Presentación

Nombre y Apellido: Rosa Sanchez

Matricula: 20231707

Docente: Kelyn Tejada

Materia: Programación 3

Fecha: 02/04/2024

Índice

1. ¿Qué es Git?

Git es un sistema de control de versiones que se distribuye, lo que significa que puedes seguir los cambios en archivos y colaborar con otros desarrolladores de manera efectiva. Está diseñado para manejar proyectos de cualquier tamaño de forma rápida y eficiente. A diferencia de otros sistemas que dependen de un servidor central, Git guarda una copia completa del historial del proyecto en cada máquina que lo clona. Esto lo hace más fiable, ya que permite trabajar sin conexión y facilita la recuperación de versiones anteriores si algo sale mal.

2. ¿Para qué sirve el comando git init?

El comando git init se utiliza para crear un nuevo repositorio Git en una carpeta específica. Al ejecutar este comando, se genera un subdirectorio oculto llamado .git, que contiene todos los archivos y metadatos necesarios para seguir los cambios en el proyecto. Una vez que hayas ejecutado git init, puedes empezar a añadir archivos, realizar commits y gestionar versiones dentro del repositorio. Es el primer paso para transformar cualquier carpeta en un proyecto Git.

3. ¿Qué es una rama en Git?

Una rama en Git representa una versión independiente del código dentro del mismo repositorio. Esto permite trabajar en nuevas características, corregir errores o hacer experimentos sin afectar la rama principal del proyecto. Cada repositorio tiene al menos una rama principal, que generalmente se llama main o master. Desde esta rama, puedes crear otras para desarrollar nuevas funcionalidades y luego fusionarlas cuando estén listas. El uso de ramas permite que varios desarrolladores trabajen simultáneamente sin interferir en el trabajo de los demás.

4. ¿Cómo saber en cuál rama estoy trabajando?

Para averiguar en qué rama estás trabajando dentro de un repositorio Git, puedes usar un comando que muestra la lista de ramas disponibles y destaca la rama activa. También puedes ver la rama actual en algunas herramientas gráficas o en la terminal, siempre que el prompt esté configurado para mostrar la información de Git.

5. ¿Quién creó Git?

Git fue creado por Linus Torvalds en 2005, el mismo desarrollador del kernel de Linux. Antes de Git, el código del kernel de Linux se gestionaba con un sistema de control de versiones llamado BitKeeper, pero debido a problemas de licencia, la comunidad necesitaba una alternativa. Linus Torvalds diseñó Git para ser rápido, eficiente y distribuido, convirtiéndose rápidamente en el estándar para la gestión de código en el desarrollo de software.

6. ¿Cuáles son los comandos esenciales de Git?

Los comandos más importantes en Git incluyen:

- **Inicializar un repositorio (git init)**: Convierte una carpeta en un repositorio Git.
- **Clonar un repositorio (git clone)**: Copia un repositorio remoto en tu máquina local.
- **Agregar archivos al área de preparación (git add)**: Indica qué archivos serán incluidos en el próximo commit.
- **Hacer un commit (git commit -m "mensaje")**: Guarda los cambios en el historial del repositorio.
- **Crear una nueva rama (git branch)**: Permite trabajar en una versión alternativa del código.
- **Fusionar ramas (git merge)**: Une los cambios de una rama con otra.
- **Subir cambios a un repositorio remoto (git push)**: Envía los commits al servidor donde está alojado el repositorio.

7. ¿Qué es Git Flow?

Git Flow es una metodología para organizar el trabajo con Git mediante el uso de ramas bien definidas para cada etapa del desarrollo. Su objetivo es mantener un flujo de trabajo estructurado y facilitar la colaboración en equipo.

Se basa en el uso de las siguientes ramas:

- **main**: Contiene el código listo para producción.
- **develop**: Rama de integración donde se combinan los cambios antes de lanzarlos a producción.
- **feature/***: Ramas para el desarrollo de nuevas funcionalidades. Se crean desde develop y se fusionan de vuelta cuando están listas.
  release/*: Ramas para preparar versiones. Se crean desde develop y, cuando están listas, se fusionan en main y develop.
- **hotfix/***: Ramas para corregir errores críticos en producción. Se crean desde main y se fusionan de vuelta en main y develop.

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

El Trunk Based Development es una estrategia en la que todos los desarrolladores integran sus cambios directamente en la rama principal del repositorio (trunk o main). A diferencia de Git Flow, donde las ramas de características pueden existir durante semanas o meses, en Trunk Based Development las ramas son pequeñas y temporales. Se fusionan rápidamente en main, evitando divergencias largas y reduciendo la complejidad de las fusiones.

Las principales ventajas de esta metodología son:

- Facilita la integración continua y entrega continua (CI/CD).
- Evita la acumulación de cambios pendientes que podrían generar conflictos.
- Reduce la cantidad de ramas largas y minimiza los problemas de fusión.
- Permite lanzamientos rápidos y frecuentes de nuevas versiones.

Bibliografía

Chacon, S., & Straub, B. (2014). *Pro Git* (2ª ed.). Apress. https://git-scm.com/book/en/v2

Git. (s.f.). *Git documentation*. https://git-scm.com/doc

Atlassian. (s.f.). *Git Flow Workflow*. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Atlassian. (s.f.). *Trunk-Based Development vs. Git Flow*. https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development

NVIDIA. (2022). *Trunk-Based Development vs. Git Flow*. https://developer.nvidia.com/blog/trunk-based-development-vs-git-flow/

Driessen, V. (2010). *A successful Git branching model*. https://nvie.com/posts/a-successful-git-branching-model/