



Introducción a GIT

Sistema de control de versiones


Autor [Alfredo Sánchez Alberca](#)



Introducción a GIT - Sistema de control de versiones

¿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 sobre los elementos de un proyecto o repositorio, guardando así versiones del mismo en todas sus fases de desarrollo.*

- Registra cada cambio en el proyecto o repositorio, quién y cuándo lo hace, en una base de datos.
- Permite volver a estados previos del desarrollo.
- Permite gestionar diferentes versiones del proyecto (ramas) para trabajar en paralelo y luego fundirlas.
- Permite colaborar entre diferentes usuarios en un mismo repositorio, facilitando la resolución de conflictos.
- Se utiliza principalmente en proyectos de desarrollo de software, pero  sirve para cualquier otro tipo de proyecto.

Introducción a GIT - Sistema de control de versiones



¿Qué es Git?

Git es un sistema de control de versiones de código abierto ideado por Linus Torvalds (el padre del sistema operativo Linux) y actualmente es el sistema de control de versiones más extendido.

A diferencia de otros SCV Git tiene una arquitectura *distribuida*, lo que significa que en lugar de guardar todos los cambios de un proyecto en un único sitio, cada usuario contiene una copia del repositorio con el historial de cambios completo del proyecto. Esto aumenta significativamente su rendimiento.



Introducción a GIT - Sistema de control de versiones



Configuración de Git

git config

- Establecer el nombre de usuario
`git config --global user.name "Your-Full-Name"`
- Establecer el correo del usuario
`git config --global user.email "your-email-address"`
- Activar el coloreado de la salida
`git config --global color.ui auto`
- Mostrar el estado original en los conflictos
`git config --global merge.conflictstyle diff3`
- Mostrar la configuración
`git config --list`



Introducción a GIT - Sistema de control de versiones



Creación de repositorios

Creación de un repositorio nuevo

git init

- **git init <nombre-repositorio>** crea un repositorio nuevo con el nombre **<nombre-repositorio>**.

Este comando crea una nueva carpeta con el nombre del repositorio, que a su vez contiene otra carpeta oculta llamada **.git** que contiene la base de datos donde se registran los cambios en el repositorio.



Introducción a GIT - Sistema de control de versiones



Copia de repositorios

git clone

- **git clone <url-repositorio>** crea una copia local del repositorio ubicado en la dirección **<url-repositorio>**.

A partir de que se hace la copia, los dos repositorios, el original y la copia, son independientes, es decir, cualquier cambio en uno de ellos no se verá reflejado en el otro.



Introducción a GIT - Sistema de control de versiones



Añadir cambios a un repositorio

Con Git, cualquier cambio que hagamos en un proyecto tiene que pasar por tres estados hasta que guarde definitivamente en el repositorio.

- **Directorio de trabajo** Es el directorio que contiene una copia de una versión concreta del proyecto en la que se está trabajando. Puede contener ficheros que no pertenecen al repositorio.
- **Zona temporal de intercambio** (staging area) es una zona donde se guardan los cambios temporalmente desde el directorio de trabajo antes de hacerlos definitivos y registrarlos en el repositorio.
- **Repositorio** Es donde finalmente se guardan los cambios definitivos desde la zona temporal de intercambio.



Añadir cambios a la zona de intercambio temporal

git add

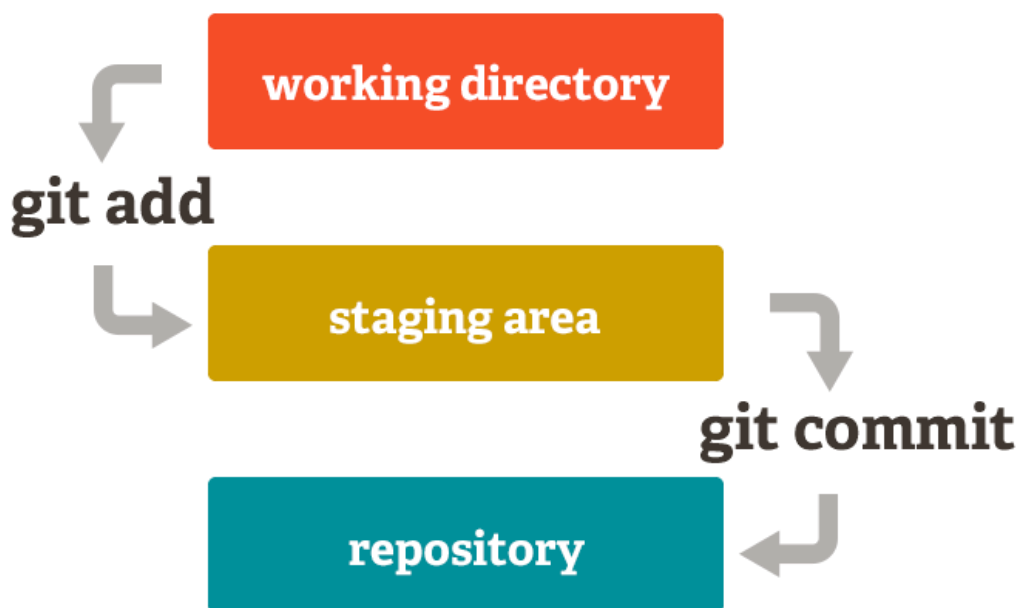
- **git add <fichero>** añade los cambios en el fichero <fichero> del directorio de trabajo a la zona de intercambio temporal.
- **git add <carpeta>** añade los cambios en todos los ficheros de la carpeta <carpeta> del directorio de trabajo a la zona de intercambio temporal.
- **git add .** añade todos los cambios de todos los ficheros no guardados aún en la zona de intercambio temporal.



