Introducción a git

Git se ha convertido en el estándar mundial para el control de versiones.

Hoy en día, Git es, con diferencia, el sistema de control de versiones moderno más utilizado del mundo. 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. Un asombroso número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Los desarrolladores que han trabajado con Git cuentan con una buena representación en la base de talentos disponibles para el desarrollo de software, y este sistema funciona a la perfección en una amplia variedad de sistemas operativos e IDE (entornos de desarrollo integrados).

Rendimiento

Las características básicas de rendimiento de Git son muy sólidas en comparación con muchas otras alternativas. La confirmación de nuevos cambios, la ramificación, la fusión y la comparación de versiones anteriores se han optimizado en favor del rendimiento. Los algoritmos implementados en Git aprovechan el profundo conocimiento sobre los atributos comunes de los auténticos árboles de archivos de código fuente, cómo suelen modificarse con el paso del tiempo y cuáles son los patrones de acceso.

Seguridad

Git se ha diseñado con la principal prioridad de conservar la integridad del código fuente gestionado. El contenido de los archivos y las verdaderas relaciones entre estos y los directorios, las versiones, las etiquetas y las confirmaciones, todos ellos objetos del repositorio de Git, están protegidos con un algoritmo de hash criptográficamente seguro llamado "SHA1". De este modo, se salvaguarda el código y el historial de cambios frente a las modificaciones accidentales y maliciosas, y se garantiza que el historial sea totalmente trazable.

Con Git, puedes tener la certeza de contar con un auténtico historial de contenido de tu código fuente.

Algunos otros sistemas de control de versiones carecen de protección contra las modificaciones ocultas realizadas con posterioridad, algo que puede suponer una grave vulnerabilidad de seguridad de la información para cualquier organización que se base en el desarrollo de software.

Flexibilidad

Uno de los objetivos clave de Git en cuanto al diseño es la flexibilidad. Git es flexible en varios aspectos: 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.

Git se ha ideado para posibilitar la ramificación y el etiquetado como procesos de primera importancia (a diferencia de SVN) y las operaciones que afectan a las ramas y las etiquetas (como la fusión o la reversión) también se almacenan en el historial de cambios. No todos los sistemas de control de versiones ofrecen este nivel de seguimiento.

Comandos básicos de git

- *Git clone*
  Git clone es un comando para descargarte el código fuente existente desde un repositorio remoto (como Github, por ejemplo). En otras palabras, Git clone básicamente realiza una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en tu ordenador.

  ```
  git clone <https://link-con-nombre-del-repositorio>
  ```

- Git branch
  Las ramas (branch) son altamente importantes en el mundo de Git. Usando ramas, varios desarrolladores pueden trabajar en paralelo en el mismo proyecto simultáneamente. Podemos usar el comando git branch para crearlas, listarlas y eliminarlas.

  ```
  git branch <nombre-de-la-rama>
  ```

- *Git checkout*
  Este es también uno de los comandos más utilizados en Git. Para trabajar en una rama, primero tienes que cambiarte a ella. Usaremos git checkout principalmente para cambiarte de una rama a otra. También lo podemos usar para chequear archivos y commits.

  ```
  git checkout <nombre-de-la-rama>
  ```

- *Git status*
  El comando de git status nos da toda la información necesaria sobre la rama actual.

  ```
  git status
  ```

  podemos encontrar información como:

  - Si la rama actual esta modificada
  - Si hay algo para confirmar, enviar o recibir (pull)
  - Si hay archivos creados, modificados o eliminados

- *Git add*
  Cuando creamos, modificamos o eliminamos un archivo, estos cambios suceden en local y no se incluirán en el siguiente commit (a menos que cambiemos la configuración).

  Necesitamos usar el comando git add para incluir los cambios del o de los archivos en tu siguiente commit.

```
git add <archivo>
```

- *Git commit*
  Este sea quizás el comando más utilizado de Git. Una vez que se llega a cierto punto en el desarrollo, queremos guardar nuestros cambios (quizás después de una tarea o asunto específico).

  Git commit es como establecer un punto de control en el proceso de desarrollo al cual puedes volver más tarde si es necesario.

  También necesitamos escribir un mensaje corto para explicar qué hemos desarrollado o modificado en el código fuente.

```
git commit -m "mensaje de confirmación"
```

- *Git push*
  Después de haber confirmado tus cambios, el siguiente paso que quieres dar es enviar tus cambios al servidor remoto. Git push envía tus commits al repositorio remoto.

```
git push <nombre-remoto> <nombre-de-tu-rama>
```

  De todas formas, si tu rama ha sido creada recientemente, puede que tengas que cargar y subir tu rama con el siguiente comando:

```
git push --set-upstream <nombre-remoto> <nombre-de-tu-rama>
```

Trabajando con repositorios en Github (Branches, Merge, Conflicts)

Git Branch

El uso de las ramas de desarrollo de Git es una excelente manera de trabajar con una aplicación mientras rastreas sus versiones. En general, una rama de desarrollo ("Git Branch") es una bifurcación del estado del código que crea un nuevo camino para la evolución del mismo. Puede ir en paralelo a otras Git Branch que se pueden generar.

