Nombre y Apellido

Jose Manuel Santillan

Matricula

2023-1156

Nombre del maestro/a

Kelyn Tejada

Materia

Programación III

Asignación

Evaluación de conocimientos sobre Git y Git Flow

Fecha

04/04/2025

Índice

Bibliografía:

1. **¿Qué es Git?** Qué es Git | Atlassian Git Tutorial
2. **¿Para qué sirve el comando git init?** git init | Atlassian Git Tutorial
3. **¿Qué es una rama en Git?** Ramas en Git: Qué son y cómo funcionan [2025] | KeepCoding
4. **¿Cómo saber en cuál rama estoy trabajando?** ¿Cómo saber en qué rama estoy en Git? Guía completa en español - Como Reclamar
5. **¿Quién creó Git?** Git - Wikipedia, la enciclopedia libre
6. **¿Cuáles son los comandos esenciales de Git?** Comandos de GIT Básicos - Guía Completa
7. **¿Qué es Git Flow?** Git Flow: Qué Es y Cómo Funciona » CodigoNautas
8. **¿Qué es el desarrollo basado en trunk (Trunk Based Development)?** Desarrollo basado en troncos | Atlassian

1. ¿Qué es Git?

Git es el sistema de control de versiones moderno más utilizado en el mundo, desarrollado originalmente por Linus Torvalds en 2005. Es un software de código abierto con una arquitectura distribuida, lo que significa que cada desarrollador tiene una copia completa del historial del proyecto.

Git está diseñado con un enfoque en el rendimiento, seguridad y flexibilidad. Su estructura permite realizar tareas como confirmar cambios, crear ramas, fusionar y comparar versiones de forma rápida y eficiente. Además, protege el historial del proyecto mediante un sistema de hash criptográfico (SHA-1), asegurando la integridad del código y la trazabilidad de los cambios.

2. ¿Para qué sirve el comando git init?

El comando git init se utiliza para crear un nuevo repositorio de Git. Puede convertir un proyecto ya existente en un repositorio con control de versiones, o iniciar uno completamente vacío. Es, por lo general, el primer comando que se ejecuta al comenzar un nuevo proyecto con Git.

Cuando se ejecuta, git init crea un subdirectorio oculto llamado .git, que contiene todos los metadatos necesarios para el control de versiones, incluyendo objetos, referencias, archivos de configuración y un archivo HEAD que apunta a la rama activa. El contenido del proyecto en sí no se modifica, lo que lo hace ideal para proyectos ya iniciados.

Además, permite personalizar la ubicación del directorio .git mediante la variable de entorno $GIT_DIR o la opción --separate-git-dir, lo cual es útil para separar la configuración del repositorio del código fuente.

A diferencia de sistemas como SVN, Git no requiere privilegios de administrador ni un servidor central para comenzar a trabajar. Solo se necesita ejecutar git init dentro de un directorio y se habilita de inmediato el registro de cambios del proyecto.

También puede usarse como:

- git init Inicializa un repositorio en el directorio actual.
- git init <directorio> → Crea un nuevo repositorio vacío en el directorio especificado.

Reejecutar git init en un proyecto que ya tiene un repositorio no sobrescribirá su configuración existente, lo cual lo hace seguro para mantener la configuración intacta.

3. ¿Qué es una rama en Git?

Una rama en Git es un puntero que señala un commit específico dentro del historial del proyecto. Desde ese punto, se pueden crear nuevos commits, lo que permite desarrollar funcionalidades o hacer pruebas de forma aislada sin afectar la rama principal generalmente main o master.

Usar ramas es como crear una copia temporal del proyecto, ideal para trabajar en nuevas ideas o correcciones sin poner en riesgo el código principal. Si los cambios resultan útiles, se pueden integrar (fusionar) a la rama principal; si no, simplemente se descartan.

Ventajas de usar ramas en Git:

- **Aislamiento de cambios:** Permiten trabajar en nuevas funciones o correcciones sin interferir con el desarrollo principal.
- **Colaboración eficiente:** Cada desarrollador puede trabajar en su propia rama, lo cual reduce los conflictos de código.

- **Organización del historial:** Facilitan un historial de commits más limpio y comprensible.
- **Pruebas seguras:** Se puede experimentar libremente sin comprometer la estabilidad del código en producción.

4. ¿Cómo saber en cuál rama estoy trabajando?

Para saber en qué rama está usted en Git, se pueden seguir los siguientes pasos sin repetir y sin dejar espacios entre ellos:

1. Abrir la terminal.
2. Ir al directorio del repositorio con cd.
3. Ejecutar git branch.
4. La rama activa aparecerá con un * al lado de su nombre.

5. ¿Quién creó Git?

