Git

Git es un software de control de versiones distribuido, de software libre y código abierto, que fué desarrollado por Linus Torvalds, el mismo creador del kernel Linux. De hecho, nació de su propia frustración a la hora de utilizar los sistemas de control de versiones que había hasta el momento, a la hora trabajar con el desarrollo de Linux.

La primera versión de Git se lanzó en 2007, y en esos pocos años, se ha convertido en un estándar y referente absoluto, siendo con una gran diferencia, el sistema de control de versiones más utilizado a nivel mundial, tanto para pequeños desarrollos, como por los proyectos mas grandes e importantes que puedan esicstir a día de hoy, como son el propio kernel Linux.

Ramas

Las ramas en Git son esencialmente punteros que apuntan a commits específicos. El concepto clave aquí es que los commits en Git forman un grafo dirigido acíclico (DAG), donde cada commit (excepto el commit inicial) tiene al menos un commit padre.

Propósitos de las Ramas

- Aislamiento de Trabajo: Las ramas permiten que los desarrolladores trabajen en funciones o correcciones de errores de manera aislada, sin afectar la rama principal de desarrollo.
- Experimentación: Permiten experimentar con nuevas ideas y cambios sin riesgo de romper el código en la rama principal.
- Colaboración: Facilitan la colaboración entre desarrolladores, permitiendo a cada uno trabajar en su propia rama y luego integrar los cambios.

Convenciones de Nombres

- main/master: La rama principal donde el código considerado estable y listo para producción se mantiene.
- develop: A menudo utilizada en flujo de trabajo GitFlow, donde se integra el trabajo desarrollado.
- feature/xxx: Para desarrollar nuevas características.
- bugfix/xxx: Para corregir errores.
- release/xxx: Para preparar nuevas versiones de lanzamiento.
- hotfix/xxx: Para corregir errores críticos en producción.

Merge

El proceso de merge combina la historia de dos ramas diferentes. Git usa una estrategia de tres puntos: el commit de base común más reciente y los dos commits de las ramas a combinar.

Tipos de Estrategias de Merge

- Fast-Forward Merge:

Ocurre cuando no hay commits nuevos en la rama de destino desde que se creó la rama de origen.
Simplemente mueve el puntero de la rama de destino hacia adelante.
No crea un nuevo commit.
Ideal para mantener una historia lineal.

- Recursive Merge:

Utilizado cuando ambas ramas han avanzado independientemente.
Git crea un nuevo commit que tiene dos padres.
Permite una historia más detallada, mostrando claramente las fusiones.

- Ours/Theirs:

Estrategias para resolver conflictos automáticamente, eligiendo siempre los cambios de la rama actual (ours) o de la rama fusionada (theirs).

Consideraciones para Merge

Historia Lineal vs. No Lineal: Algunos equipos prefieren mantener una historia lineal usando rebase en lugar de merge, para evitar commits de merge innecesarios.

Commits de Merge: Proveen un punto claro en la historia donde dos líneas de desarrollo se integraron.

Conflictos

Los conflictos en Git ocurren cuando hay cambios incompatibles en las mismas partes de los mismos archivos en dos ramas que se están fusionando. Git no puede resolver estos cambios automáticamente y necesita intervención manual.

Tipos Comunes de Conflictos

- Conflictos de Contenido: Ocurren cuando el mismo fragmento de un archivo ha sido modificado de manera diferente en ambas ramas.
- Conflictos de Eliminación/Modificación: Ocurren cuando un archivo se elimina en una rama pero se modifica en la otra.
- Conflictos de Adición: Ocurren cuando se añade un archivo con el mismo nombre en diferentes ramas.

Resolución de Conflictos

- Identificación: Git marca los archivos en conflicto y te proporciona indicadores dentro del archivo.
- Edición Manual: Edita el archivo para resolver los conflictos. Puedes elegir qué cambios conservar o combinar partes de ambos.

- Herramientas de Resolución: Utiliza herramientas visuales para facilitar la resolución de conflictos.
- Marcar como Resuelto: Una vez resueltos los conflictos, añade los archivos resueltos y realiza un commit.