En cualquier proyecto Git pudes ver todas las ramas ingresando el siguiente comando en la línea de comando:

```
git branch
```

Si no has creado una rama antes, no verás ningún resultado en el terminal. Crear una rama es realmente simple:

```
git branch [nueva_rama]
```

Luego, tienes que pasar a la rama de desarrollo recién creada. Para hacer esto, ejecuta el siguiente comando:

```
git checkout [nueva_rama]
```

La salida te informará que cambiaste a una nueva rama. Para el ejemplo, digamos que llamamos a esta rama “prueba”, entonces será:

```
Switched to branch ‘prueba’
```

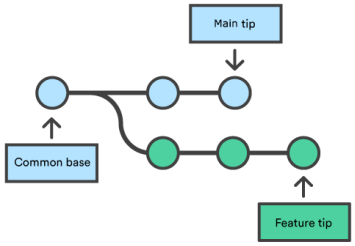
Ahora, en esa nueva rama de desarrollo, puedes crear tantas modificaciones de código como quieras, sin tener que cambiar nada en la principal. Como puedes ver, el programa se mantendrá organizado para nuevas inclusiones de código.

Git merge

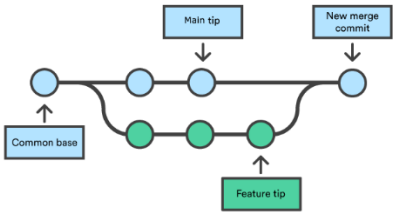
La fusión es la forma que tiene Git de volver a unir un historial bifurcado. El comando git merge permite tomar las líneas independientes de desarrollo creadas por git branch e integrarlas en una sola rama.

git merge combinará varias secuencias de confirmaciones en un historial unificado. En los casos de uso más frecuentes, git merge se utiliza para combinar dos ramas. En estos casos, git merge toma dos punteros de confirmación, normalmente los extremos de la rama, y encuentra una confirmación base común entre ellos. Una vez que Git encuentra una confirmación base en común, crea una "confirmación de fusión" nueva que combina los cambios de cada secuencia de confirmación de fusión puesta en cola.

Supongamos que tenemos una rama de función nueva que se basa en la rama main. Ahora, queremos fusionar esa rama de función con la rama main.



Al invocar este comando, la rama de función especificada se fusionará con la rama actual, la cual asumiremos que es la main. Git determinará el algoritmo de fusión automáticamente (véase más adelante).



Git conflictos

Los conflictos usualmente se crean cuando dos desarrolladores han cambiado las mismas líneas en un archivo, o si un desarrollador eliminó un archivo mientras otro lo estaba modificando. En estos casos, Git no puede determinar automáticamente qué es correcto.

Los conflictos en git solo afectan al desarrollador que realiza la fusión o unión de versiones, el resto del equipo no está al tanto del conflicto. Por lo tanto, Git marcará el archivo como en conflicto y detendrá el proceso de fusión. Entonces es responsabilidad de los desarrolladores dar respuesta al problema del git resolve conflicts.

Git no puede iniciar una fusión

Una fusión no podrá iniciarse cuando Git vea que hay cambios en el directorio de trabajo o en el área de preparación de tu proyecto actual. Como tal, Git no puede iniciar la fusión porque estos cambios pendientes podrían ser sobrescritos por las confirmaciones que estas fusionando. Cuando esto sucede, no se debe a conflictos con otros desarrolladores, sino a conflictos con cambios locales que tengas pendientes. El estado local deberá estabilizarse con git stash, git checkout. Una falla de combinación al inicio generará el siguiente mensaje de error: git commit git reset.

Git falla durante la fusión

Una falla durante una fusión indica un conflicto entre la rama local actual y la rama que estas fusionando. Esto indica un conflicto con el código de otro desarrollador. Git hará todo lo posible para fusionar los archivos, pero dejará las cosas para que las resuelva manualmente en los archivos en conflicto.

FUENTES:

- https://www.atlassian.com/es/git/tutorials/what-is-git
- https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/
- https://www.hostinger.mx/tutoriales/como-usar-git-branch
- https://www.atlassian.com/es/git/tutorials/using-branches/git-merge#:~:text=El%20comando%20git%20merge%20permite,fusionan%20en%20la%20rama%20actual.
- https://keepcoding.io/blog/que-son-conflictos-en-git-y-como-solucionarlos/#:~:text=Los%20conflictos%20usualmente%20se%20crean,determinar%20autom%C3%A1ticamente%20qu%C3%A9%20es%20correcto.