Nombres: Andy Jesús.

Apellidos: Mota González.

Matrícula: 2022-1969.

Carrera: Desarrollador de Software.

Materia y profesor/a: Programación III (Kelyn tejada).

Tema: Tarea 3 (Práctica Git).

Fecha: 03/04/2024.

índice

1-¿Qué es Git?

Git es un sistema de control de versiones distribuido de código abierto que se utiliza para rastrear los cambios en el código fuente durante el desarrollo de software. Fue creado por Linus Torvalds en 2005 con el propósito de gestionar el desarrollo del kernel de Linux de una manera más eficiente que los sistemas de control de versiones existentes en ese momento. Desde entonces, se ha convertido en uno de los sistemas de control de versiones más populares y ampliamente utilizados en la industria del desarrollo de software.

Principios Básicos de Git:

- **Distribuido:** Git es un sistema de control de versiones distribuido, lo que significa que cada usuario tiene una copia completa del repositorio, incluido todo el historial de cambios. Esto permite que los usuarios trabajen de forma independiente y realicen un seguimiento de los cambios de manera local antes de sincronizarlos con el repositorio remoto.

- **Rápido y Eficiente:** Git está diseñado para ser rápido y eficiente, incluso en proyectos grandes con miles de archivos y cientos de colaboradores. Utiliza algoritmos inteligentes para optimizar la velocidad de las operaciones, lo que lo hace ideal para proyectos de cualquier tamaño.

- **Ramificación y Fusión:** Una de las características más poderosas de Git es su sistema de ramificación y fusión. Esto permite a los desarrolladores crear ramas independientes para trabajar en nuevas características o correcciones de errores sin afectar la rama principal del proyecto. Una vez completadas, estas ramas se pueden fusionar de nuevo en la rama principal de manera limpia y sin problemas.

- **Integridad de los Datos:** Git utiliza una estructura de datos basada en SHA-1 para garantizar la integridad de los archivos y los metadatos en el repositorio. Cada archivo y cada cambio en el repositorio se representa mediante un hash único, lo que permite detectar cualquier corrupción de datos o manipulación no autorizada.

Componentes Principales de Git:

- **Repositorio:** Un repositorio Git es un directorio que contiene todos los archivos y carpetas de un proyecto, junto con toda la información de control de versiones.

- **Árbol de Trabajo (Working Tree):** El árbol de trabajo es el directorio de nivel superior del repositorio donde se editan y manipulan los archivos del proyecto.

- **Área de Preparación (Staging Area):** También conocida como "índice", el área de preparación es un espacio temporal donde se pueden preparar los cambios antes de confirmarlos en el repositorio.

- **Historial de Confirmaciones (Commit History):** El historial de confirmaciones es una lista de todos los cambios realizados en el repositorio a lo largo del tiempo. Cada confirmación está asociada con un mensaje descriptivo y un identificador único.

- **Ramas:** Las ramas son líneas de desarrollo independientes que permiten a los desarrolladores trabajar en funcionalidades específicas sin afectar la rama principal del proyecto.

- **Repositorio Remoto:** Un repositorio remoto es una copia del repositorio Git almacenada en un servidor remoto, como GitHub, GitLab o Bitbucket. Se utiliza para facilitar la colaboración entre múltiples desarrolladores y para realizar copias de seguridad del código fuente.

2-¿Cuál es el propósito del comando git init en Git?

El comando git init es una parte fundamental en la configuración inicial de un proyecto de Git. Su propósito principal es iniciar un nuevo repositorio Git en un directorio existente, lo que significa que convierte ese directorio en un repositorio de Git vacío, listo para gestionar el control de versiones de los archivos en ese directorio.

Funcionalidades del comando git init:

- **Inicializar un Nuevo Repositorio:**
  Al ejecutar el comando git init en un directorio de trabajo, Git crea un nuevo subdirectorio oculto llamado .git. Este directorio contiene todos los metadatos necesarios para el control de versiones de los archivos en ese directorio, incluidas las confirmaciones de cambios, las ramas y la configuración del repositorio.

