git

parte 1: trabajando en local

¿Qué es un sistema de control de versiones?

Un Sistema de Control de Versiones (SCV) es una aplicación que permite gestionar los cambios que se realizan en un proyecto, guardando así versiones del mismo en todas sus fases de desarrollo.

¿Qué aporta un SVC?

- Registra cada cambio en el proyecto, quién y cuándo lo hace.
- Permite volver a estados previos del desarrollo.
- Permite gestionar diferentes versiones del proyecto para trabajar en paralelo y luego fusinarlas.
- Permite colaborar entre diferentes usuarios, facilitando la resolución de conflictos.

¿Qué SVC existen?

- git
- Mercurial
- Subversion
- Perforce (utilizado en desarrollo videojuegos)
- GIT LFS (plugin de git)

¿Por qué usar git?

- Creado en 2005.
- El SVC más extendido.
- Arquitectura distribuida.
- De código abierto.
- Manejo eficiente de ramas.

Comandos git que veremos hoy

- git config
- git init
- git add
- git status
- git commit
- git log

```
git config [opciones] [clave] [valor]
```

El comando `git config` se utiliza para establecer y configurar las opciones de Git:

- Nombre de usuario
- Correo electrónico
- Varias preferencias que controlan el comportamiento de Git.

Este comando es esencial al principio, ya que Git necesita saber quién eres (nombre y correo electrónico) para registrar adecuadamente tus cambios.

```
git config [opciones] [clave] [valor]
```

Ejemplo de uso

Configuración global

Afecta a todos los repositorios

```
git config --global user.name "nombre"
git config --global user.email "email"
```

Configuración local

Afecta al repositorio donde estás trabajando

```
git config user.name "nombre"
git config user.email "email"
```

```
git init
```

Su función principal es crear un **nuevo repositorio de Git** en un directorio, permitiendo que el proyecto en ese directorio sea gestionado por Git.

```
git init
```

¿Qué hace `git init`?

1. **Crea un nuevo repositorio de Git**: Ejecutar `git init` en un directorio (ya sea un directorio vacío o un proyecto existente) inicializa ese directorio como un repositorio de Git.
2. **Directorio oculto** `.git`: Al ejecutarlo, se crea una carpeta oculta llamada `.git` en el directorio. Este es el repositorio donde Git guarda toda la información necesaria para gestionar el historial del proyecto (commits, ramas, etc.).
3. **No afecta los archivos existentes**: Si el directorio ya tiene archivos, `git init` no los modifica; simplemente prepara el directorio para comenzar a rastrear cambios en esos archivos.

```
git init
```

Ejemplo

Crear un nuevo repositorio desde cero

Supongamos que tienes un directorio vacío llamado `mi-proyecto` y quieres iniciar un repositorio de Git en él.

```
mkdir mi-proyecto
cd mi-proyecto
git init
```

>Initialized empty Git repository in /ruta/a/mi-proyecto/.git/

```
ls -lart mi-proyecto
```

>

total 7096

drwxr-xr-x 38 jose staff   1216 23 sep 10:27 ..

drwxr-xr-x 12 jose staff   384 23 sep 10:27 .git

3 comandos en grupo

- → git status
- → git add
- → git commit

```
git status
```

El comando `git status` proporciona información sobre el estado actual de tu repositorio:

- Qué archivos han cambiado
- Cuáles están listos para ser confirmados (committed)
- Cuáles no están siendo rastreados (untracked).

Este comando **no modifica el estado de los archivos ni realiza cambios en el repositorio**, solo te informa sobre su estado actual.

```
git add <archivo_o_directorio>
```

El comando `git add` es utilizado en Git para agregar archivos al **área de preparación** (staging area), que es un paso intermedio antes de confirmar los cambios en el repositorio con un commit.

Es esencial para decirle a Git qué cambios (nuevos archivos, archivos modificados, etc.) deben incluirse en el próximo commit.

```
git add <archivo_o_directorio>
```

¿Qué hace `git add`?

1. **Prepara los archivos para el commit**: `git add` toma los archivos o cambios y los coloca en el área de preparación. Esto no los guarda de forma permanente en el historial del repositorio; solo los prepara para el commit.
2. **No realiza un commit**: `git add` no guarda los cambios de manera definitiva en el historial del proyecto, solo los marca para que sean incluidos en el próximo commit.
3. **Agregar archivos nuevos y cambios en archivos existentes**: Puedes usarlo para agregar archivos que nunca han sido rastreados por Git, así como para actualizar archivos modificados que ya están siendo rastreados.

```
git commit -m "Mensaje descriptivo del commit"
```

Comando `git commit`
Uno de los comandos más importantes en Git, ya que se utiliza para **guardar los cambios en el historial** del repositorio.

Después de agregar archivos al área de preparación con `git add`, el siguiente paso es confirmarlos usando `git commit`. Esto crea un "snapshot" o instantánea de los cambios en tu proyecto, permitiéndote revertir a este punto en el futuro si es necesario.

```
git commit -m "Mensaje descriptivo del commit"
```

¿Qué hace `git commit`?

