



[koko**a**]

Taller de GIT.

Sistemas de versionamiento de código

El ponente.

Juan Mite.

CEO en Enjambre desde Enero del 2014, La Libertad Península de Santa Elena.

Fundador y director del departamento de Hardware Libre en Kokoa-Espol, desarrollador de aplicaciones móviles, aplicaciones web y sistemas embebidos para la industria, Researcher de tecnologías libres tanto en hardware, software y aeromodelismo, aficionado a la seguridad informática, inteligencia artificial y a la música.





Premios

GuayasTech:

2do lugar en el reto de Hacking Day (Criptoanálisis).

Ganador del reto de ciudades inteligentes (Participación Ciudadana, sistema de seguimiento de quejas)

MachalaTech:

Ganador del reto Sensores inteligentes.

Campus party 4:


Finalista en reto Ikiam y muros libres para la ciudad de Quito, OWASP 2014 (5to lugar)



Proyectos.

- | Sistema operativo para motos.
- | Cerebro domótico.
- | Bee-Tracker
- | Bee-Sensors
- | SAT para INOCAR
- | Uplay.ec
- | entre otros...

@juanmisak



**¿Qué tienen
en común
todos estos
proyectos?**



¿Por qué están aquí?

EL PROCESO DE TU TESIS



Tesis



Tesis final



Tesis final este
sí



Tesis final este
sí sí sí



Tesis final 2



Tesis final final



Tesis final listo



Tesis final por fin



Tesis final por fin
eso espero



Tesis final por fin
the end



Tesis finalisimo



Tesis ultimo



Tesis ultimo
ahora si



Tesis ultimo de
los ultimos



Tesis ultimo
final ok



Introducción

- | ¿Por qué es importante aprender a manejar un sistema de versionamiento de código?
- | - Crear copias de seguridad y restaurarlas.
- | - Mantenerme al día con los cambios.
- | - Saber que ha pasado, cuándo y quién lo hizo.
- | - Status del proyecto en cualquier instante.
- | - Seguimiento de la productividad.
- | - Evitar tropiezos codeando.

Ejercicio mental, imaginen que...

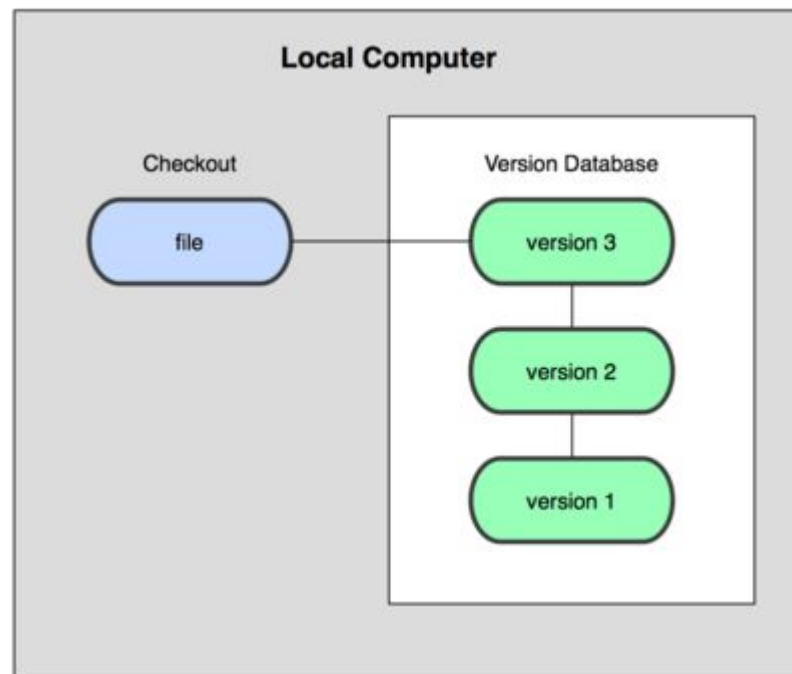


¿Qué nos permite GIT?