- **Preparar un Proyecto Existente para Control de Versiones:**
  Si ya tienes un proyecto existente en un directorio y deseas comenzar a utilizar Git para gestionar su control de versiones, puedes utilizar git init para convertir ese directorio en un repositorio de Git. Esto te permite rastrear y controlar los cambios en el código y colaborar con otros desarrolladores de manera más efectiva.

- **Configuración Inicial del Repositorio:**
  Además de inicializar el directorio como un repositorio de Git, git init también realiza una configuración inicial del repositorio, incluyendo la configuración del nombre del usuario y la dirección de correo electrónico del autor de las confirmaciones. Estos detalles se utilizan para identificar al autor de los cambios en el historial de confirmaciones.

- **Preparación para la Etapa de Preparación (Staging):**
  Después de ejecutar git init, puedes comenzar a usar otros comandos de Git para preparar los archivos para la etapa de preparación (staging) y para realizar confirmaciones en el repositorio. Esto te permite realizar un seguimiento de los cambios en los archivos y mantener un historial de versiones completo y organizado de tu proyecto.

3- ¿Qué representa una rama en Git y cómo se utiliza?

En Git, una rama es una línea independiente de desarrollo que permite a los desarrolladores trabajar en funcionalidades específicas o en paralelo con otros cambios sin afectar la rama principal del proyecto, que comúnmente se denomina master o main. Las ramas proporcionan un mecanismo poderoso para organizar y gestionar el trabajo en un proyecto de desarrollo de software. Se pueden crear nuevas ramas con el comando **"git Branch"** y cambiar entre ellas con **"git checkout"**.

Aspectos Clave de las Ramas en Git:

- **Independencia de Desarrollo:**
  Cada rama en Git representa una secuencia independiente de confirmaciones de cambios. Esto significa que los cambios realizados en una rama no afectan directamente a otras ramas, lo que permite a los desarrolladores trabajar en nuevas características, correcciones de errores o experimentos sin interferir con el trabajo en otras partes del proyecto.

- **Creación y Gestión de Ramas:**
  Puedes crear nuevas ramas en Git utilizando el comando git branch. Por ejemplo, git branch nueva-caracteristica crea una nueva rama llamada nueva-caracteristica basada en la rama actual. También puedes cambiar entre ramas existentes utilizando el comando git checkout, por ejemplo, git checkout otra-rama.

- **Fusión de Ramas:**
  Cuando el trabajo en una rama se completa y se desea integrar esos cambios en la rama principal del proyecto, puedes fusionar la rama con la rama principal. Esto se hace utilizando el comando git merge. La fusión combina los cambios de la rama seleccionada con la rama actual, manteniendo un historial de confirmaciones lineal.

- **Resolución de Conflictos:**
  A veces, al fusionar ramas, pueden surgir conflictos si dos ramas han modificado el mismo archivo en líneas conflictivas. En estos casos, Git pausará la fusión y requerirá que resuelvas manualmente los conflictos antes de continuar. Esto asegura que los cambios se fusionen de manera coherente y sin perder información importante.

- **Ramificación para Experimentación:**
  Las ramas también se pueden utilizar para experimentar con nuevas ideas o funcionalidades sin afectar la rama principal del proyecto. Esto permite a los desarrolladores probar diferentes enfoques sin comprometer la estabilidad del código principal y facilita la colaboración en equipo.

Ventajas de Utilizar Ramas en Git:

- **Aislamiento de Cambios:** Las ramas permiten a los desarrolladores trabajar en cambios independientes sin afectar el código en otras partes del proyecto.

- **Facilita la Colaboración:** Varios desarrolladores pueden trabajar en diferentes ramas simultáneamente y fusionar sus cambios de manera ordenada cuando estén listos.

