Integrante:

Engel Yanset Rodriguez Contreras 2022-0849

Maestro:

Kelyn Tejada Belliard

Asignatura:

Programación III

Asunto:

Tarea 3 (Individual)

Git

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 la copia del repositorio con la del 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 lo convierten en 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 los desarrolladores y la popularidad de Git facilita recibir ayuda cuando se necesita. Casi todos los entornos de desarrollo tienen compatibilidad con Git y las herramientas de línea de comandos de Git implementadas en todos los sistemas operativos principales.

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.

¿Qué es una rama?

Las ramas son una de las principales utilidades que disponemos en Git para llevar un mejor control del código. Se trata de una bifurcación del estado del código que crea un nuevo camino de cara a la evolución del código, en paralelo a otras ramas que se puedan generar.

Las ramas nos pueden servir para muchos casos de uso. Por ejemplo, tras la elección del hosting para crear una página web o desarrollar una tienda online, para la creación de una funcionalidad que queramos integrar en un programa y para la cual no queremos que la rama principal se vea afectada. Esta función experimental se puede realizar en una rama independiente, de modo que, aunque tardemos varios días o semanas en terminarla, no afecte a la producción del código que tenemos en la rama principal y que permanecerá estable.

El trabajo con ramas resulta muy cómodo en el desarrollo de proyectos, porque es posible que todas las ramas creadas evolucionen al mismo tiempo, pudiendo el desarrollador pasar de una rama a otra en cualquier momento según las necesidades del proyecto. Si en un momento dado el trabajo con una rama nos ha resultado interesante, útil y se encuentra estable, entonces podemos fusionar ramas para incorporar las modificaciones del proyecto en su rama principal.

En todo proyecto creado con Git podemos ver la rama en la que estamos actualmente situados. Para ello, utilizamos el comando:

Git Branch (Por defecto nos llevará a la rama Main o Master)

¿Cómo saber en qué rama estoy?

En todo proyecto creado con Git podemos ver la rama en la que estamos actualmente situados. Para ello, utilizamos el comando:

Git Branch (Por defecto nos llevará a la rama Main o Master).

También tenemos la posibilidad de ver todas las ramas que tenemos instanciadas con tan sólo escribir:

Git show-branch

Origen de Git

Hasta abril de 2005, Linus Torvalds utilizaba BitKeeper para el control de versiones del desarrollo del kernel de Linux. Tenía un gran número de desarrolladores voluntarios trabajando en el Kernel de Linux y sus contribuciones tenían que ser gestionadas. BitKeeper fue una buena herramienta para gestionar la enorme contribución de los desarrolladores. Los desarrolladores de Linux utilizaron la herramienta de forma gratuita después de un acuerdo entre las dos partes, ya que BitKeeper era un sistema de gestión de control de código fuente propietario, lo que significa que había que pagar por el uso de la herramienta. Surgió un conflicto de intereses después de que Andrew Tridgell creara un cliente de código abierto para acceder al sistema de control de versiones de Bitkeeper mediante ingeniería inversa de los protocolos de BitKeeper. Esto provocó que el titular de los derechos de autor retirara la política de uso gratuito que había acordado anteriormente. Muchos desarrolladores del kernel de Linux renunciaron al acceso a BitKeeper.

Linux sabía que tenía que actuar rápido para reemplazar el sistema de control de versiones que conocía y amaba, por lo que se tomó unas vacaciones de trabajo para decidir qué hacer, ya que los sistemas de control de versiones actuales de uso gratuito no podían resolver sus problemas en ese momento. El resultado de sus vacaciones fue el nacimiento de un nuevo sistema de control de versiones llamado Git.

Tenía algunos objetivos en mente sobre cómo hacer el próximo sistema de control de versiones que pudiera gestionar un gran proyecto como el suyo. Se propuso construir un sistema de control de versiones que fuera completamente opuesto al Sistema de Versiones Concurrentes (CVS), que pudiera soportar un sistema de control de versiones distribuido como BitKeeper y que incluyera salvaguardas muy sólidas contra la corrupción, ya sea accidental o maliciosa. El desarrollo inicial de Git comenzó en 2005 el 3 de abril. El 6 de abril se dio a conocer el proyecto y al día siguiente pasó a ser autohospedado.

