JAIME SAUL VICENTE MALDONADO
CUI: 1922953730101
CARNET: 999004067

Que es GIT

Git es un sistema de control de versiones distribuido (DVCS) que permite registrar los cambios que se hacen en archivos y volver a versiones anteriores si algo sale mal. Fue diseñado por Linus Torvalds para garantizar la eficiencia y confiabilidad del mantenimiento de versiones de aplicaciones que tienen un gran número de archivos de código fuente.

Control de versiones con GIT

El control de versiones, también conocido como "control de código fuente", es la práctica de rastrear y gestionar los cambios en el código de software. Los sistemas de control de versiones (VCS) son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo. A medida que los entornos de desarrollo se aceleran, los sistemas de control de versiones ayudan a los equipos de software a trabajar de forma más rápida e inteligente. Son especialmente útiles para los equipos de DevOps, ya que les ayudan a reducir el tiempo de desarrollo y a aumentar las implementaciones exitosas.

El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir hacia atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error, al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

A continuación, se enlistan los principales motivos por los que GIT es una de las mejores opciones para control de versiones de software:

- GIT es un estándar de facto: Un gran número de desarrolladores ya tienen experiencia con Git y una parte importante de los graduados universitarios puede que solo haya aprendido a usar dicha solución. Aunque algunas organizaciones puedan necesitar escalar la curva de aprendizaje al migrar a Git desde otro sistema de control de versiones, muchos de sus desarrolladores actuales y futuros no precisan de formación para utilizar esta herramienta.

- GIT es un proyecto de código abierto de calidad: Git es un proyecto de código abierto muy bien respaldado con más de una década de gestión de gran fiabilidad. Los encargados de mantener el proyecto han demostrado un criterio equilibrado y un enfoque maduro para satisfacer las necesidades a largo plazo de sus usuarios con publicaciones periódicas que mejoran la facilidad de uso y la funcionalidad. La calidad del software de código abierto resulta sencilla de analizar y un sinnúmero de empresas dependen en gran medida de esa calidad.

El hecho de que sea de código abierto reduce el coste para los desarrolladores aficionados, puesto que pueden utilizar Git sin necesidad de pagar ninguna cuota. En lo que respecta a los proyectos de código abierto, no cabe duda de que Git es el sucesor de las anteriores generaciones de los exitosos sistemas de control de versiones de código abierto, SVN y CVS.

- Criticas de GIT: Una crítica habitual sobre Git es que puede resultar difícil aprender a utilizarlo. Los usuarios nuevos y los procedentes de otros sistemas desconocerán parte de la terminología de Git, ya que esta puede ser diferente; por ejemplo, revert en Git tiene un significado distinto que en SVN o CVS. No obstante, Git es una herramienta muy competente y ofrece multitud de posibilidades a sus usuarios. Aprender a aprovechar estas posibilidades puede llevar un tiempo, pero una vez asimilados los nuevos conocimientos, el equipo puede hacer uso de ellos para acelerar su desarrollo.

Estados de un archivo en GIT

GIT tiene tres estados principales en los que se pueden encontrar los archivos de un proyecto:

- Confirmado: significa que los datos están almacenados de manera segura en la base de datos local.

- Modificado: significa que se ha modificado el archivo, pero todavía no se ha confirmado a la base de datos.

- Preparado: significa que se ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación.

Como se configura un repositorio

Inicio de un nuevo repositorio: git init
Para crear un nuevo repositorio, usa el comando git init. git init es un comando que se utiliza una sola vez durante la configuración inicial de un repositorio nuevo. Al ejecutar este comando, se creará un nuevo subdirectorio .git en tu directorio de trabajo actual. También se creará una nueva rama principal.

git init <project directory>

Clonación de un repositorio existente: git clone
Si un repositorio ya se ha configurado en un repositorio central, el comando de clonación es la manera más común de obtener una copia de desarrollo local. Igual que git init, la clonación suele ser una operación única. Una vez que un desarrollador ha obtenido una copia de trabajo, todas las operaciones de control de versiones se administran por medio de su repositorio local.

git clone <repo url>

El comando git clone se usa para crear una copia o clonar un repositorio remoto. Se utiliza git clone con la URL de un repositorio.

Comandos en GIT

Git clone:

Git clone es un comando para descargar el código fuente existente desde un repositorio remoto (como Github, por ejemplo). En otras palabras, Git clone básicamente hace una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en su computadora.

git clone https://name-of-the-repository-link

Git branch:

Las ramas son muy importantes en el mundo de git. Mediante el uso de ramas, varios desarrolladores pueden trabajar en paralelo en el mismo proyecto simultáneamente. Podemos usar el comando git branch para crear, enumerar y eliminar ramas. Creando una nueva rama:

git branch <branch-name>

Este comando creará una rama localmente. Para insertar la nueva rama en el repositorio remoto, se debe usar el siguiente comando:

git push -u <remote> <branch-name>

Para ver las ramas:

git branch or git branch –list

Para borrar las ramas:

git branch -d <branch-name>

Git checkout:

Este es también uno de los comandos Git más utilizados. Para trabajar en una rama, primero debe cambiarse a ella. Usamos git checkout principalmente para cambiar de una rama a otra. También podemos usarlo para verificar archivos y confirmaciones.

git checkout <name-of-your-branch>

Hay algunos pasos que podemos seguir para cambiar con éxito entre ramas:

- ✓ Los cambios en tu rama actual deben confirmarse o guardarse antes de cambiar.
- ✓ La rama que deseas verificar debe existir en tu local.

También hay un comando de acceso directo que nos permite crear y cambiar a una rama al mismo tiempo:

git checkout -b <name-of-your-branch>

Este comando crea una nueva rama en su local (-b significa rama) y marca la rama como nueva justo después de que se haya creado, si observas bien, el cambio se encuentra en -b).