- **Control de Versiones Mejorado:** Las ramas proporcionan un historial de versiones claro y organizado, lo que facilita la revisión y el seguimiento de los cambios en el proyecto a lo largo del tiempo.

4-¿Cómo puedo determinar en qué rama estoy actualmente en Git?

En Git, es importante saber en qué rama estás trabajando en un momento dado, especialmente cuando trabajamos en proyectos con múltiples ramas o colaboramos con otros desarrolladores. Algunas de las formas para determinar en qué rama estamos actualmente en Git son:

- **Comando git branch:**
  El comando “git Branch” muestra una lista de todas las ramas en el repositorio local y resalta la rama actual con un asterisco (*). Puedes ejecutar este comando en cualquier momento para obtener una visión general de todas las ramas y para identificar la rama actual.

- **Comando git status:**
  El comando “git status” también proporciona información sobre el estado del repositorio, incluida la rama actual. Junto con otros detalles como los cambios sin confirmar y los archivos no rastreados, git status muestra la rama actual en la que te encuentras.

5-¿Quién es la persona responsable de la creación de Git y cuándo fue desarrollado?

Linus Torvalds:

Git fue creado por Linus Torvalds, el creador del kernel de Linux, en 2005. La creación de Git fue impulsada por una serie de eventos relacionados con el mantenimiento del kernel de Linux y la necesidad de un sistema de control de versiones eficiente.

Durante la mayor parte del mantenimiento del kernel de Linux (1991-2002), los cambios en el software se realizaban a través de parches y archivos. En 2002, el proyecto del kernel de Linux comenzó a usar un sistema de control de versiones distribuido (DVCS) propietario llamado BitKeeper. Sin embargo, en 2005, la relación entre la comunidad que desarrollaba el kernel de Linux y la compañía que desarrollaba BitKeeper se rompió, y la herramienta dejó de ser ofrecida de manera gratuita.

Esto llevó a la comunidad de desarrollo de Linux, y en particular a Linus Torvalds, a desarrollar su propia herramienta basada en algunas de las lecciones que aprendieron mientras usaban BitKeeper. Algunos de los objetivos del nuevo sistema fueron: velocidad, diseño sencillo, gran soporte para desarrollo no lineal (miles de ramas paralelas), completamente distribuido, y capaz de manejar grandes proyectos (como el kernel de Linux) de manera eficiente (en términos de velocidad y tamaño de los datos).

Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y conservar sus características iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal.

El diseño de Git se basó en BitKeeper y en Monotone. Originalmente fue diseñado como un motor de sistema de control de versiones de bajo nivel sobre el cual otros podrían codificar interfaces frontales, tales como Cogito o StGIT. Desde entonces, el núcleo del proyecto Git se ha convertido en un sistema de control de versiones completo, utilizable directamente.

6-¿Cuáles son algunos de los comandos esenciales de Git y para qué se utilizan?

Git ofrece una amplia variedad de comandos para gestionar el control de versiones de un proyecto de software. Estos son algunos de los comandos esenciales de Git y cómo se utilizan:

- **git init:** Este comando se utiliza para inicializar un nuevo repositorio Git en un directorio existente. Crea un nuevo subdirectorio oculto llamado **".git"** que contiene todos los metadatos necesarios para el control de versiones.

- **git clone:** Clona un repositorio Git existente en un nuevo directorio local. Se utiliza para obtener una copia completa de un repositorio remoto en tu máquina local.

- **git add:** Añade cambios al área de preparación **(staging area)** para ser incluidos en la próxima confirmación. Puedes añadir archivos individuales **("git add archivo")** o todos los archivos modificados **("git add .")**.

- **git commit:** Guarda los cambios confirmados en el repositorio. Cada confirmación debe tener un mensaje descriptivo que explique los cambios realizados.

- **git push:** Envía los cambios locales confirmados al repositorio remoto. Se utiliza para actualizar el repositorio remoto con tus cambios locales.

- **git pull:** Obtiene y fusiona los cambios del repositorio remoto en tu rama local. Esencialmente, realiza **“git fetch”** seguido de **“git merge”** automáticamente.

