Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Estudios de Postgrado
Maestría con especialidad en Telecomunicaciones

NOMBRE: Érito Fernando Tecú Xitumul

¿Qué es GIT?

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).

Git, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés). En lugar de tener un único espacio para todo el historial de versiones del software, como sucede de manera habitual en los sistemas de control de versiones antaño populares, como CVS o Subversion (también conocido como SVN), en Git, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.

¿Qué es un control de versiones, y por qué debería importarte?

Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. Aunque en los ejemplos de este libro usarás archivos de código fuente como aquellos cuya versión está siendo controlada, en realidad puedes hacer lo mismo con casi cualquier tipo de archivo que encuentres en una computadora.

Si eres diseñador gráfico o de web y quieres mantener cada versión de una imagen o diseño (es algo que sin duda vas a querer), usar un sistema de control de versiones (VCS por sus siglas en inglés) es una decisión muy acertada. Dicho sistema te permite regresar a versiones anteriores de tus archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un VCS también significa generalmente que si arruinas o pierdes archivos, será posible recuperarlos fácilmente. Adicionalmente, obtendrás todos estos beneficios a un costo muy bajo.

Estados de un archivo GIT

Al momento de trabajar un Git, nuestros archivos se mueven en 4 estados con diferentes características.

1. Archivos untracked:
   - Archivos que no viven adentro de Git, solo en el disco duro.
   - Nunca han sido afectados por el comando git add
   - Git no tiene registros de su existencia.
1. Archivos unstaged:
   - Archivos que viven dentro de Git
   - No han sido afectados por el comando git add ni git commit
   - Git tiene un registro desactualizado de estos archivos
   - Sus últimas versiones solo están guardadas en el disco duro.
1. Archivos staged:
   - Archivos en staging
   - Viven dentro de Git
   - Hay registros de ellos por que han sido afectados por el comando ** git add** (no, los últimos cambios).
   - Git ya sabe de la existencia de los últimos cambios, pero aún no se han guardado en el repositorio por que falta ejecutar el comando git commint.
1. Archivos traked:
   - Son los archivos que viven dentro de Git
   - No tienen cambios pendientes
   - Sus ultimas actualizaciones se han guardado en el repositorio por medio de comandos git add y git comment.

Nota:

Hay un caso muy raro donde los archivos tienen dos estados al mismo tiempo: staged y untracked.

Esto pasa cuando guardas los cambios de un archivo en el área de Staging (con el comando git add), pero antes de hacer git commit para guardar los cambios en el repositorio, haces nuevos cambios que todavía no han sido guardados en el área de Staging.

COMO CREAR UN REPOSITORIO EN GIT

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.

COMANDOS EN GIT

1. 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.

Hay un par de formas de descargar el código fuente, pero principalmente yo prefiero clonar de la forma con https:

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

2. 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.

Creando una nueva rama:

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

Este comando creará una rama en local. Para enviar (push) la nueva rama al repositorio remoto, necesitarás usar el siguiente comando:

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

Visualización de ramas:

```
git branch
```

```
git branch --list
```

Borrar una rama:

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

3. 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>
```

Hay también un comando de acceso directo que te permite crear y cambiarte a esa rama al mismo tiempo:

```
git checkout -b <nombre-de-tu-rama>
```

4. Git status

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

```
git status
```

5. 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.

Añadir un único archivo:

```
git add <archivo>
```

Añadir todo de una vez:

```
git add -A
```

6. 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"
```

7. 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>
```

or

```
git push -u origin <nombre-de-tu-rama>
```

8. Git pull

El comando git pull se utiliza para recibir actualizaciones del repositorio remoto. Este comando es una combinación del git fetch y del git merge lo cual significa que cundo usemos el git pull recogeremos actualizaciones del repositorio remoto (git fetch) e inmediatamente aplicamos estos últimos cambios en local (git merge).

```
git pull <nombre-remoto>
```