Más tarde ese mismo año, Linux Torvalds logró su objetivo de rendimiento después de que se realizara un benchmark y gestionara la versión 2.6.12 del kernel. Desde 2005, el 26 de julio, el mantenimiento fue entregado a Junio Hamano, quien fue uno de los principales contribuyentes al proyecto (responsable de la versión 1.0) y sigue siendo el principal mantenedor del proyecto.

Comandos esenciales en Git

Comando	Descripción
Git init	Este comando inicia un nuevo repositorio de Git dentro de un directorio. Este es el uso básico de git init. Podemos utilizar git init [nombre del proyecto] para inicializar git con un nombre específico.
Git add	Este comando se utiliza para preparar los cambios en los archivos, preparándolos para la siguiente confirmación: git add file1.txt
Git commit	Utilice este comando para crear un mensaje de confirmación para los cambios, haciéndolos parte del historial de su proyecto: git commit -m [agregar nueva característica].
Git status	Este comando muestra información valiosa sobre las modificaciones y el estado de preparación de los archivos: git status.
Git log	El uso básico del registro de git te permite ver una lista cronológica del historial de confirmaciones: git log
Git diff	Este comando le permite comparar los cambios entre su directorio de trabajo y la confirmación más reciente. Por ejemplo, este uso de git diff identifica las diferencias en un archivo específico: Git diff file1.txt o para confirmar entre dos commits: Git diff commit1 commit2.
Git rm	Este comando elimina los archivos del directorio de trabajo y prepara la eliminación para la siguiente confirmación.
Git mv	Utilice este comando para cambiar el nombre y mover archivos dentro de su directorio de trabajo. Este es el comando de Git para cambiar el nombre de un archivo: Git mv file1.txt file2.txt. Git mv file2.txt new_directory/
Git config	Este comando configura varios aspectos de Git, incluida la información y las preferencias del usuario. Por ejemplo, ingrese este comando para establecer su dirección de correo electrónico para confirmaciones: Git config user.email [email@email.com]

GitFlow

Gitflow es un modelo alternativo de bifurcación de Git que implica el uso de ramas de características y varias ramas principales. Fue publicado y popularizado por primera vez por Vincent Driessen en nvie. En comparación con el desarrollo basado en troncos, Gitflow tiene numerosas ramas más longevas y confirmaciones más grandes. En este modelo, los desarrolladores crean una rama de función y retrasan su fusión con la rama troncal principal hasta que se completa la función. Estas ramas de características de larga duración requieren más colaboración para fusionarse y tienen un mayor riesgo de desviarse de la rama troncal. También pueden introducir actualizaciones contradictorias.

Gitflow se puede usar para proyectos que tienen un ciclo de lanzamiento programado y para la práctica recomendada de DevOps de entrega continua. Este flujo de trabajo no agrega ningún concepto o comando nuevo más allá de lo que se requiere para el flujo de trabajo de rama de entidad. En cambio, asigna roles muy específicos a diferentes ramas y define cómo y cuándo deben interactuar. Además de las ramas, utiliza ramas individuales para preparar, mantener y grabar lanzamientos. Por supuesto, también puede aprovechar todos los beneficios del flujo de trabajo de la rama de funciones: solicitudes de incorporación de cambios, experimentos aislados y una colaboración más eficiente.

Trunk Based Development

El desarrollo basado en troncos es una práctica de gestión de control de versiones en la que los desarrolladores fusionan actualizaciones pequeñas y frecuentes en un "tronco" principal o rama principal. Es una práctica común entre los equipos de DevOps y forma parte del ciclo de vida de DevOps, ya que agiliza las fases de fusión e integración. De hecho, el desarrollo basado en troncos es una práctica obligatoria de CI/CD. Los desarrolladores pueden crear ramas de corta duración con algunas confirmaciones pequeñas en comparación con otras estrategias de ramificación de características de larga duración. A medida que crece la complejidad de la base de código y el tamaño del equipo, el desarrollo basado en troncos ayuda a mantener el flujo de versiones de producción.