Añadir cambios al repositorio

`git commit`

- `git commit -m "mensaje"` añade al repositorio todos los cambios almacenados en la zona de intercambio temporal creando una nueva versión del proyecto. "**mensaje**" es un breve mensaje describiendo los cambios realizados que se asociará a la nueva versión del proyecto.





Registro de cambios

Para guardar los cambios en un repositorio Git utiliza una estructura de tres niveles:

- **Commit** Contiene información sobre el autor, el momento y el mensaje de los cambios.
- **Árbol (tree)** Cada commit contiene además un árbol donde se registran los nombres y rutas de los ficheros en el repositorio cuando se hizo el commit.
- **Blob (binary file object)** Para cada uno de los ficheros listados en el árbol hay un blob, que contiene una instantánea comprimida del contenido del fichero cuando se hizo el commit.
Si un fichero del repositorio no ha cambiado en el commit, el árbol apunta al blob del fichero del último commit donde el fichero cambió.

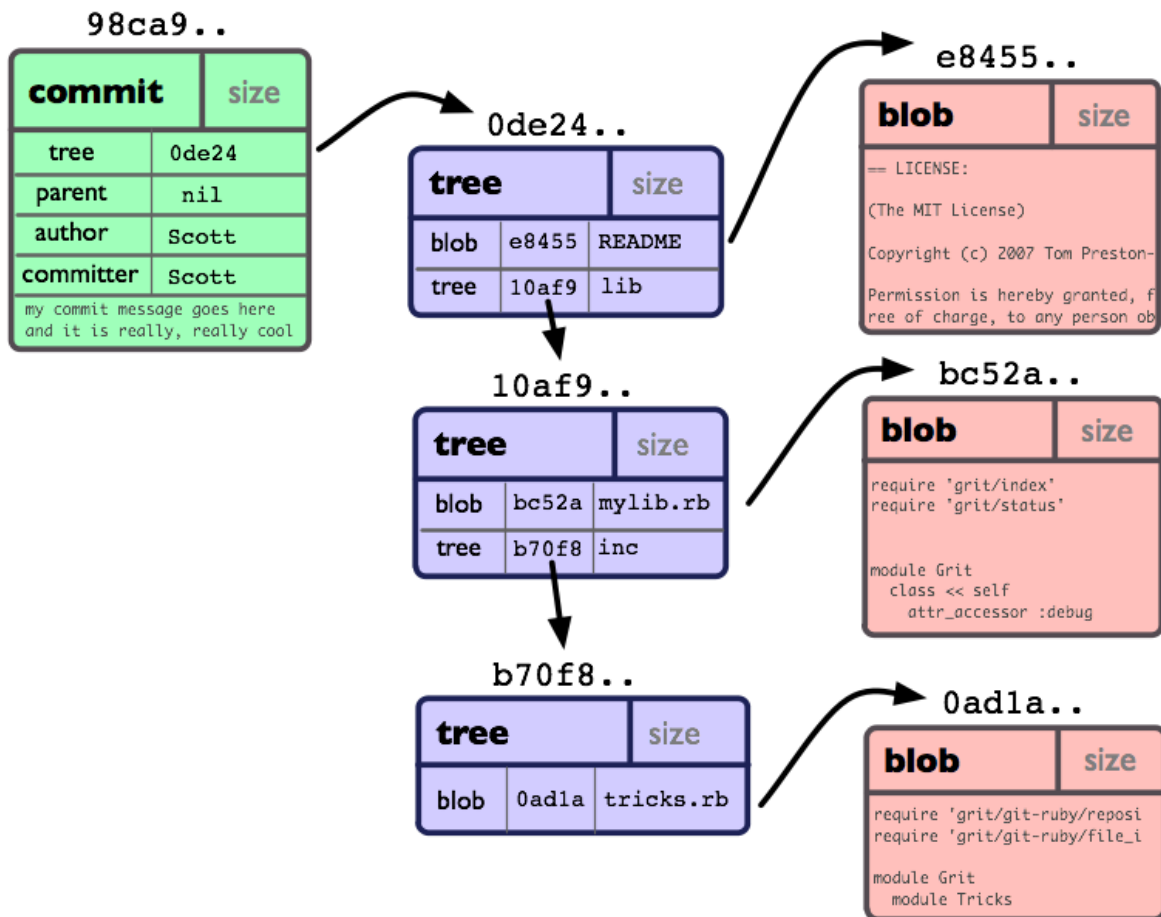


Referenciar un commit

Cada commit tiene asociado un código hash de 40 caracteres hexadecimales que lo identifica de manera única. Hay dos formas de referirse a un commit:

- **Nombre absoluto:** Se utiliza su código hash (basta indicar los 4 o 5 primeros dígitos).
- **Nombre relativo:** Se utiliza la palabra **HEAD** para referirse siempre al último commit. Para referirse al penúltimo commit se utiliza **HEAD~1**, al antepenúltimo **HEAD~2**, etc.





Estado e historia de un repositorio

Mostrar el estado de un repositorio

git status

- **git status** muestra el estado de los cambios en el repositorio desde la última versión guardada. En particular, muestra los ficheros con cambios en el directorio de trabajo que no se han añadido a la zona de intercambio temporal y los ficheros en la zona de intercambio temporal que no se han añadido al repositorio.

Mostrar el historial de versiones de un repositorio

git log

- **git log** muestra el historial de commits de un repositorio ordenado cronológicamente. Para cada commit muestra su código hash, el autor, la fecha, la hora y el mensaje asociado.



Mostrar los datos de un commit

git show

- **git show** muestra el usuario, el día, la hora y el mensaje del último commit, así como las diferencias con el anterior.
- **git show <commit>** muestra el usuario, el día, la hora y el mensaje del commit indicado, así como las diferencias con el anterior.





Mostrar el historial de cambios de un fichero

git annotate

- **git annotate** muestra el historial de cambios de un fichero ordenado cronológicamente.
