2023-1376

Profesor:

Kelyn Tejada Belliard

Estudiante:

Fherdy Sánchez Saviñón

Materia:

Programación III

Tema:

Tarea 3 Git y Cuestionario

Contenido

ITLA

2023-1376

¿Qué es Git?

Git es una herramienta que realizar el sistema de control de versiones de código de forma distribuida. Es de código abierto, con mantenimiento activo y la herramienta de este tipo más empleada en el mundo.

Muchos proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Esta herramienta fue creada por Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005

Este sistema funciona muy bien en una amplia variedad de sistemas operativos e IDE (entornos de desarrollo integrados). Git cuenta con una arquitectura distribuida. En lugar de tener un único espacio para todo el historial de versiones del software, propio de los sistemas de control como CVS o Subversion, aquí la copia de trabajo del código de cada desarrollador es un repositorio que puede albergar el historial completo de todos los cambios.

Rendimiento, seguridad y flexibilidad son las bases sobre las que se ha diseñado Git. Es una herramienta muy potente, rápida, ágil y de software libre. Tiene un sistema de trabajo con ramas que lo hace especialmente potente. Estas ramas se destinan a hacer proyectos divergentes de un proyecto principal, para hacer experimentos o para probar nuevas funcionalidades. Cada rama puede tener una línea de progreso diferente de la rama principal donde está el core de nuestro desarrollo. Podemos algunas de esas mejoras o cambios en el código y hacer una fusión a nuestro proyecto principal.

Si vas a usar Git para tus proyectos debes saber que este sistema cuenta con múltiples ventajas:

- Los algoritmos implementados en Git aprovechan el conocimiento sobre los atributos comunes de los auténticos árboles de archivos de código fuente, cómo se modifican y sus patrones de acceso. De ahí que las nuevas versiones de Git sean cada vez más óptimas en cuanto al rendimiento.

- Git no se deja engañar por los nombres de los archivos a la hora de determinar cuál debería ser el almacenamiento y el historial de versiones del árbol de archivos tal como pasa en otros programas. En este sentido lo que hace es centrarse en el contenido del propio archivo.

- El formato de objeto de los archivos del repositorio de Git emplea una combinación de codificación delta y compresión, y guarda explícitamente el contenido de los directorios y los objetos de metadatos de las versiones.

- Usar Git es también garantía de seguridad pues cuenta con un auténtico historial de contenido de tu código fuente.

- Su prioridad básica es conservar la integridad del código fuente gestionado. El contenido de los archivos y las verdaderas relaciones entre estos están protegidos con un algoritmo de hash criptográficamente seguro llamado "SHA1". Así se protege el código y el historial de cambios frente a las modificaciones accidentales y maliciosas, y se garantiza que el historial sea totalmente trazable.

- Otros sistemas de control de versiones carecen de protección contra las modificaciones ocultas posteriores, algo que puede ser una amenaza a la seguridad de la información.

- Git es un sistema especialmente flexible. En la capacidad para varios tipos de flujos de trabajo de desarrollo no lineal, en su eficiencia en proyectos tanto grandes como pequeños y en su compatibilidad con numerosos sistemas y protocolos. Permite la ramificación y el etiquetado como procesos de primera importancia y permite que las operaciones que afectan a las ramas y las etiquetas se almacenen en el historial de cambios, algo que no ofrecen otros sistemas de control de seguimiento.

- Git permite trabajar en equipo de una manera mucho más simple y optima cuando estamos desarrollando software. Podemos controlar todos los cambios que se hacen en nuestra aplicación y en nuestro código y vamos a tener control absoluto de todo lo que pasa en el código.

En definitiva, es una buena opción para la mayoría de los equipos de software actuales. Git es un proyecto de código abierto con más de una década de gestión de gran fiabilidad. Cuenta con el respaldo de la comunidad de programadores y permite un bajo coste para los desarrolladores aficionados, puesto que pueden utilizar Git sin necesidad de pagar ninguna cuota al ser de código abierto.

¿Para qué sirve 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.

¿Qué es una rama en Git?

Una rama o branch es una versión del código del proyecto sobre el que estás trabajando. Estas ramas ayudan a mantener el orden en el control de versiones y manipular el código de forma segura.