- | Realizar pruebas aisladas.
 - | Deshacer cambios (volver a un status anterior).
 - | Forkear un proyecto.
 - | Hacer refactoring fácilmente.
 - | Integrar nuestro proyecto perfectamente con la metodología TDD.
 - | Optimizar y permite el trabajo en equipo de muchas personas
- ...Demos una vuelta por ahí. (Linux, Acat)

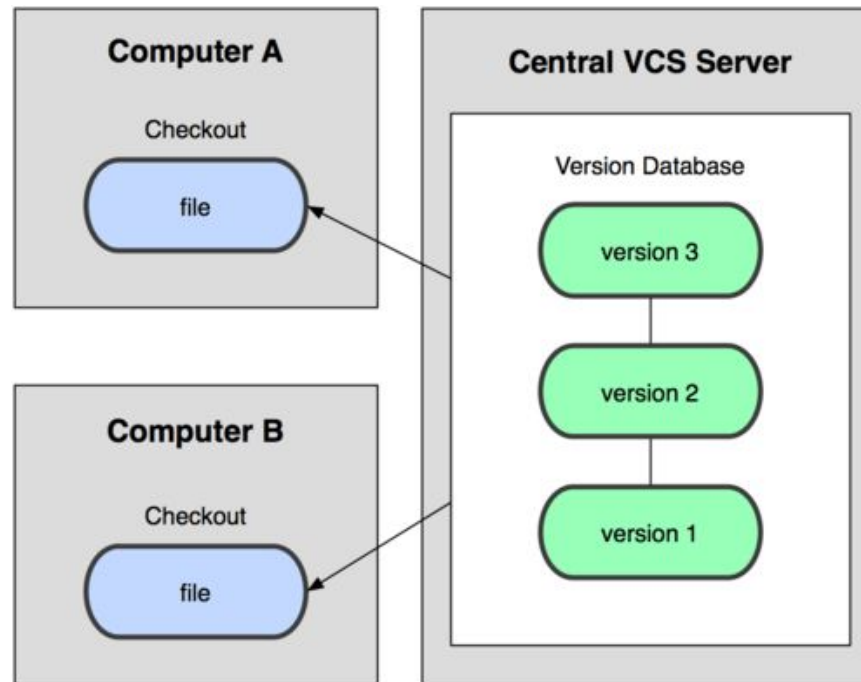
Fundamentos de sistemas de versionamientos.

- | Sistemas de control de versiones locales.
 - RCS.
 - Base de datos local.



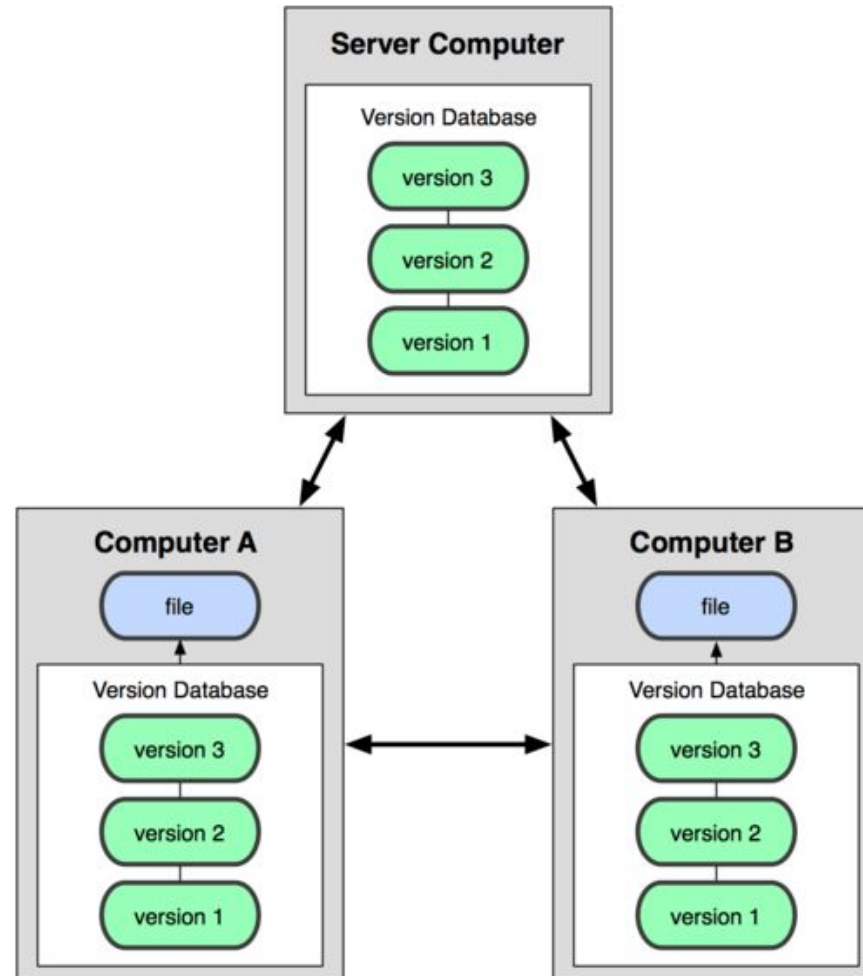
Fundamentos de sistemas de versionamientos.

