Tarea 3 – Cuestionario

Programación III

INSTITUTO TECNOLÓGICO DE LAS AMÉRICAS

Las Americas Institute of Technology

Presentación

Nombre:

Biaysel Minyety Mejía

Matricula

2023-1106

Materia

Programación III

Profesor

Kelyn Tejada Belliard

Sección

2024-C-3-1615-2880-TDS-007 (10)

Fecha

15/11/2024

Parque Cibernético Santo Domingo (PCSD)

Biaysel Minyety Mejía - 20231106
Biayselminyety001@gmail.com

Desarrolla el siguiente Cuestionario

1- Que es Git?

Bueno, **Git**, ese fascinante sistema de control de versiones distribuido, es como un laberinto de posibilidades que permite a un sinfín de desarrolladores, sí, ¡múltiples! colaborar en un proyecto de software. Imagínate, fue creado por el célebre **Linus Torvalds** en el año 2005, un año que marcó el inicio de una era para gestionar el desarrollo del núcleo de Linux, un hito, sin duda.

Git, en su esencia, permite a los usuarios rastrear los cambios en los archivos. Pero no solo eso, ¡oh no! También pueden revertir a esas versiones anteriores que creían olvidadas, y, además, trabajar en diferentes **ramas de un mismo proyecto** de manera simultánea. Su naturaleza, distribuida, significa que cada desarrollador tiene en sus manos una copia completa del repositorio, lo que, en términos simples, **facilita la colaboración y mejora**, sí, ¡mejora! la seguridad de los datos, como una fortaleza digital.

Ahora, cuando se utiliza **Git**, los desarrolladores, esos valientes navegantes del código, pueden realizar cambios en su copia local, en su pequeño mundo del código, y luego, ¡oh sorpresa!, "empujar" esos cambios a un repositorio remoto, donde otros, como sombras en la noche, pueden acceder y colaborar. Esto, además, **permite trabajar sin conexión a internet**, porque, claro, todas las operaciones se pueden realizar localmente, como un secreto guardado entre amigos.

2- Para que funciona el comando Git INIT?

¿Como no podríamos hablar de este código? Es el primero, es el mágico comando **git init**, esa llave maestra que se utiliza para crear un **nuevo repositorio de Git** en un directorio específico, como si de un nuevo mundo se tratara. Al ejecutar este comando, **Git**, en mi opinión, como un alquimista digital, inicializa un nuevo repositorio vacío,

Biaysel Minyety Mejía - 20231106
Biayselminyety001@gmail.com

creando (atención) una subcarpeta oculta, sí, oculta, **llamada .git**, que contiene todos esos archivos necesarios, casi secretos, para el seguimiento de versiones del proyecto.

Es paso es fundamental y yo diría que, hasta crucial, para comenzar a utilizar **Git** en cualquier proyecto, ya que permite al usuario, al intrépido desarrollador (nosotros), empezar a registrar cambios en los archivos, como un cronista del código, y gestionar el vasto historial de versiones. Una vez que se ha dado el primer paso y se ha inicializado el repositorio, se abre un abanico de posibilidades: se pueden añadir archivos, realizar **commits**, y, en general, utilizar todas esas **funcionalidades asombrosas que Git ofrece** para el control de versiones, como un vasto océano de opciones esperando ser explorado.

3- Que es una rama?

¿Una rama en el vasto universo de Git? Bueno, una rama, esa **versión independiente del proyecto** (así yo lo veo), es como un sendero que permite a los desarrolladores (nosotros nuevamente), los valientes arquitectos del código, trabajar en diferentes características o, por supuesto, **correcciones de errores**, sin afectar la línea principal de desarrollo (esta es una de sus principales características), esa venerada "rama principal" o "**main**". Las ramas son como **islas en un océano de código** (por lo menos yo lo veo así), permitiendo realizar cambios de manera aislada, lo que, sin duda, facilita la experimentación y el desarrollo paralelo, como un ballet de creatividades.

Cuando se **crea una rama**, se hace una **copia** del estado actual **del proyecto** "es fascinante", como si se congelara un momento en el tiempo, y a partir de ahí, se pueden realizar **modificaciones de codigo** valga la redundancia, como un artista esculpiendo su obra (sin exagerar). Una vez que los cambios en la rama están completos, pulidos y, por supuesto, probados, se pueden fusionar (**merge**) de nuevo en la rama principal, como dos **ríos que se encuentran**. Esto ayuda, oh, a mantener un historial de cambios limpio y

organizado, permitiendo que varias personas, como un coro sincronizado, trabajen en el mismo proyecto simultáneamente, sin interferencias, como un reloj bien engrasado.

4- ¿Cómo saber en cual rama estoy?