En otras palabras, un branch o rama en Git es una rama que proviene de otra. Imagina un árbol, que tiene una rama gruesa, y otra más fina, en la rama más gruesa tenemos los commits principales y en la rama fina tenemos otros commits que pueden ser de hotfix, devlopment entre otros.
Estas son las ramas base de un proyecto en Git:

-Rama main (Master)

Por defecto, el proyecto se crea en una rama llamada Main (anteriormente conocida como Master). Cada vez que añades código y guardas los cambios, estás haciendo un commit, que es añadir el nuevo código a una rama. Esto genera nuevas versiones de esta rama o branch, hasta llegar a la versión actual de la rama Main.

-Rama development

Cuando decides hacer experimentos, puedes generar ramas experimentales (usualmente llamadas development), que están basadas en alguna rama main, pero sobre las cuales puedes hacer cambios a tu gusto sin necesidad de afectar directamente al código principal.

-Rama hotfix

En otros casos, si encuentras un bug o error de código en la rama Main (que afecta al proyecto en producción), tendrás que crear una nueva rama (que usualmente se llaman bug fixing o hot fix) para hacer los arreglos necesarios. Cuando los cambios estén listos, los tendrás que fusionar con la rama Main para que los cambios sean

aplicados. Para esto, se usa un comando llamado Merge, que mezcla los cambios de la rama que originaste a la rama Main.

Todos los commits se aplican sobre una rama. Por defecto, siempre empezamos en la rama Main (pero puedes cambiarle el nombre si no te gusta) y generamos nuevas ramas, a partir de esta, para crear flujos de trabajo independientes.

¿Cómo saber en cuál rama estoy trabajando?

Para saber qué ramas están disponibles y cuál es el nombre de la rama actual, ejecuta git branch.

```
$> git branch

main

another_branch

feature_inprogress_branch

$> git checkout feature_inprogress_branch
```

¿Quién creó Git?

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

¿Cuáles son los comandos esenciales de Git?

git init
El comando git init crea un nuevo repositorio Git o reinicia uno existente.

Cuando ejecutas el comando, tu directorio actual se convierte en un repositorio Git, lo que te permite seguir sus cambios. Añade una carpeta oculta .git en el directorio,

donde Git almacena todos los metadatos, el historial de versiones y los registros relacionados con el proyecto.

clonar git

Para copiar un repositorio Git de una ubicación a otra, utilizamos el comando git clone. Normalmente copia un repositorio existente, incluyendo registros y versiones, desde servidores remotos como GitHub o GitLab a tu máquina local.

estado git
El comando git status nos muestra detalles sobre:

archivos modificados (archivos modificados, pero no puestos en escena).
archivos no rastreados (archivos que Git no está rastreando).
archivos preparados (archivos preparados y listos para ser enviados).

git añadir
El comando git add añade tus cambios al área de preparación. Indica a Git que el repositorio debe actualizar estos cambios una vez que el usuario ejecute el comando confirmar.

git add . : escenifica los cambios realizados en todos los archivos.
git add : incluye sólo los cambios realizados en un archivo concreto del área de preparación.

git commit

Un comando git commit guarda los cambios que has realizado (o puesto en escena) en el repositorio local. Cada vez que ejecutas git commit, Git crea una instantánea de tu repositorio en ese momento. Esto te permite volver a una confirmación anterior siempre que sea necesario.

git push

El comando git push sincroniza tu repositorio remoto con el repositorio local. Una vez que ejecutes este comando, el repositorio remoto reflejará todos los cambios que hayas confirmado localmente.

```
git push <remote> <branch>
```

git pull

El comando git pull recupera y fusiona los cambios del repositorio remoto con los del repositorio local.

El comando git pull combina dos comandos: git fetch y git merge.

En primer lugar, git fetch recupera todos los cambios del repositorio remoto, incluidos los nuevos commits, etiquetas y ramas. A continuación, se ejecuta git merge, integrando esos cambios en la rama local actual.

git recuperar

El comando git fetch te permite revisar los cambios en el repositorio remoto antes de fusionarlos en el local. Descarga los cambios y actualízalos en ramas de seguimiento remotas. Para los que no estén familiarizados, las ramas de seguimiento remoto son copias de ramas de repositorios remotos.