- | Sistemas de control de versiones centralizados.
 - Te arriesgas a perderlo todo.



Fundamentos de sistemas de versionamientos.

| Sistemas de control de versiones distribuidos



¿Cómo nació git?

- | Linux empezó a usar un DVCS propietario llamado BitKeeper.
- | En 2005, la relación entre la comunidad que desarrollaba el núcleo de Linux y la compañía que desarrollaba BitKeeper se vino abajo.
 - Velocidad
 - Diseño sencillo
 - Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
 - Completamente distribuido
 - Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)



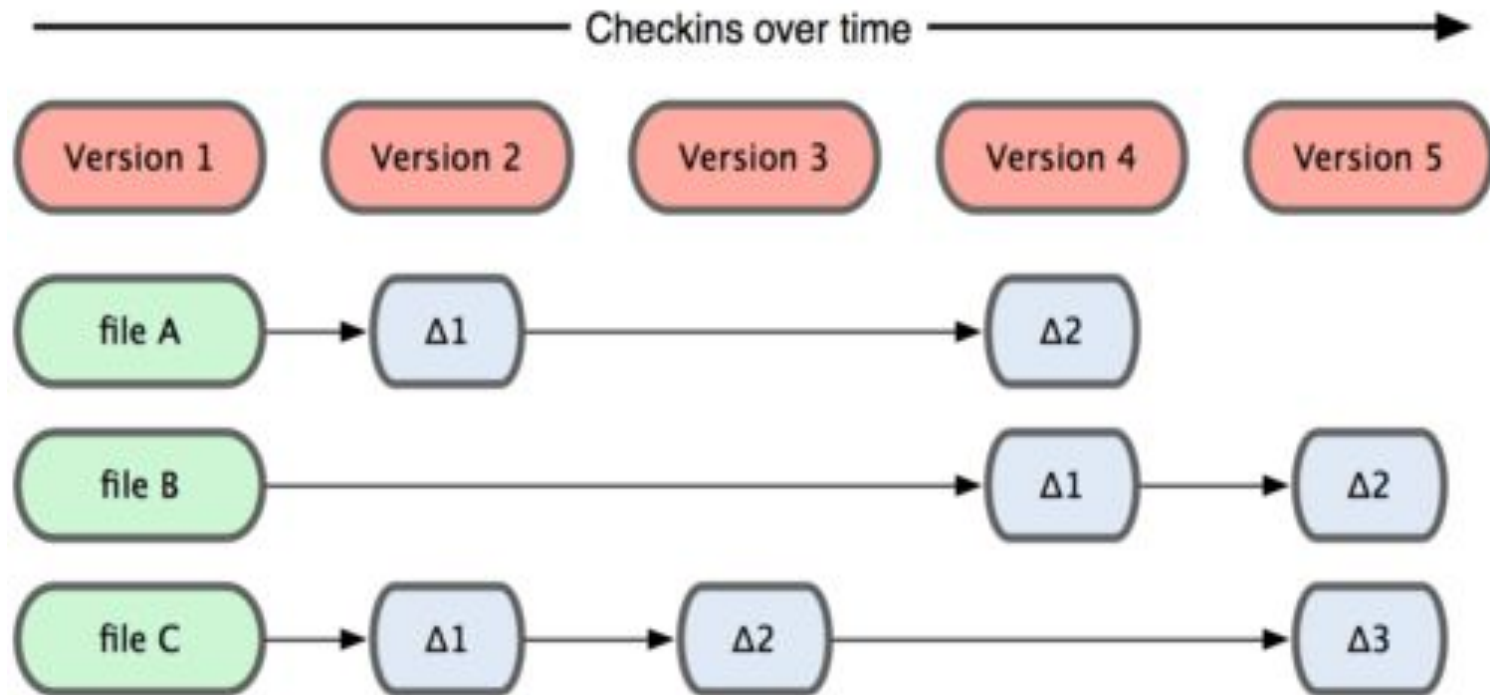
Git
!=
GitHub



**¿Cómo
funciona
git?**

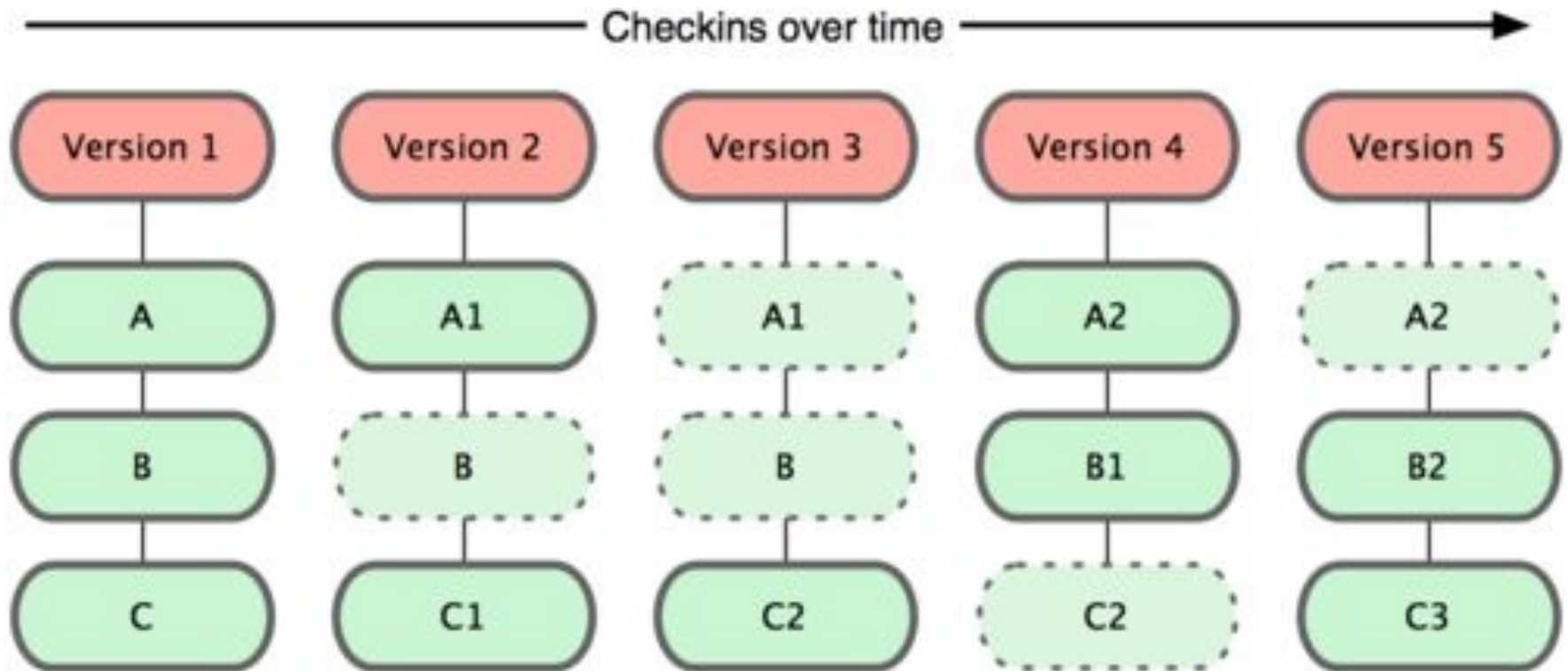
¿Cómo funciona git?

- | Instantáneas, no diferencias.
 - Así lo hace Subversion y compañía.



¿Cómo funciona git?

- Así lo hace git.



¿Cómo funciona git?

I Casi cualquier operación es local.

- Por ejemplo, para navegar por la historia del proyecto, Git no necesita salir al servidor para obtener la historia y mostrártela, simplemente la lee directamente de tu base de datos local. Esto significa que ves la historia del proyecto casi al instante.
- En Perforce, por ejemplo, no puedes hacer mucho cuando no estás conectado al servidor; y en Subversion y CVS, puedes editar archivos, pero no puedes confirmar los cambios a tu base de datos (porque tu base de datos no tiene conexión).