Git fue creado por Linus Torvalds, el mismo desarrollador del kernel de Linux, en abril de 2005. Su objetivo era contar con un sistema de control de versiones eficiente, confiable y compatible con proyectos que manejan grandes cantidades de archivos de código fuente.

Torvalds desarrolló Git luego de que se revocara la licencia gratuita de BitKeeper, el sistema que usaban para gestionar el código del kernel de Linux. Esta revocación surgió a raíz de una controversia relacionada con ingeniería inversa, lo que motivó a la comunidad a desarrollar una herramienta libre y robusta como Git.

6. ¿Cuáles son los comandos esenciales de Git?

- **git init:** Inicializa un nuevo repositorio de Git en el directorio actual.
- **git clone /path/to/repository:** Clona un repositorio existente (ya sea local o remoto).
- **git add <archivo>:** Agrega archivos al área de preparación (staging area) para ser confirmados.

- **git config:** Configura opciones de Git como el nombre y el correo del usuario.

- **git status:** Muestra el estado actual de los archivos (cambiados, preparados, sin seguimiento, etc.).
- **git push origin <rama>:** Envía los commits locales al repositorio remoto, a la rama especificada.

- **git pull:** Descarga y fusiona los cambios del repositorio remoto con el directorio de trabajo local.

- **git Branch:** Lista todas las ramas, crea nuevas o elimina ramas existentes.
- **git checkout <rama>:** Cambia a una rama existente.
- **git merge <rama>:** Fusiona el historial de la rama especificada con la rama actual.

7. ¿Qué es git Flow?

Git Flow es un enfoque de trabajo que organiza y gestiona el desarrollo en proyectos usando Git. Fue ideado por Vincent Driessen en 2009 y se ha vuelto muy popular por su claridad al definir roles específicos para diferentes ramas y cómo interactúan entre ellas, ideal para equipos de desarrollo trabajando en proyectos grandes y complejos.

Ramas principales

1. **Master**: Almacena el historial de lanzamientos oficiales y tiene un tiempo de vida indefinido. Los cambios de esta rama son estables y están listos para producción.
2. **Develop**: Es la rama de integración donde se gestiona el desarrollo activo. Contiene todo el historial del proyecto y, al estabilizarse el código, se fusiona con la rama Master.

Ramas de soporte

1. **Feature branches**: Se utilizan para desarrollar nuevas funciones. Se crean desde la rama Develop y regresan a ella al terminar la función.
2. **Release branches**: Se generan cuando hay suficientes funciones listas para un lanzamiento. Permiten preparar y estabilizar el producto antes de lanzarlo. Su origen también es Develop y, al finalizar, se fusionan tanto en Master como en Develop.
3. **Hotfix branches**: Son ramas de mantenimiento para corregir errores críticos en producción. Estas se crean desde Master y, al terminar, se fusionan tanto en Master como en Develop.

Ventajas de Git Flow

- **Simplificación del trabajo paralelo**: Aísla el desarrollo en ramas específicas, como funciones y correcciones, evitando interferencias.
- **Colaboración fluida**: Claridad al gestionar versiones del proyecto, lo cual mejora la comunicación del equipo.
- **Flexibilidad**: Permite trabajar en varias versiones del proyecto a la vez.
- **Mantenimiento ágil**: Facilita la solución de errores urgentes sin interrumpir el flujo general.

Problemas que resuelve

- Evita mala calidad del código y trabajo desorganizado.
- Mejora la satisfacción del cliente al entregar productos más estables.
- Ayuda a retener conocimiento sobre el proyecto en el equipo.
- Fomenta la responsabilidad colectiva frente a los problemas.

8. ¿Qué es el desarrollo basado en trunk (Trunk Based Development)?

El desarrollo basado en troncos es una práctica de control de versiones en la que los desarrolladores integran pequeñas actualizaciones frecuentemente en una rama principal o "tronco". Este enfoque es fundamental para la integración continua (CI) y la

entrega continua (CD), ya que reduce la complejidad de las fusiones y acelera la entrega de software, mejorando el rendimiento de los equipos.

En el pasado, antes de los sistemas modernos de control de versiones, los desarrolladores mantenían dos versiones de software en paralelo para gestionar los cambios, lo que era ineficiente y costoso. Con la evolución de estas herramientas, se definieron estilos como Git Flow y el desarrollo basado en troncos.

Git Flow es un modelo más rígido que centraliza la aprobación del código en unas pocas personas para mantener la calidad, mientras que el desarrollo basado en troncos es más abierto. Permite a todos los desarrolladores acceder al código principal, promoviendo iteraciones rápidas y facilitando la CI/CD. En este último modelo, el flujo de trabajo es más ágil al integrar cambios pequeños y frecuentes, eliminando la necesidad de ramas de larga duración y minimizando conflictos.

Este método es especialmente útil en entornos de DevOps, donde se busca mantener la rama principal estable y lista para producción en todo momento.