INSTITUTO TECNOLÓGICO DE LAS AMÉRICAS (ITLA)

Materia: Programación III

Maestra: Kelyn Tejada Belliard

Nombre: Johlbert Angel Volquez De Los Angeles

Matrícula: 2021-0152

Carrera: Desarrollo de software

Tema: Tarea 3

Sección: 10

Contenido

Introducción

Git es un instrumento esencial en la creación de software contemporáneo, creado para administrar el código fuente de proyectos de forma eficaz y cooperativa. Mediante un sistema de control de versiones distribuido, Git ofrece a los programadores la posibilidad de monitorear las modificaciones en su código a lo largo del tiempo, trabajar en paralelo sin conflictos y cooperar de forma organizada con otros integrantes del equipo. Desde que Linus Torvalds lo fundó en 2005 para desarrollar el núcleo de Linux, Git ha transformado el modo en que los programadores administran y comparten su labor.

1 ¿Qué es Git?

Git es un sistema para la gestión de versiones que asiste a los programadores en mantener un seguimiento de las modificaciones en su código a través del tiempo. Considera esto como un medio para almacenar versiones de tu labor, similar a cuando redactas un libro y deseas volver a una versión previa si no te agrada lo que has realizado. Git te facilita esta tarea, almacenando "instantáneas" de tu proyecto en distintos instantes. Y no solo es beneficioso para el desarrollo de software, sino que también resulta beneficioso en cualquier proyecto en el que se realicen modificaciones de manera constante.

Otro beneficio significativo de Git es que promueve la colaboración. Si varios programadores colaboran simultáneamente en un proyecto, Git les proporciona su propia réplica del código, y posteriormente pueden fusionar sus modificaciones en una versión central sin generar desorden. Esto previene que se compartan el trabajo entre ellos y garantiza que todo el equipo se mantenga actualizado con las más recientes actualizaciones. Resulta fundamental para proyectos de fuente abierta y equipos ágiles.

Git también proporciona instrumentos para cotejar las modificaciones, consultar el registro de cambios y administrar diversas versiones del código. Esto te permite comprender cómo y por qué se realizaron determinadas modificaciones a lo largo del tiempo, lo que resulta muy beneficioso para preservar y expandir el software.

2 ¿Para qué sirve el comando git init?

El comando git init representa el inicio del uso de Git en tu proyecto. En esencia, le solicita a Git que comience un repositorio en el que se almacenará el historial de tu labor. Es similar a abrir un nuevo cuaderno en el que puedes continuar anotando y conservando todo lo que hagas, sin que nada se pierda.

Al ejecutar git init, Git genera una carpeta oculta denominada.git en tu proyecto, en la que almacena toda la información requerida para seguir el rastreo de las modificaciones. Sin este sitio, Git no podría operar, dado que allí se guarda todo el registro y la configuración del repositorio.

Tras la ejecución de git init, ya puedes comenzar a manejar Git de forma local. Y si alguna vez deseas compartir tu trabajo con otros, puedes vincular tu proyecto a un repositorio remoto, como GitHub, y emplear instrucciones como git push para cargar tus modificaciones. Es el paso inicial para activar todo el sistema de versiones.

3 ¿Qué es una rama?

En Git, una rama representa fundamentalmente un método para realizar modificaciones paralelas sin modificar la "línea principal" del proyecto. Visualiza que estás redactando un libro y optas por indagar en una nueva trama o personajes sin alterar el contenido inicial. Eso es lo que una rama hace: te permite trabajar en algo novedoso sin impactar en lo que ya se ha realizado.

Las ramas resultan muy beneficiosas al colaborar en equipo. Si cada integrante del equipo posee su propia disciplina, puede crear nuevas características o rectificar fallos sin tener que inquietarse porque su labor interrumpa lo que otros han realizado. Una vez finalizados los cambios, estas ramas pueden unirse de forma organizada con la rama principal. Esto posibilita que diversas personas colaboren eficazmente en el proyecto.

Git hace que gestionar ramas sea bastante fácil. Puedes crear nuevas ramas para diferentes tareas, hacer los cambios que necesites y luego fusionarlas con la rama principal cuando estén listas. Las ramas son una herramienta clave para trabajar en equipo de manera organizada.

4 ¿Cómo saber en cuál rama estoy?

Es crucial identificar en qué disciplina te encuentras para prevenir malentendidos. Si trabajas en múltiples ramas simultáneamente, puedes utilizar el comando git branch para visualizar todas las ramas del proyecto y determinar cuál está en funcionamiento en ese instante. Es similar a examinar un mapa para determinar dónde te ubicas.