- **git checkout:** Cambia entre ramas o restaura archivos. Puede utilizarse para cambiar a una rama existente **(“git checkout nombre-de-la-rama”)** o para crear y cambiar a una nueva rama **(“git checkout -b nombre-de-la-rama”)**.

- **git branch:** Muestra una lista de todas las ramas en el repositorio local. También se puede utilizar para crear, eliminar y renombrar ramas.

- **git merge:** Fusiona los cambios de una rama en otra. Se utiliza para integrar cambios de una rama secundaria en la rama principal del proyecto.

- **git status:** Muestra el estado actual del repositorio, incluidos los archivos modificados, añadidos y sin seguimiento, así como la rama actual y otros detalles relevantes.

- **git log:** Muestra un historial detallado de todas las confirmaciones en el repositorio, incluyendo el autor, la fecha y el mensaje de cada confirmación.

7-¿Puedes mencionar algunos de los repositorios de Git más reconocidos y utilizados en la actualidad?

- **FreeCodeCamp:** Es una organización sin fines de lucro y una de las mejores comunidades de código abierto en línea donde puedes aprender a codificar y ayudar a otros.

- **Awesome Python:** Es un repositorio de Python que contiene los frameworks y librerías basadas en Python que se utilizan para un sinfín de cosas.

- **Public APIs:** Es un repositorio que contiene una impresionante lista de APIs de uso gratuito para fines de desarrollo y aprendizaje. Si buscas datos para implementar una aplicación o solución, aquí seguramente encuentras una API que cubra las necesidades del proyecto.

- **Tech Interview Handbook:** Alberga una increíble colección de preguntas que se hacen en una entrevista de trabajo. No se trata solo de preguntas para la entrevista, sino que permite conocer los procedimientos que se siguen durante el proceso de contratación.

- **TensorFlow:** Es un proyecto de computación con gráficos de flujo para machine learning que en menos de dos años tiene ya más de 38,000 forks, 25,000 commits y 1,100 contribuyentes y es usado por grandes empresas como Airbnb, Snapchat o Twitter.

- **Kubernetes:** Es un sistema de código abierto para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores.

- **React Native:** Es un framework de JavaScript que te permite construir aplicaciones móviles nativas.

- **Flutter:** Es un framework de UI de código abierto creado por Google para desarrollar aplicaciones móviles, web y de escritorio desde una única base de código.

- **Build your own X**: Si estás buscando algún lugar para aprender mientras haces proyectos, build your own X es el lugar ideal para ti. Este repositorio contiene varios proyectos relacionados con las ciencias de la computación como la realidad aumentada, el renderizado 3D, las bases de datos, los emuladores, los juegos y muchos otros temas más.

- **GitHub:** Es uno de los repositorios de código más grandes y populares del mundo. Proporciona herramientas para control de versiones y colaboración en proyectos de software.

- **GitLab:** Similar a GitHub, GitLab ofrece un conjunto completo de herramientas para el ciclo de vida del desarrollo de software, incluyendo control de versiones, seguimiento de problemas y CI/CD (integración continua/entrega continua).

Bilbiografias:

- https://tecnoloco.istocks.club/los-10-mejores-sistemas-de-control-de-versiones-para-linux/2021-03-01/
- https://kinsta.com/es/blog/gitlab-vs-github/
- https://leninmhs.com/historia-de-git/
- https://es.wikipedia.org/wiki/Git
- https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Una-breve-historia-de-Git
- https://desarrollowp.com/blog/tutoriales/aprende-git-de-manera-sencilla-trabajando-con-ramas/
- https://learn.microsoft.com/es-es/devops/develop/git/understand-git-history
- https://leojimzdev.com/descubre-que-es-gitlab-y-como-utilizarlo-para-optimizar-tu-desarrollo/
- https://www.buscabiografias.com/biografia/verDetalle/8294/Linus%20Torvalds