25-7-2023

Programación III

Wander Alexis Bautista Garcia
ITLA

Presentación

Nombre:

Wander Alexis

Apellido:

Bautista Garcia

Materia:

Programación III

Profesor:

Kelyn Tejada

Fecha:

7/25/2023

1-Que es Git?

Git se ha convertido en el estándar mundial para el control de versiones. Entonces, ¿qué es exactamente?

Git es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales plenamente funcionales permiten trabajar sin conexión o de forma remota con facilidad. Los desarrolladores confirman su trabajo localmente y, a continuación, sincronizan su copia del repositorio con la copia en el servidor. Este paradigma es distinto del control de versiones centralizado, donde los clientes deben sincronizar el código con un servidor antes de crear nuevas versiones.

La flexibilidad y popularidad de Git hacen que sea una excelente opción para cualquier equipo. Muchos desarrolladores y graduados universitarios ya saben cómo usar Git. La comunidad de usuarios de Git ha creado recursos para entrenar a desarrolladores y la popularidad de Git facilita la ayuda cuando sea necesario. Casi todos los entornos de desarrollo tienen compatibilidad con Git y las herramientas de línea de comandos de Git implementadas en cada sistema operativo principal.

2-Para que funciona el comando Git init?

El comando git init crea un nuevo repositorio de Git. Puede utilizarse para convertir un proyecto existente y sin versión en un repositorio de Git, o para inicializar un nuevo repositorio vacío. La mayoría de los demás comandos de Git no se encuentran disponibles fuera de un repositorio inicializado, por lo que este suele ser el primer comando que se ejecuta en un proyecto nuevo.

Al ejecutar git init, se crea un subdirectorio de. git en el directorio de trabajo actual, que contiene todos los metadatos de Git necesarios para el nuevo repositorio. Estos metadatos incluyen subdirectorios de objetos, referencias y archivos de plantilla. También se genera un archivo HEAD que apunta a la confirmación actualmente extraída.

Aparte del directorio de. git, en el directorio raíz del proyecto, se conserva un proyecto existente sin modificar (a diferencia de SVN, Git no requiere un subdirectorio de. git en cada subdirectorio).

De manera predeterminada, git init inicializará la configuración de Git en la ruta del subdirectorio de. git. Puedes cambiar y personalizar dicha ruta si quieres que se encuentre en otro sitio. Asimismo, puedes establecer la variable de entorno

$GIT_DIR en una ruta personalizada para que git init inicialice ahí los archivos de configuración de Git. También puedes utilizar el argumento --separate-git-dir para conseguir el mismo resultado. Lo que se suele hacer con un subdirectorio de. git independiente es mantener los archivos ocultos de la configuración del sistema (. bashrc, vimrc, etc.) en el directorio principal, mientras que la carpeta de. git se conserva en otro lugar.

3-Que es una rama?

Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones. La rama por defecto de Git es la rama master. Con la primera confirmación de cambios que realicemos, se creará esta rama principal máster apuntando a dicha confirmación. En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente.

Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones. La rama por defecto de Git es la rama master. Con la primera confirmación de cambios que realicemos, se creará esta rama principal máster apuntando a dicha confirmación. En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente.

4-Como saber es que rama estoy?

Pues mediante un apuntador especial denominado HEAD. Aunque es preciso comentar que este HEAD es totalmente distinto al concepto de HEAD en otros sistemas de control de cambios como subversión o CVS. En Git, es simplemente el apuntador a la rama local en la que tú estés en ese momento, en este caso la rama máster; pues el comando git Branch solamente crea una nueva rama, pero no salta a dicha rama.

5-Quien creo git?

Git fue inventado por el chico famoso, Linus Torvalds, el mismo que creó el corazón de Linux, allá por el 2005. Git es una herramienta muy buena que ayuda a los desarrolladores a trabajar juntos y mantener todo organizado mientras hacen sus programas y aplicaciones. Es como una especie de súper copia de seguridad que siempre guarda el historial de cambios, así si algo sale mal, pueden volver atrás y arreglarlo fácilmente. La verdad es que es muy útil y se ha convertido en algo importante en el mundo del desarrollo de software.

6-Cuales son los comandos más esenciales de Git?

1. `git init`: Es como cuando creas un álbum nuevo en tu colección de fotos, pero en este caso, estás creando un nuevo repositorio Git para guardar tus cambios.