1. **Confirma los cambios en el área de preparación**: `git commit` toma los archivos que has preparado con `git add` y los guarda en el historial del repositorio, creando un "punto de control" que registra el estado actual de esos archivos.
2. **Crea un mensaje de commit**: Cada commit debe ir acompañado de un mensaje descriptivo que explique qué cambios has realizado. Este mensaje facilita la comprensión del historial del proyecto.
3. **No afecta los archivos que no han sido preparados**: Solo los archivos que se encuentran en el área de preparación (staging area) serán incluidos en el commit. Los archivos no rastreados o modificados que no han sido agregados no serán parte del commit.

ejemplo completo

mkdir mi-proyecto

cd mi-proyecto

git init mi-proyecto

touch README.md

touch hola.txt

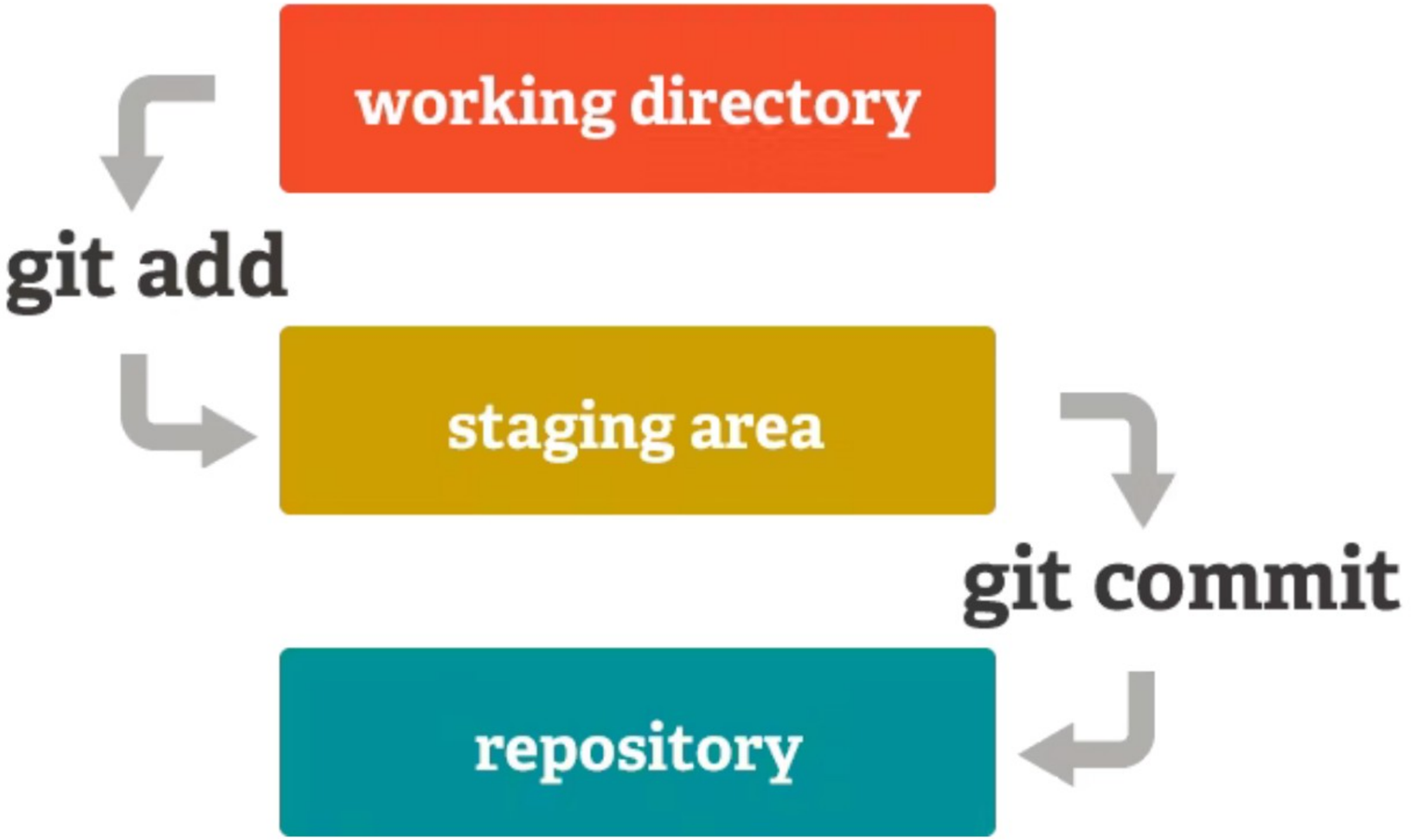
git status

git add REAME.md

git status

git commit -m "Confirmo cambios en el fichero README.md"

git status


working directory
git add
staging area
git commit
repository

```
git log
```

Comando `git log`

Es una herramienta fundamental en Git que te permite ver el historial de commits de un repositorio. Muestra una lista de todos los commits que se han realizado, permitiendo explorar quién hizo los cambios, cuándo se hicieron, qué mensajes de commit se usaron y más.

```
git log
```

¿Qué hace `git log`?

Muestra información detallada sobre cada commit en el historial del repositorio, incluyendo:

1. **SHA-1 hash**: El identificador único del commit (una larga cadena hexadecimal).
2. **Autor**: La persona que hizo el commit.
3. **Fecha**: El momento en que se realizó el commit.
4. **Mensaje del commit**: El mensaje asociado con el commit que describe los cambios realizados.

```
git log
```

ejemplo

Al ejecutar `git log` en tu repositorio, verás algo similar a esto:

```
commit 3f1e948f8721d28f1843f2f (HEAD -> main, origin/main)
Author: Juan<juan@exale.com> Date: Tue Sep 14 15:20:42 2024 +0200
Actualización de los estilos CSS y correcciones menores en el HTML

commit 124c35fe50cfbb5a3c44b8f74d8c4df8799d5df0
Author: María García <maria@example.com>
Date: Mon Sep 13 10:34:08 2024 +0200
Implementación de la nueva funcionalidad de autenticación
```

Mentimeter

¿Dudas?