Instituto Tecnológico de las Américas

(ITLA)

Nombre: Randy Emilio Ramírez Aybar

Matricula: 2019-8888

Materia: Programación III

Tema: Asignación Individual Github

Maestro: Kelyn Tejada

Desarrolla el siguiente Cuestionario

1-Que es Git?

Git es un sistema de control de versiones distribuido, diseñado para gestionar proyectos de software de manera eficiente y colaborativa. Fue creado por Linus Torvalds en 2005 y se ha convertido en una herramienta fundamental para el desarrollo de software.

El propósito principal de Git es llevar un registro de los cambios en el código fuente durante el desarrollo de un proyecto. Permite a los desarrolladores colaborar de manera efectiva, realizar un seguimiento de las modificaciones, revertir a versiones anteriores del código, y trabajar en paralelo en diferentes ramas de desarrollo.

2-Para que funciona el comando Git init?

El comando git init se utiliza para inicializar un nuevo repositorio Git. Cuando ejecutas git init en un directorio, Git crea un nuevo repositorio en esa ubicación y comienza a realizar un seguimiento de los cambios en los archivos de ese directorio.

Aquí hay algunas cosas clave que hace git init:

Crea un Repositorio Git Vacío: El comando git init establece un nuevo repositorio Git en el directorio actual o en el directorio especificado como argumento.

git init

Crea el Subdirectorio. git: Git almacena su información y metadatos internos en un subdirectorio oculto llamado .git en el directorio del repositorio. Este subdirectorio contiene la estructura interna del repositorio, incluidas las configuraciones, la base de datos de objetos y referencias, entre otras cosas.

Prepara el Seguimiento de Cambios: Después de ejecutar git init, Git está listo para realizar un seguimiento de los cambios en los archivos. Sin embargo, en este punto, aún no está siguiendo ningún archivo específico. Para comenzar a rastrear archivos, debes agregarlos al área de preparación (staging area) usando git add y luego realizar un commit con git commit.

git add nombre_del_archivo

git commit -m "Primer commit"

Establece la Rama Maestra: El primer commit que hagas después de git init se coloca en la rama maestra, que es la rama principal de desarrollo por defecto en Git. A partir de este punto, puedes crear ramas adicionales para trabajar en características o correcciones de errores sin afectar directamente la rama maestra.

3-Que es una rama?

En el contexto de sistemas de control de versiones, como Git, una "rama" (o "branch" en inglés) se refiere a una línea de desarrollo independiente dentro de un repositorio. Cada repositorio de Git tiene al menos una rama, generalmente llamada "main" o "master", que es la rama principal de desarrollo. Las ramas permiten que los desarrolladores trabajen en distintas características, correcciones de errores o experimentos de manera aislada, sin afectar directamente la rama principal.

4. Como saber en que rama estoy?

Para saber en qué rama estás actualmente en un repositorio de Git, puedes utilizar el siguiente comando en la línea de comandos:

git branch

Este comando te mostrará una lista de todas las ramas presentes en tu repositorio, y la rama actual estará resaltada de alguna manera. Generalmente, la rama actual se indica con un asterisco (*) delante de su nombre.

Por ejemplo, si estás en la rama "main", la salida del comando podría ser algo así:

*** main**

feature/nueva-funcionalidad

bugfix/correccion-error

En este caso, el asterisco delante de "main" indica que estás actualmente en esa rama.

Adicionalmente, puedes utilizar el siguiente comando para obtener información más detallada sobre la rama actual y su estado:

git status

Este comando te mostrará el nombre de la rama actual y te dirá si hay cambios pendientes, si estás en una rama con cambios locales que no se han confirmado, etc.

5-Quien creo git?

Git fue creado por Linus Torvalds, un ingeniero de software finlandés conocido por ser el creador del kernel Linux. Linus Torvalds desarrolló Git para gestionar el desarrollo del kernel de Linux y para abordar las limitaciones que encontró en otros sistemas de control de versiones disponibles en ese momento.

El desarrollo de Git comenzó en abril de 2005, cuando Torvalds anunció su intención de crear un nuevo sistema de control de versiones. Tenía la necesidad

de un sistema que fuera rápido, eficiente y que pudiera manejar proyectos grandes, como el desarrollo del kernel de Linux, que implicaba la colaboración de numerosos desarrolladores distribuidos en todo el mundo.

Torvalds diseñó Git con principios clave en mente, como la velocidad, la eficiencia en el manejo de grandes proyectos y la capacidad de manejar operaciones de ramificación y fusión de manera eficiente. A medida que Git se desarrolló y demostró su eficacia en el proyecto de Linux, su popularidad creció rápidamente, y se convirtió en una herramienta esencial en el desarrollo de software en todo tipo de proyectos.

Es importante destacar que aunque Linus Torvalds inició el desarrollo de Git y sigue siendo su figura central, Git es un proyecto de código abierto, y ha recibido contribuciones significativas de la comunidad de desarrolladores en todo el mundo. Git ha evolucionado con el tiempo gracias a la colaboración de numerosos individuos y organizaciones.

6-Cuales son los comandos más esenciales de Git?