Otro método para conocerlo es mediante el uso del comando git status. Este te indica la posición actual y también te proporciona información sobre las modificaciones que has realizado desde el último commit. Es una excelente manera de mantenerse al día, particularmente si estás trabajando en diversas disciplinas.

5 ¿Quién creó Git?

Git fue creado por Linus Torvalds en 2005, el mismo que creó Linux. La historia de Git comenzó porque Linus necesitaba un sistema de control de versiones para el desarrollo del kernel de Linux. El sistema que usaban antes no cumplía con sus necesidades, especialmente en términos de velocidad y eficiencia, así que decidió crear uno nuevo. Así nació Git.

Linus diseñó Git para ser rápido y flexible, permitiendo que los desarrolladores trabajen de forma distribuida. Esto significa que cada persona tiene una copia completa del repositorio, lo que les permite trabajar sin necesidad de estar siempre conectados a un servidor central. Esta característica es clave, especialmente cuando la conexión a internet no es estable. El objetivo de Linus era que Git fuera una herramienta que facilitara la colaboración global, y con el tiempo se ha convertido en el estándar para el desarrollo de software.

6 ¿Cuáles son los comandos más esenciales de Git?

Existen diversos comandos de Git que resultan fundamentales para operar de manera eficaz. Uno de los componentes clave es git init, que pone en marcha un repositorio recién creado, y git add, que te facilita la elección de los archivos que deseas incorporar en tu próximo commit. Git add es esencial ya que te proporciona dominio sobre qué modificaciones se conservan en el registro del proyecto.

Tras añadir los archivos con git add, es necesario utilizar git commit para almacenar esos cambios en el historial. Cada commit funciona como una "imagen" del proyecto en ese instante, y resulta crucial redactar un mensaje preciso que explique qué modificaciones se realizaron. Esto facilita que tú y otros programadores comprendan qué se realizó en cada fase del proyecto.

7 ¿Qué es Git Flow?

Git Flow es una táctica de ramificación que estructura la labor en grupos de desarrollo. Vincent Driessen lo propuso y se ha convertido en muy popular. El concepto es que cada clase de transformación posee su propia rama. Por ejemplo, el código está preparado para ser producido en la rama principal (main), mientras que en la rama development se incorporan las nuevas funciones. Si un programador desea incorporar algo novedoso, genera una rama a partir de develop y, al concluir, la reintegra en esa misma rama.

Git Flow contribuye a mantener el código ordenado y facilita la colaboración sin interferir con el trabajo de otros. También promueve hacer pruebas exhaustivas antes de fusionar cualquier cambio en la rama principal, lo que reduce el riesgo de errores.

Este modelo también facilita la gestión de lanzamientos, ya que cada nueva versión del software puede ser etiquetada y revisada en el historial de Git.

8 ¿Qué es Trunk Based Development?

Trunk Based Development es una táctica que se enfoca en enfocarse en una única rama central, en vez de desarrollar numerosas ramas para cada nueva característica o corrección. Los programadores realizan modificaciones directamente en esa rama principal, promoviendo así la cooperación continua y la incorporación regular de modificaciones. Es como si un conjunto de autores estuviera colaborando en un libro en conjunto, todos aportando al mismo texto simultáneamente.

Este método ofrece el beneficio de que simplifica el procedimiento. No debes inquietarte por combinar ramas, lo que podría resultar complicado. Y al incorporar modificaciones de manera constante, se identifican problemas con mayor rapidez, lo cual es óptimo en ambientes ágiles.

No obstante, el Desarrollo Basado en Trunk exige que todos los programadores conserven el código siempre en un estado operativo antes de realizar un commit. En otras palabras, las pruebas y la revisión del código son fundamentales para garantizar que todo opere sin dificultades.

Conclusión

Con la información anterior podemos concluir que Git no solo simplifica la administración del código fuente, sino que también fomenta la cooperación eficaz entre equipos de desarrollo de diversas dimensiones y localizaciones. Con su habilidad para gestionar versiones de forma eficaz y facilitar la ramificación de tareas, Git se ha establecido como el instrumento estándar para proyectos de software, desde aplicaciones individuales hasta sistemas de colaboración de gran envergadura. Estrategias como Git Flow y el Diseño Basado en Trunk ofrecen métodos diferentes para estructurar el trabajo colaborativo, ajustándose a las demandas de diversos proyectos. La utilización correcta de Git, sumada a buenas prácticas de colaboración y administración de ramas, posibilita a los programadores mantener un flujo de trabajo ordenado, disminuir fallos y potenciar la calidad del software.