¿Cómo funciona git?

- | Git tiene Integridad.
- | El mecanismo que usa Git para generar esta suma de comprobación se conoce como hash SHA-1. Se trata de una cadena de 40 caracteres hexadecimales (0-9 y a-f), y se calcula en base a los contenidos del archivo o estructura de directorios. Un hash SHA-1 tiene esta pinta:

`24b9da6552252987aa493b52f8696cd6d3b00373`

¿Cómo funciona git?

- | Git generalmente sólo añade información.
 - Cuando realizas acciones en Git, casi todas ellas sólo añaden información a la base de datos de Git. Es muy difícil conseguir que el sistema haga algo que no se pueda deshacer, o que de algún modo borre información.

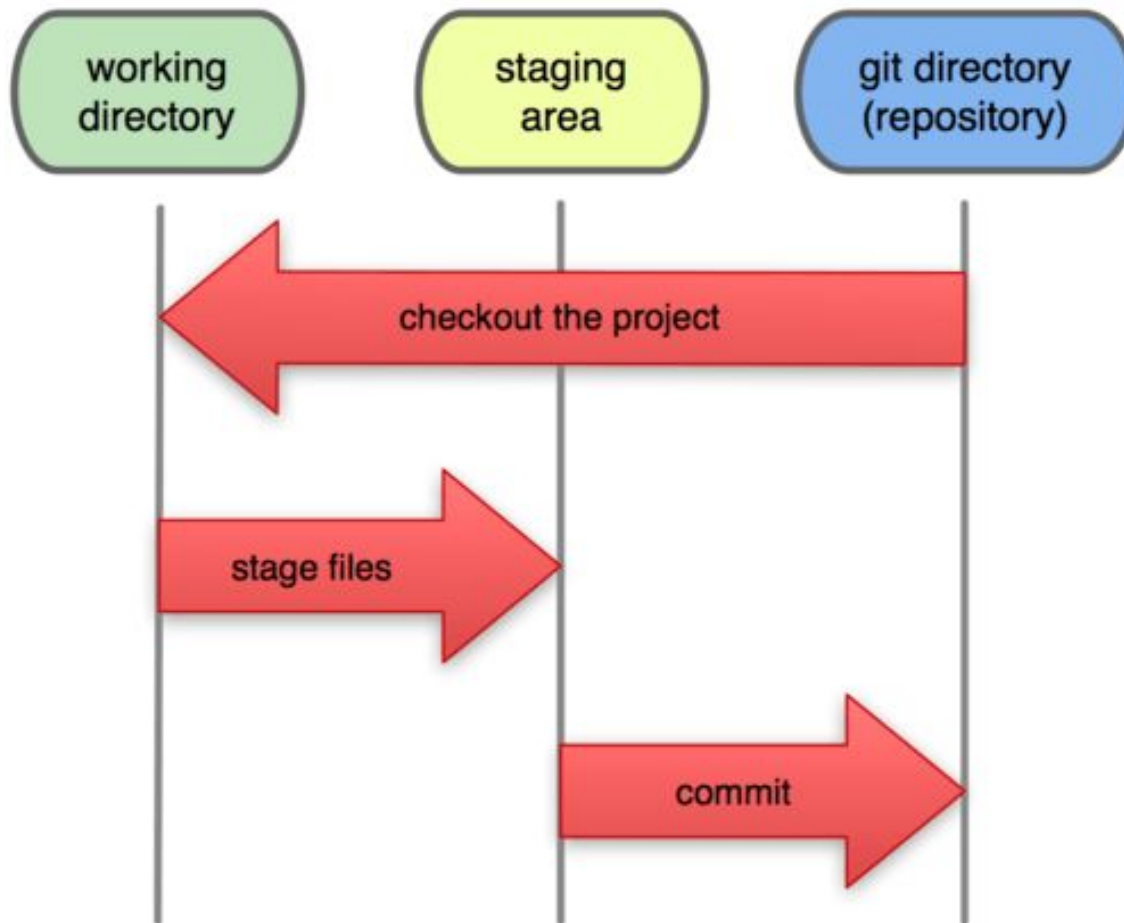


Los 3 estados

- | confirmado (committed), modificado (modified), y preparado (staged).
- | Confirmado significa que los datos están almacenados de manera segura en tu base de datos local.
- | Modificado significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
- | Preparado significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Los 3 estados

Local Operations





El flujo de trabajo básico en Git es algo así:

- | Modificas una serie de archivos en tu directorio de trabajo.
- | Preparas los archivos, añadiéndolos a tu área de preparación.
- | Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esas instantáneas de manera permanente en tu directorio de Git.