¿Cómo saber, te preguntas, en qué rama te encuentras dentro del vasto laberinto de un **repositorio de Git**? Para desentrañar este misterio, puedes utilizar el poderoso comando **git branch**. Al ejecutar este encantador comando, **Git desplegará** ante ti una **lista de todas las ramas** que habitan en el repositorio, y, atención, la rama en la que te encuentras actualmente estará marcada con un asterisco (*****), como una estrella brillante al lado de su nombre.

Pero eso no es todo, oh no. Si quieres obtener información más detallada, puedes recurrir al comando **git status**, que, además de ofrecerte un **panorama general**, también indica la rama activa en la parte superior del resultado, como un faro iluminando tu camino. Conocer **la rama activa** es esencial, vital, para asegurarte de que estás trabajando en la parte correcta del proyecto, antes de dar ese paso crucial de realizar cambios o **commits**, como un navegante que verifica su rumbo antes de zarpar.

5- Quien creo Git?

Git, ese sistema que es similar a un **árbol con ramas**, fue creado por el célebre **Linus Torvalds**, si el mismo, en el año 2005. Linus, conocido principalmente por ser el **genio detrás del núcleo de Linux** (ojo con este detalle), desarrolló Git como una respuesta audaz a la creciente necesidad de un **sistema de control de versiones** que pudiera gestionar el torrente de contribuciones al proyecto Linux (y baya que resulto). Antes de la llegada de Git, otros sistemas de control de versiones **intentaban hacer su trabajo** (resumen, no resultaron), pero Linus, con su visión brillante, buscaba una **herramienta que fuera rápida, eficiente, y que permitiera un desarrollo verdaderamente distribuido**, como un río fluyendo libremente.

6- Cuáles son los comandos más esenciales de Git?

1. **git init:** Inicializa un nuevo repositorio de Git.
2. **git clone:** Crea una copia local de un repositorio remoto.
3. **git add:** Añade archivos al área de preparación (staging area) para ser confirmados (committed).
4. **git commit:** Guarda los cambios en el repositorio con un mensaje descriptivo.
5. **git status:** Muestra el estado actual del repositorio, incluyendo archivos modificados y en el área de preparación.
6. **git branch:** Lista las ramas en el repositorio y muestra la rama activa.
7. **git checkout:** Cambia entre ramas o restaura archivos de una versión anterior.
8. **git merge:** Combina los cambios de una rama en otra.
9. **git pull:** Actualiza la rama local con los cambios del repositorio remoto.
10. **git push:** Envía los cambios confirmados a un repositorio remoto.

7- Que es Git Flow?

Git Flow es un modelo de ramificación (**branching**) propuesto por **Vincent Driessen** que proporciona una estructura organizada para gestionar el **desarrollo de software utilizando Git**. Este modelo se basa en el uso de ramas específicas para diferentes propósitos y ciclos de vida del proyecto.

En Git Flow, se utilizan las siguientes ramas principales:

1. **main:** Contiene la versión estable del proyecto.
2. **develop:** Es la rama principal para el desarrollo en curso. Aquí se integran todas las nuevas características y mejoras antes de ser lanzadas.
3. **feature:** Ramas temporales creadas a partir de develop para desarrollar nuevas características. Se fusionan de nuevo en develop una vez completadas.
4. **release:** Ramas creadas para preparar una nueva versión del software. Permiten realizar pruebas y correcciones antes de fusionar los cambios en main.

5. **hotfix:** Ramas creadas a partir de main para realizar correcciones urgentes en producción. Después de aplicar la corrección, se fusionan tanto en main como en develop.

8- Que es trunk based development?

¿Qué es, te preguntas, el Trunk Based Development (TBD)? Esta práctica fascinante de desarrollo de software se basa en la idea de que todos los desarrolladores trabajan en una única rama principal, esa venerada "trunk" o "main". En este enfoque dinámico, los cambios se integran de manera frecuente y continua, como un río que fluye sin cesar, fomentando una colaboración más estrecha y reduciendo, casi eliminando, el riesgo de conflictos de fusión.

Las características clave del Trunk Based Development son verdaderamente notables:

- Integración continua: Los desarrolladores realizan **commits pequeños y frecuentes** en la rama principal, lo que **minimiza la acumulación de código sin fusionar,** como si cada gota de agua se uniera al caudal sin formar estancamientos.

- Ramas temporales: Aunque se permite trabajar en ramas temporales, estas deben ser **de corta duración, como destellos fugaces, y fusionarse** rápidamente en la rama principal, asegurando que el flujo de trabajo sea ágil y eficiente.

- Despliegue frecuente: Al mantener el **trunk** siempre actualizado, se **facilita la implementación continua de cambios en producción**, lo que permite una entrega más rápida de nuevas funcionalidades y correcciones, como un tren que avanza sin detenerse hacia el futuro.