2. `git clone <url>`: Imagina que ves un álbum de fotos que te gusta en línea y le das clic a "descargar". Pues esto es igual, pero en lugar de fotos, estás copiando un repositorio completo a tu computadora para trabajar con él.

3. `git add <archivo>`: Es como si seleccionaras una foto específica para incluirla en tu álbum. Con este comando, seleccionas los cambios que quieres guardar en tu próximo "commit".

4. `git commit -m "Mensaje del commit"`: Cuando terminas de hacer cambios en tus fotos, haces un nuevo "commit". Es como darle un título a tus cambios para que puedas recordar qué hiciste en el futuro.

5. `git status`: Es como revisar tu álbum para ver qué fotos has cambiado o añadido. Con este comando, ves el estado actual de tu repositorio.

6. `git log`: ¡Esto es como revisar el historial de tu álbum! Puedes ver todos los cambios que has hecho, quién hizo cada cambio y cuándo lo hicieron.

7. `git push`: Es como subir tu álbum a una nube para que otros puedan verlo. Con este comando, envías tus cambios a un repositorio en línea.

8. `git pull`: Si alguien más hizo cambios en el álbum en línea, puedes "jalar" esos cambios para tener la versión más actualizada en tu computadora.

9. `git branch`: Imagina que tienes diferentes versiones de tu álbum con diferentes nombres. Cada versión es una "rama". Con este comando, puedes ver todas las ramas que tienes.

10. `git checkout <nombre_rama>`: Cuando quieres trabajar en una versión específica de tu álbum, simplemente "cambias" a esa rama para empezar a hacer tus cambios.

11. `git merge <nombre_rama>`: Si tienes dos versiones diferentes del álbum y quieres combinarlas, haces un "merge". Es como unir dos álbumes en uno solo.

12. `git remote add origin <url>`: Esto es como decirle a tu álbum dónde vivirá en línea. Con este comando, conectas tu repositorio local a un repositorio en línea como GitHub o GitLab.

7-Que es git Flow?

Imagina que estás trabajando en un proyecto de software con un montón de gente. Todos hacen cambios al mismo tiempo, y necesitas asegurarte de que todo funcione bien antes de lanzar una nueva versión del programa.

GitFlow es como un plan para mantener todo organizado. Se basa en la idea de tener dos ramas principales: la rama "master" que es donde está la última versión estable del programa que todos pueden usar, y la rama "develop" donde los desarrolladores juntan todas las cosas nuevas que quieren agregar.

Cuando alguien quiere agregar una nueva función o arreglo, crea su propia rama "feature" (función) para trabajar en ella sin afectar el código principal. Una vez que termina, lo combina de vuelta en la rama "develop".

Cuando llega el momento de lanzar una nueva versión, se crea una rama "release" (lanzamiento) para hacer las últimas pruebas y corregir pequeños problemas antes de ponerlo en "master".

Y si por casualidad hay un error urgente que necesita ser arreglado en la versión actual, se crea una rama "hotfix" (corrección rápida) para resolverlo rápidamente y luego se mezcla de vuelta en "master" y "develop".

Así, con GitFlow, todos pueden trabajar en sus propias cosas sin causar un desastre en el código principal, y cuando es el momento adecuado, las nuevas características y correcciones se integran de manera ordenada y controlada. ¡Es como una receta para una gestión de versiones exitosa en proyectos colaborativos!

8-Que es trunk-based development?

El desarrollo basado en tronco es una práctica de gestión de control de versiones en la que los desarrolladores fusionan pequeñas actualizaciones de forma frecuente en un "tronco" o rama principal (main).

Dado que esta práctica simplifica las fases de fusión e integración, ayuda a lograr la CI/CD (Continuous Integration / Continuous Deployment, siendo estos la Integración Continua y Despliegue Continuo) y, al mismo tiempo, aumenta la entrega de software y el rendimiento de la organización. A medida que los sistemas de control de versiones se desarrollaron, surgieron varios estilos de desarrollo que permitieron a los programadores encontrar errores con más facilidad, crear código en paralelo con sus compañeros y acelerar el ritmo de publicación. Hoy en día, la mayoría de los programadores aprovechan uno de estos dos modelos de desarrollo para ofrecer software de calidad: Git Flow y desarrollo basado en troncos (Trunk Based Development).

https://github.com/Wander2/Pagina-de-Barberia-Alura.git