Instalación y configuración.

- | Instalando git.
 - `sudo apt-get install git`
- | Creando una cuenta en GitHub.
 - `github.com`
- | Configuración de mi usuario git.
 - `git config --global user.name "Esteban Quito"`
 - `git config --global user.email estebanquito@mail.com`



Inicializando un repositorio en un directorio existente

- | `git init`

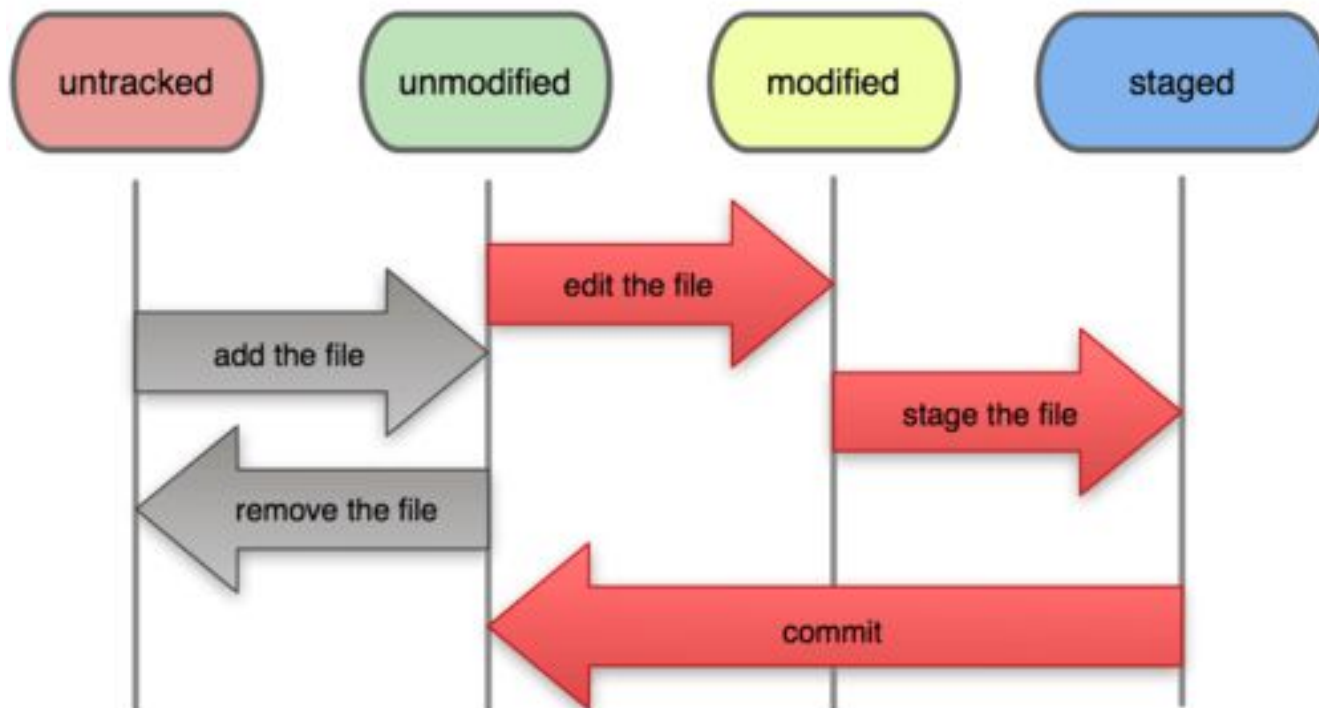


Clonando un repositorio existente

- | `git clone <url>`

Guardando cambios en el repositorio

File Status Lifecycle





Práctica

- | Comprobando el estado de tus archivos.
- | Seguimiento de nuevos archivos.
- | Preparando archivos modificados.
- | Ignorando archivos.
- | Viendo tus cambios preparados y no preparados.
- | Confirmando tus cambios.