Git status:
El comando de estado de Git nos brinda toda la información necesaria sobre la rama actual.

git status

Podemos recopilar información acerca de:

- ✓ Si la rama actual está actualizada.
- ✓ Si hay algo que necesita un commit, un add, o borrarse.
- ✓ Si hay archivos preparados, sin preparar o sin seguimiento
- ✓ Si hay archivos creados, modificados o eliminados

```
Cem-MacBook-Pro:my-new-app cem$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory
)

        modified:   src/App.js

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        src/components/
```

Git add
Cuando creamos, modificamos o eliminamos un archivo, estos cambios ocurrirán en nuestro local y no se incluirán en la próxima confirmación (a menos que cambiemos las configuraciones).

Necesitamos usar el comando git add para incluir los cambios de un archivo(s) en nuestro próximo commit.

Para agregar un solo archivo:

git add <file>

Para añadir todo de una vez:

git add –A

Hay nombres de archivos que se encontraban en rojo, lo que significa que son archivos sin preparar. Los archivos no preparados no se incluirán en sus confirmaciones. Para incluirlos, necesitamos usar git add:

```
Cem-MacBook-Pro:my-new-app cem$ git add -A
Cem-MacBook-Pro:my-new-app cem$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   src/App.js
        new file:   src/components/myFirstComponent.js
```

Importante: el comando git add no cambia el repositorio y los cambios no se guardan hasta que usamos git commit.

Git commit

Este es quizás el comando más utilizado de Git. Una vez que llegamos a cierto punto en el desarrollo, queremos guardar nuestros cambios (tal vez después de una tarea o problema específico).

Git commit es como establecer un punto de control en el proceso de desarrollo al que puede volver más tarde si es necesario. También necesitamos escribir un mensaje corto para explicar lo que hemos desarrollado o cambiado en el código fuente.

git commit -m "commit message"

Importante: Git commit guarda tus cambios solo localmente.

Git push

Después de confirmar los cambios (con git commit), lo siguiente que hay que hacer es enviar estos cambios al servidor remoto. Git push sube tus confirmaciones al repositorio remoto.

git push <remote> <branch-name>

Sin embargo, si tu rama se creó recientemente, también debes cargar la rama con el siguiente comando:

git push --set-upstream <remote> <name-of-your-branch>

O bien:

git push -u origin <branch_name>

Importante: Git push solo carga los cambios que están confirmados.

Git pull

El comando git pull se usa para obtener actualizaciones del repositorio remoto. Este comando es una combinación de git fetch y git merge, lo que significa que, cuando usamos git pull,

obtienes las actualizaciones del repositorio remoto (git fetch) e inmediatamente aplica los últimos cambios en su local (git merge). (En simples palabras, sirve para traer el repositorio remoto a tu repositorio local).

git pull <remote>

Esta operación puede causar conflictos que debes resolver manualmente.

Git revert

A veces necesitamos deshacer los cambios que hemos hecho. Hay varias formas de deshacer nuestros cambios de forma local o remota (depende de lo que necesitemos), pero debemos usar estos comandos con cuidado para evitar eliminaciones no deseadas.

Una forma más segura de deshacer nuestras confirmaciones es usando git revert. Para ver nuestro historial de confirmaciones, primero debemos usar git log--oneline:

```
Cem-MacBook-Pro:my-new-app cem$ git log --oneline
3321844 (HEAD -> master) test
e64e7bb Initial commit from Create React App
```

Luego, solo necesitamos especificar el código hash junto a nuestro commit que nos gustaría deshacer:

git revert 3321844

Después de esto, verás una pantalla como la siguiente: simplemente presiona shift + q para salir:

```
Revert "test"

This reverts commit 332184490ef2b5db289d85ed3f1a13ff2d5f94b9.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Committer: Cem <cem@Cem-MacBook-Pro.local>
#
# On branch master
# Changes to be committed:
#       modified:   src/App.js
#       deleted:    src/components/myFirstComponent.js
#
~
~
~
~
~
~
~
~
~
~
"~/Desktop/my-new-app/.git/COMMIT_EDITMSG" 14L, 384C
```

El comando Git revert deshará la confirmación dada, pero creará una nueva confirmación sin eliminar la anterior:

```
Cem-MacBook-Pro:my-new-app cem$ git log --oneline
cd7fe6f (HEAD -> master) Revert "test"
3321844 test
e64e7bb Initial commit from Create React App
```

La ventaja de usar git revert es que no toca el historial de commits. Esto significa que aún puede ver todas las confirmaciones en su historial, incluso las revertidas. Otra medida de seguridad aquí es que todo sucede en nuestro sistema local a menos que los insertemos en el repositorio remoto. Es por eso que git revert es más seguro de usar y es la forma preferida de deshacer nuestros commits.

Git merge

Cuando hayamos completado el desarrollo en nuestra rama y todo funcione bien, el paso final es fusionar la rama con la rama principal (dev o master branch). Esto se hace con el comando git merge.

Git merge básicamente integra su rama de características (feature branch) con todas sus confirmaciones en la rama dev (o master). Es importante recordar que primero debes estar en la rama específica que deseas fusionar con tu rama de características.

Por ejemplo, cuando deseamos fusionar nuestra rama de características en la rama dev:

Primero debemos cambiar a la rama dev:

git checkout dev

Antes de hacer la fusión, debes actualizar la rama de desarrollo local:

git fetch

Finalmente, podemos hacer la fusión:

git merge <branch-name>

Sugerencia: asegúrate que tu rama de desarrollo tenga la última versión antes de fusionar las ramas, de lo contrario, puede enfrentar conflictos u otros problemas no deseados.