Por ejemplo, el siguiente comando descarga los cambios en el repositorio remoto y los actualiza en ramas de seguimiento remotas bajo origin:

```
git fetch origin
```

git Merge

Si has realizado algún trabajo en una nueva rama, puedes fusionarla con la rama principal utilizando git merge para implementar los cambios. Git realiza la fusión de dos formas:

Fusión rápida: Supongamos que has creado una nueva rama llamada "feature_x" a partir de la rama principal y has trabajado en ella. Si la rama principal no ha tenido ninguna actualización desde que creaste "feature_x", en lugar de confirmar los cambios de "feature_x" en la rama principal, Git actualiza la rama principal para que apunte a la última versión de "feature_x". En este caso, no se crea ninguna nueva confirmación de fusión.
Fusión a tres bandas: Si tanto "feature_x" como la rama principal tienen ediciones, Git combina los cambios y crea una nueva confirmación de fusión en la rama principal.

Ejemplo:

```
C---D---F (feature-branch)
   /
A---B---E---G (main)
```

¿Qué es Git Flow?

Git flow es una estrategia popular de ramificación de Git que simplifica la gestión de lanzamientos. Fue introducida por el desarrollador de software Vincent Driessen en 2010. Básicamente, Git flow consiste en aislar el trabajo en diferentes tipos de ramas de Git. En este artículo, abordaremos las diferentes ramas del flujo de trabajo de Git flow, cómo usarlo en el cliente de GitKraken y analizaremos brevemente otras dos estrategias de ramificación de Git: GitHub flow y GitLab flow.

¿Qué es el desarrollo basado en trunk (Trunk Based Development)?

El desarrollo basado en troncos (TBD) es una estrategia de desarrollo de software en la que los ingenieros integran cambios menores con mayor frecuencia en el código base principal y trabajan a partir de la copia principal en lugar de trabajar en ramas de características de larga duración. Este modelo de desarrollo se suele utilizar como parte de un flujo de trabajo de desarrollo de integración continua.

Con muchos ingenieros trabajando en la misma base de código, es importante contar con una estrategia para el control del código fuente y la colaboración entre los ingenieros. Para evitar anular los cambios de los demás, los ingenieros crean su propia copia del código base, denominadas ramas. Siguiendo una analogía con un árbol, la copia maestra a veces se denomina línea principal o tronco. El proceso de incorporar los cambios de la copia de un individuo al tronco maestro principal se denomina fusión.

El desarrollo basado en troncos adopta un enfoque de entrega más continua para el desarrollo de software, y las ramas son de corta duración y se fusionan con la mayor frecuencia posible. Las ramas son más pequeñas porque suelen contener solo una parte de una característica. Estas ramas de desarrollo de corta duración facilitan el proceso de fusión, ya que hay menos tiempo para la divergencia entre el tronco principal y las copias de las ramas.

Bibliografía

- Atlassian. (s.f.). *Configuración de un repositorio con git init*. Atlassian. https://www.atlassian.com/es/git/tutorials/setting-up-a-repository/git-init

- Atlassian. (s.f.). *Uso de ramas con git checkout*. Atlassian. https://www.atlassian.com/es/git/tutorials/using-branches/git-checkout

- Datacamp. (s.f.). *Comandos básicos de Git*. https://www.datacamp.com/es/blog/git-commands

- GitKraken. (s.f.). *¿Qué es Git Flow?*. https://www.gitkraken.com/learn/git/git-flow

- Optimizely. (s.f.). *Trunk-Based Development*. https://www.optimizely.com/optimization-glossary/trunk-based-development/

- Platzi. (s.f.). *¿Qué es un branch (rama) y cómo funciona un merge en Git?*. https://platzi.com/clases/1557-git-github/19947-que-es-un-branch-rama-y-como-funciona-un-merge-en/

- Tokio School. (s.f.). *¿Qué es Git?* https://www.tokioschool.com/noticias/que-es-git/

- Wikipedia. (s.f.). *Git*. https://es.wikipedia.org/wiki/Git