



GitHub Universe Cloud Skills Challenge

Cuaderno de notas sobre el desafío

Elaborado por:

Marco Silva Huerta

01 de Noviembre de 2023

Índice

| | |
|---|----------|
| 1. Introducción a Git | 2 |
| 1.1. Importancia del Control de Versiones | 2 |
| 2. Terminología de Git | 2 |
| 2.1. Configuración de Git | 4 |
| 3. Inicio de un proyecto | 5 |

1. Introducción a Git

1.1. Importancia del Control de Versiones

El control de versiones es esencial en el desarrollo de software, ya que permite gestionar los cambios en el código fuente y la colaboración entre desarrolladores de manera eficiente y segura. Git es un VCS de código abierto rápido, versátil, muy escalable y gratuito. Su autor principal es Linux Torvalds, creador de Linux.

Imaginemos que estamos escribiendo un reporte de practica con nuestro equipo, seguramente a muchos nos paso que todos nos conectábamos al mismo tiempo escribiendo sobre el documento haciendo un mezcla en todo el texto, agregando referencias o imágenes, incluso hasta borrando cosas de los demás. Usando control de versiones, cada modificación se registra, permitiendo ver quién hizo qué y cuándo.

¿Qué ofrece un control de versiones?

- **Historial de Cambios:** Un controlador de versiones mantiene un registro detallado de todos los cambios realizados en el código a lo largo del tiempo, lo que permite rastrear quién hizo qué y cuándo.
- **Ramificación y Fusiones:** Permite la creación de ramas (*branching*) para trabajar en nuevas características o correcciones sin afectar la versión principal. Posteriormente, se pueden fusionar (*mergear*) estas ramas de manera controlada.
- **Recuperación de Versiones Anteriores:** Si surge un problema en una versión de software, es posible volver a una versión anterior para solucionar el problema rápidamente.
- **Copias de Seguridad:** Los controladores de versiones actúan como una copia de seguridad automática del código fuente. En el caso de Git, cada clon del repositorio es una copia completa del historial, lo que garantiza la seguridad y la disponibilidad de los datos.
- **Revisiones y Pruebas Colaborativas:** Los miembros del equipo pueden realizar revisiones de código de manera conjunta, identificar problemas y realizar pruebas en paralelo, acelerando el proceso de desarrollo.

Control de versiones distribuido

Los sistemas de control de versiones centralizados, como CVS, SVN y Perforce, almacenan el historial de un proyecto en un solo servidor. Esto puede ser un problema si el servidor falla o está inaccesible. Git es un sistema de control de versiones distribuido, lo que significa que el historial de un proyecto se almacena en el cliente y en el servidor. Esto hace que Git sea más robusto y fiable que los sistemas centralizados.

2. Terminología de Git

Árbol de trabajo

Conjunto de directorios y archivos anidados que contienen el proyecto en el que se trabaja.

Repositorio (repo):

Un repositorio, también conocido como repo, es un directorio situado en el nivel superior de un árbol de trabajo en Git, donde se almacena todo el historial y los metadatos de un proyecto. Puede haber repositorios vacíos, que no forman parte de un árbol de trabajo y se utilizan para compartir o realizar copias de seguridad. Un repositorio vacío generalmente es un directorio con un nombre que termina en `.git`, por ejemplo, `project.git`.

Inicializar un Repo:

Es el proceso de crear una nueva copia de trabajo de Git y asociarla con un remoto. Este proceso se puede realizar mediante el comando `git init`.

Hash:

Un hash es un número generado por una función hash que representa el contenido de un archivo u otro objeto como un número de dígitos fijo. En Git, se utilizan hashes de 160 bits de longitud. La ventaja de los códigos hash en Git es que permiten verificar si un archivo ha cambiado mediante la comparación de su hash actual con el hash anterior. Si el contenido del archivo no cambia, aunque se modifique la marca de fecha y hora, el hash seguirá siendo el mismo.

Objeto:

Un repositorio de Git contiene cuatro tipos de objetos, cada uno identificado de forma única por un hash SHA-1. Un objeto blob contiene un archivo normal, un objeto árbol representa un directorio y contiene nombres, valores hash y permisos, un objeto de confirmación representa una versión específica del árbol de trabajo, y una etiqueta es un nombre asociado a una confirmación en Git.

Confirmación:

El término *confirmación* puede usarse como un verbo para referirse a la acción de crear un objeto de confirmación en Git. Esto implica guardar los cambios realizados en un proyecto para que otros usuarios puedan acceder a ellos.