Comandos Básicos

- **git init** creará un nuevo repositorio local GIT. El siguiente comando de Git creará un repositorio en el directorio actual:

```
git init
```

- Como alternativa, puedes crear un repositorio dentro de un nuevo directorio especificando el nombre del proyecto:

```
git init [nombre del proyecto]
```

- **git clone** se usa para copiar un repositorio. Si el repositorio está en un servidor remoto, usa:

```
git clone nombredeusuario@host:/path/to/repository
```

- A la inversa, ejecuta el siguiente comando básico para copiar un repositorio local:

```
git clone /path/to/repository
```

- **git add** se usa para agregar archivos al área de preparación. Por ejemplo, el siguiente comando de Git básico indexará el archivo temp.txt:

```
git add <temp.txt>
```

- **git commit** creará una instantánea de los cambios y la guardará en el directorio git.

```
git commit –m “El mensaje que acompaña al commit va aquí”
```

- **git config** puede ser usado para establecer una configuración específica de usuario, como el email, nombre de usuario y tipo de formato, etc. Por ejemplo, el siguiente comando se usa para establecer un email:

```
git config --global user.email tuemail@ejemplo.com
```

- La opción -global le dice a GIT que vas a usar ese correo electrónico para todos los repositorios locales. Si quieres utilizar diferentes correos electrónicos para diferentes repositorios, usa el siguiente comando:

```
git config --local user.email tuemail@ejemplo.com
```

- **git status** muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser preparados o confirmados.

```
git status
```

- **git push** se usa para enviar confirmaciones locales a la rama maestra del repositorio remoto. Aquí está la estructura básica del código:

```
git push  origin <master>
```

- **git checkout** crea ramas y te ayuda a navegar entre ellas. Por ejemplo, el siguiente comando crea una nueva y automáticamente se cambia a ella:

```
command git checkout -b <branch-name>
```

- Para cambiar de una rama a otra, sólo usa:

```
git checkout <branch-name>
```

- Para conectar el repositorio local a un servidor remoto, usa este comando:

```
git remote add origin <host-or-remoteURL>
```

- Por otro lado, el siguiente comando borrará una conexión a un repositorio remoto especificado:

```
git remote <nombre-del-repositorio>
```

- **git branch** se usa para listar, crear o borrar ramas. Por ejemplo, si quieres listar todas las ramas presentes en el repositorio, el comando debería verse así:

```
git branch
```

- Si quieres borrar una rama, usa:

```
 git branch -d <branch-name>
```

- **git pull** fusiona todos los cambios que se han hecho en el repositorio remoto con el directorio de trabajo local.

```
git pull
```

- **git merge** se usa para fusionar una rama con otra rama activa:

```
git merge <branch-name>
```

- **git diff** se usa para hacer una lista de conflictos. Para poder ver conflictos con respecto al archivo base, usa:

```
git diff --base <file-name>
```

- El siguiente comando se usa para ver los conflictos que hay entre ramas antes de fusionarlas:

```
git diff <source-branch> <target-branch>
```

- Para ver una lista de todos los conflictos presentes usa:

```
git diff
```

- **git tag** marca commits específicos. Los desarrolladores lo usan para marcar puntos de lanzamiento como v1.0 y v2.0.

```
git tag 1.1.0 <instert-commitID-here>
```

- **git log** se usa para ver el historial del repositorio listando ciertos detalles de la confirmación. Al ejecutar el comando se obtiene una salida como ésta:

```
commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw
Author: Alex Hunter <alexh@gmail.com>
Date:   Mon Oct 1 12:56:29 2016 -0600
```

- **git reset** sirve para resetear el index y el directorio de trabajo al último estado de confirmación.

```
git reset - -hard HEAD
```

- **git rm** se puede usar para remover archivos del index y del directorio de trabajo.

```
git rm filename.txt
```

- **git stash** guardará momentáneamente los cambios que no están listos para ser confirmados. De esta manera, pudes volver al proyecto más tarde.

```
git stash
```

- **git show** se usa para mostrar información sobre cualquier objeto git.

```
git show
```

- **git fetch** le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no se encuentran en el directorio de trabajo local.

```
git fetch origin
```

- **git rebase** se usa para aplicar ciertos cambios de una rama en otra. Por ejemplo:

```
git rebase master
```