PRESENTACION

Nombre:

Franklin Junior Espinal C.

Matricula:

2021-0410.

Materia:

Programación III.

Profesor:

Kelyn Tejada Belliard.

Tema:

Tarea de programacion 3 - Git flow.

1-Desarrolla el siguiente Cuestionario

1- **¿Qué es Git?**

Git es un sistema de control de versiones distribuido, lo que implica que una copia local del proyecto se convierte en un repositorio completo de control de versiones. Estos repositorios locales totalmente operativos facilitan el trabajo sin conexión o a distancia. Los programadores verifican su labor a nivel local y, posteriormente, sincronizan la versión del repositorio con la del servidor. Este modelo se diferencia del control de versiones centralizado, en el que los clientes necesitan sincronizar el código con un servidor previo a la creación de nuevas versiones.

La adaptabilidad y la popularidad de Git lo hacen una elección magnífica para cualquier equipo. Numerosos programadores y egresados universitarios ya conocen el uso de Git. La comunidad de usuarios de Git ha desarrollado herramientas para instruir a los programadores y la popularidad de Git hace más sencillo obtener asistencia cuando es requerida. Aproximadamente todos los ambientes de desarrollo cuentan con compatibilidad con Git y las herramientas de línea de comandos de Git están incorporadas en todos los sistemas operativos más importantes.

2- **¿Para que funciona el comando Git init?**

El proceso git init genera un repositorio nuevo de Git. Es útil para transformar un proyecto ya existente y sin versión en un repositorio de Git, o para poner en marcha un nuevo repositorio desocupado. La mayor parte de los comandos adicionales de Git no están disponibles más allá de un repositorio inicializado, por lo que generalmente este es el primer comando que se ejecuta en un proyecto recién iniciado.

Cuando se ejecuta git init, se establece un subdirectorio de.git en el sitio de trabajo actual, que alberga todos los metadatos de Git requeridos para el repositorio nuevo. Esta información meta abarca subdirectorios de objetos, referencias y archivos de plantilla. Además, se produce un archivo HEAD que se dirige a la confirmación que se ha obtenido actualmente.

Aparte de la carpeta.git, en el directorio principal del proyecto, se mantiene un proyecto ya existente sin alteraciones (en contraposición a SVN, Git no necesita una subcarpeta de.git en cada subdirectorio).

3- **¿Qué es una rama?**

Una rama Git simplemente funciona como un puntero móvil dirigido a una de esas confirmaciones. La rama principal de Git se configura por defecto como la rama principal. Una vez que confirmemos las primeras modificaciones que hagamos, se

establecerá esta rama principal máster en función de dicha confirmación. Cada vez que confirmemos las modificaciones que hagamos, la rama seguirá avanzando de forma automática.

4- **¿Cómo saber en cual rama estoy?**

Es simplemente debido a que ha alterado la variable de entorno PS1 en su terminal para mostrar la rama actual. Para verificar la rama a la que se dirige tu HEAD, puedes redactar el estado de git. Considera que el uso del nombre de la rama, tal como el del videotutorial, puede generar confusión. Si, por ejemplo, te encuentras en la rama principal y en otra consola cambias de rama, al regresar a la primera consola te continuará mostrando la rama principal, no se actualizará. La manera más confiable de identificar la rama en curso es ejecutar git status.

Comando para identificar la rama en curso:

Escribe la dependencia git en la terminal. Este comando te presenta todas las ramas locales y señala con un asterisco (*) la rama donde estás realizando tus tareas.

Alternativa alternativa:

Utiliza la línea de comandos git status. Este comando también muestra la rama en curso junto con otros datos acerca de la condición del repositorio.

5- **¿Quién creo git?**

Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005.

6- **¿Cuáles son los comandos más esenciales de Git?**

git add

Mueve los cambios del directorio de trabajo al área del entorno de ensayo. Así puedes preparar una instantánea antes de confirmar en el historial oficial.

rama de git

Este comando es tu herramienta de administración de ramas de uso general. Permite crear entornos de desarrollo aislados en un solo repositorio.

Git Checkout

Además de extraer las confirmaciones y las revisiones de archivos antiguas, git checkout también sirve para navegar por las ramas existentes. Combinado con los comandos básicos de Git, es una forma de trabajar en una línea de desarrollo concreta.

git clean

Elimina los archivos sin seguimiento de tu directorio de trabajo. Es la contraparte lógica de git reset, que normalmente solo funciona en archivos con seguimiento.

git clone

Crea una copia de un repositorio de Git existente. La clonación es la forma más habitual de que los desarrolladores obtengan una copia de trabajo de un repositorio central.

git commit

Confirma la instantánea preparada en el historial del proyecto. En combinación con git add, define el flujo de trabajo básico de todos los usuarios de Git.

git commit --amend

Pasar la marca --amend a git commit permite modificar la confirmación más reciente. Es muy práctico si olvidas preparar un archivo u omites información importante en el mensaje de confirmación.

git config

Este comando va bien para establecer las opciones de configuración para instalar Git. Normalmente, solo es necesario usarlo inmediatamente después de instalar Git en un nuevo equipo de desarrollo.

git fetch

Con este comando, se descarga una rama de otro repositorio junto con todas sus confirmaciones y archivos asociados. Sin embargo, no intenta integrar nada en el repositorio local. Esto te permite inspeccionar los cambios antes de fusionarlos en tu proyecto.

git init

Inicializa un nuevo repositorio de Git. Si quieres poner un proyecto bajo un control de revisiones, este es el primer comando que debes aprender.

7- ¿Qué es git Flow?

Gitflow es una alternativa para la creación de ramas en Git, donde se emplean ramas de función y diversas ramas principales. Vincent Driessen en noviembre fue el primero en publicarlo y el que lo hizo popular. En contraste con el desarrollo basado en troncos, Gitflow cuenta con múltiples ramas que requieren más tiempo y más confirmaciones. De acuerdo con este modelo, los programadores generan una rama de función y posponen su unión con la rama principal del tronco hasta que la función se encuentra completa. Estas áreas de trabajo de larga duración necesitan más cooperación para la fusión y presentan un mayor riesgo de desvincularse de la rama principal. Además, pueden implementar actualizaciones con conflictos.

Gitflow es apto para proyectos con un ciclo de publicación establecido, además de la práctica sugerida de DevOps de entrega constante. Este proceso laboral no incorpora ningún nuevo concepto o comando, además de los que se requieren para el proceso de trabajo de las ramas de función. Lo que realiza es asignar responsabilidades muy concretas a las diferentes ramas y establecer cómo y cuándo deben interactuar. Además de las ramas de funcionamiento, emplea ramas específicas para la preparación, conservación y registro de publicaciones. Naturalmente, también puedes beneficiarte de todos los beneficios que brinda el flujo de trabajo de ramas de función:

peticiones de inclusión de modificaciones, experimentos independientes y una cooperación más eficiente.

8- ¿Que es trunk based development?

El desarrollo basado en troncos o trunk based development es un método de administración de versiones donde los programadores combinan frecuentemente pequeñas actualizaciones en un "tronco" o rama principal. Se ha vuelto una práctica común entre los equipos de DevOps y un componente del ciclo de vida de DevOps, dado que facilita las etapas de fusión e integración. En realidad, el desarrollo fundamentado en troncos es un ejercicio indispensable de la CI y la CD. Facilita a los programadores la creación de ramas de funciones de breve duración con escasas confirmaciones, en contraposición a otras tácticas de ramas de funciones de larga duración. Conforme la complejidad del código base y la magnitud del equipo se incrementan, el desarrollo basado en troncos contribuye a preservar el equilibrio.