Rama:

Una rama en Git es una serie de confirmaciones vinculadas con un nombre. La confirmación más reciente en una rama se conoce como el nivel superior de esa rama. La rama predeterminada, que se crea al inicializar un repositorio, se llama `main` y suele tener el nombre `master` en Git. La rama actual se identifica como `HEAD`. Las ramas son una característica poderosa de Git que permite a los desarrolladores trabajar de forma independiente o colaborativa en diferentes ramas y luego fusionar los cambios en la rama predeterminada.

Remoto:

Un remoto en Git es una referencia con nombre a otro repositorio de Git. Al crear un repositorio, Git generalmente crea un remoto llamado `origin`, que es el remoto predeterminado para las operaciones de envío e incorporación de cambios.

Comandos, subcomandos y opciones:

Las operaciones en Git se realizan mediante comandos, subcomandos y opciones. El comando principal, como `git push` o `git pull`, especifica la operación que se desea realizar. Los comandos suelen ir acompañados de opciones que se utilizan con guiones (-) o guiones dobles (--). Por ejemplo, `git reset --hard` es un comando que implica un subcomando `reset` con la opción `--hard`.

Git y GitHub

Git es un sistema de control de versiones distribuido (DVCS) que varios desarrolladores y otros colaboradores pueden usar para trabajar en un proyecto. Proporciona una manera de trabajar con una o varias ramas locales y luego insertarlas en un repositorio remoto.

GitHub es una plataforma en la nube que usa Git como tecnología principal. Simplifica el proceso de colaboración en proyectos y proporciona un sitio web, más herramientas de línea de comandos y un flujo integral que los desarrolladores y usuarios pueden usar para trabajar juntos. GitHub actúa como el repositorio remoto mencionado anteriormente.

2.1. Configuración de Git

- Para comprobar que Git está instalado, usar el comando `git --version`
- Para configurar Git, se define por variables globales: `user.name` y `user.email`. Ambas necesarias para hacer confirmaciones.

```
git config --global user.name <USER_NAME>
git config --global user.email <USER_EMAIL>
```

- Lista detallada de la configuración de GIT `git config --list`

Configuración del repositorio de Git

1. Cree una carpeta, la cual será el directorio del proyecto y nos movemos a ella

```
mkdir Cats
cd Cats
```

2. Inicializar el repositorio

```
git init -b main
```

3. Usando el comando

```
git status
```

para mostrar el estado del árbol de trabajo

3. Inicio de un proyecto

1. Estando dentro de la carpeta Cats:

```
touch index.html
```

Creación de un archivo

2. Usar `git status` para ver el estado del árbol de trabajo
3. Ahora usar

```
git add .
```

para agregar el nuevo archivo al índice de Git, el punto al final es para agregar todo lo que no tiene seguimiento

4. Volver a usar un `git status`
5. Realización de la primera confirmación

- Utilizar el comando siguiente para crear otra confirmación:

```
git commit -m "Añadiendo archivo index.html"
```

- `git status` Para ver que todo salió bien
- `git log` Para mostrar la información del commit

6. Modifique index.html y confirme el cambio.
7. Usando `code index.html` se abre en VSC el archivo, así mismo si se usa `code .` en una carpeta se abre toda la carpeta
8. Modificamos el archivo
9. Ahora usamos el comando:

```
git commit -a -m "Add a heading to index.html"
```

- No se ha usado `git add`
- Usamos la marca `-a` para agregar los archivos modificados desde la última confirmación. No nuevos

10. Ahora volvemos a modificar el html
11. Usando el comando `git diff` para ver lo que ha cambiado
12. Añadiendo `.gitignore`

13. Con `git add -A` para agregar todos los archivos sin seguimiento

14. Añadiendo subdirectorio

- Vamos a crear una carpeta

```
mkdir CSS
git status
```

- Git no considera directorios vacíos
- El comando `touch` sirve también para actualizar

```
touch CSS/.git-keep
git add . CSS
git status
```

- Ahora si la carpeta se ve en el área de trabajo

15. Remplazo de un archivo

- Eliminar `.git-kepp` del subdirectorio

```
rm CSS/.git-keep
cd CSS
code . site.css
```

Con esto habremos borrado el kepp, cambiado a la carpeta CSS y abierto VSC en el archivo para modificarlo

16. Enumeración de confirmaciones

- Con el comando `git log` se revisan todas las confirmaciones
- Con el comando `git log --oneline` se obtiene una lista más simplificada