, la rama de hotfix se fusiona tanto con la rama principal como con la rama de desarrollo.

El flujo de trabajo de GitFlow se basa en la idea de tener una rama de desarrollo separada para integrar todas las características nuevas y evitar trabajar directamente en la rama principal, lo que minimiza el riesgo de problemas en producción. Además, las ramas de características, versiones y correcciones de errores facilitan el aislamiento y la organización del desarrollo.

¿Qué es trunk based development?

El Trunk Based Development (TBD) es un enfoque de desarrollo de software que se basa en la utilización de una única rama principal (trunk) como la rama principal de desarrollo, eliminando o minimizando la creación de ramas de larga duración. En este modelo, los desarrolladores trabajan directamente en la rama principal y realizan commits frecuentes para integrar sus cambios.

Las características y principios clave del Trunk Based Development incluyen:

Rama Principal (Trunk): En el Trunk Based Development, la rama principal (trunk o rama principal) es considerada como la fuente de verdad y es donde se encuentran siempre los cambios más recientes y estables. A diferencia de otros enfoques, no se crean ramas de desarrollo prolongadas ni ramas de características, y todas las adiciones de código se realizan directamente en la rama principal.

Commits Frecuentes: Los desarrolladores realizan commits frecuentes en la rama principal después de implementar una pequeña cantidad de cambios. Esto fomenta la integración continua y evita la acumulación de cambios no probados.

Pequeños Incrementos: En lugar de trabajar en grandes funcionalidades en ramas separadas, los desarrolladores buscan realizar pequeños incrementos en la rama principal, lo que facilita la revisión y la detección temprana de problemas.

Pruebas Automatizadas: El uso de pruebas automatizadas es fundamental en el Trunk Based Development. Los equipos implementan suites de pruebas automatizadas para garantizar que los cambios no introduzcan errores en el código existente.

Feature Flags: Para características en desarrollo que aún no están listas para su activación en producción, se pueden utilizar "feature flags" o "banderas de características" que permiten habilitar o deshabilitar una funcionalidad de forma controlada y segura en el entorno de producción.

El Trunk Based Development es una metodología que promueve la colaboración y la integración continua del código en la rama principal, lo que ayuda a reducir el riesgo de problemas causados por la integración tardía de cambios y facilita una entrega continua y estable del software. Sin embargo, este enfoque puede no ser adecuado para todos los proyectos, especialmente aquellos con una complejidad considerable o equipos muy grandes, donde el uso de ramas de características puede facilitar un desarrollo más organizado y estructurado.