Git tiene una amplia gama de comandos que pueden adaptarse a diversas situaciones en el control de versiones y el desarrollo de software. Entre ellos se encuentran:

git init: Inicializa un nuevo repositorio Git.

git clone: Clona un repositorio existente en un nuevo directorio.

git add: Agrega cambios al área de preparación (staging area).

git commit: Guarda los cambios en el repositorio.

git status: Muestra el estado actual de los archivos en el repositorio.

git pull: Obtiene cambios desde un repositorio remoto y los fusiona con el repositorio local.

git push: Envía cambios locales a un repositorio remoto.

git branch: Lista las ramas en el repositorio.

git checkout: Cambia entre ramas o restaura archivos.

git merge: Fusiona cambios de una rama a otra.

git log: Muestra el historial de commits.

git remote: Muestra los repositorios remotos configurados.

7-Que es Git Flow?

Git Flow es un conjunto de extensiones para Git que proporciona un modelo de ramificación y un conjunto de reglas para el manejo de flujos de trabajo en proyectos de desarrollo de software. Fue propuesto por Vincent Driessen y se ha convertido en un enfoque popular para organizar el desarrollo de software con Git, especialmente en proyectos más grandes.

El modelo Git Flow establece una serie de ramas específicas y define cómo y cuándo deben fusionarse para facilitar un flujo de trabajo ordenado y controlado. Algunas de las ramas clave en el modelo Git Flow son:

master: Representa la rama principal del proyecto y contiene solo versiones estables del software.

develop: Es la rama donde se integran todas las características y se preparan para su lanzamiento. No debería contener características no probadas.

feature: Ramas temporales que se crean para el desarrollo de nuevas características. Se fusionan en la rama develop cuando están completas.

release: Ramas que se utilizan para preparar una nueva versión para el lanzamiento. Pueden contener ajustes de último minuto y se fusionan tanto en master como en develop una vez finalizadas.

hotfix: Ramas que se crean para solucionar problemas críticos en producción. Se fusionan tanto en master como en develop una vez completadas.

El uso de Git Flow puede proporcionar una estructura clara para el desarrollo de software, con reglas bien definidas sobre cómo y cuándo se deben realizar las fusiones entre ramas. Sin embargo, es importante señalar que Git Flow puede ser considerado algo pesado para proyectos más pequeños y ágiles. Otros enfoques, como el desarrollo basado en ramas directas (feature branches) sin una estructura formal, también son comunes y pueden adaptarse mejor a ciertos contextos.

8-Que es trunk based development?

El Trunk Based Development (TBD) es un enfoque de desarrollo de software que se centra en mantener una única rama principal (trunk o rama principal) como la fuente de verdad única para el código del proyecto. A diferencia de modelos más complejos de control de versiones, como Git Flow, que utilizan múltiples ramas para el desarrollo de funciones y la preparación de lanzamientos, TBD aboga por mantener la rama principal siempre en un estado de despliegue continuo y listo para producción.

Algunas características clave del Trunk Based Development son:

Rama Principal Única: En un proyecto que sigue el Trunk Based Development, generalmente existe solo una rama principal, a menudo llamada "trunk" o "master". Todo el desarrollo, ya sea para nuevas características, correcciones de errores o mejoras, se realiza directamente en esta rama.

Pequeñas Ramas de Características: En lugar de crear ramas separadas para cada función, los desarrolladores crean ramas de características más pequeñas y las fusionan rápidamente en la rama principal una vez que están completas. Esto promueve una integración continua y ayuda a evitar conflictos importantes entre ramas.

Despliegue Continuo: El objetivo es tener la rama principal siempre en un estado que pueda ser desplegado en producción. Esto significa que las nuevas características y correcciones de errores se integran de manera continua y se prueban de manera exhaustiva, lo que reduce el riesgo de problemas durante el despliegue.

Pruebas Automatizadas: Se da un énfasis significativo a las pruebas automáticas para garantizar la estabilidad y la calidad del código. Las pruebas deben ejecutarse de manera rápida y confiable, permitiendo a los desarrolladores obtener retroalimentación inmediata sobre la validez de sus cambios.

Colaboración Estrecha: El modelo fomenta la colaboración cercana entre los desarrolladores y equipos, ya que todos trabajan en la misma rama principal. Se minimiza la duplicación de esfuerzos y se reduce la complejidad asociada con el manejo de múltiples ramas.

El Trunk Based Development es especialmente eficaz en equipos que buscan una entrega rápida y continua de software, y puede ser particularmente beneficioso en proyectos ágiles y orientados a servicios. Este enfoque requiere una cultura de desarrollo madura, pruebas automatizadas sólidas y una colaboración efectiva entre los miembros del equipo.

2-Desarrolle un ejercicio práctico en Azure Devops o GitHub con las siguientes características

Entregas:

Link de GitHub donde se desarrollen las siguientes actividades:

-Crear un proyecto.

-Utilizar la técnica Git Flow en su proyecto.

-Proyecto funcional.

Link de github:
https://github.com/randyfastcode/CalculadoraWeb/tree/main