Cada línea de la salida contiene los 8 primeros dígitos del código hash del commit correspondiente al cambio, el autor de los cambio, la fecha, el número de línea del fichero donde se produjo el cambio y el contenido de la línea.



Mostrar las diferencias entre versiones

git diff

- **git diff** muestra las diferencias entre el directorio de trabajo y la zona de intercambio temporal.
- **git diff --cached** muestra las diferencias entre la zona de intercambio temporal y el último commit.
- **git diff HEAD** muestra la diferencia entre el directorio de trabajo y el último commit.





Deshacer cambios

Eliminar cambios del directorio de trabajo o volver a una versión anterior

git checkout

- **git checkout <commit> -- <file>** actualiza el fichero **<file>** a la versión correspondiente al commit **<commit>**.
Suele utilizarse para eliminar los cambios en un fichero que no han sido guardados aún en la zona de intercambio temporal, mediante el comando **git checkout HEAD -- <file>**.



Introducción a GIT - Sistema de control de versiones



Eliminar cambios de la zona de intercambio temporal

git reset

- **git reset <fichero>** elimina los cambios del fichero **<fichero>** de la zona de intercambio temporal, pero preserva los cambios en el directorio de trabajo.

Para eliminar por completo los cambios de un fichero que han sido guardados en la zona de intercambio temporal hay que aplicar este comando y después **git checkout HEAD -- <fichero>**.



Introducción a GIT - Sistema de control de versiones



Eliminar cambios de un commit

git reset

- `git reset --hard <commit>` elimina todos los cambios desde el commit `<commit>` y actualiza el HEAD este commit.
¡Ojo! Usar con cuidado este comando pues los cambios posteriores al commit indicado se pierden por completo.
Suele usarse para eliminar todos los cambios en el directorio de trabajo desde el último commit mediante el comando `git reset --hard HEAD`.
- `git reset <commit>` actualiza el HEAD al commit `<commit>`, es decir, elimina todos los commits posteriores a este commit, pero no elimina los cambios del directorio de trabajo.



Introducción a GIT - Sistema de control de versiones



Ramas

Inicialmente cualquier repositorio tiene una única rama llamada **master** donde se van sucediendo todos los commits de manera lineal.

Una de las características más útiles de Git es que permite la creación de ramas para trabajar en distintas versiones de un proyecto a la vez.

Esto es muy útil si, por ejemplo, se quieren añadir nuevas funcionalidades al proyecto sin que interfieran con lo desarrollado hasta ahora.

Cuando se termina el desarrollo de las nuevas funcionalidades las ramas se pueden fusionar para incorporar los cambios al proyecto principal.



Introducción a GIT - Sistema de control de versiones

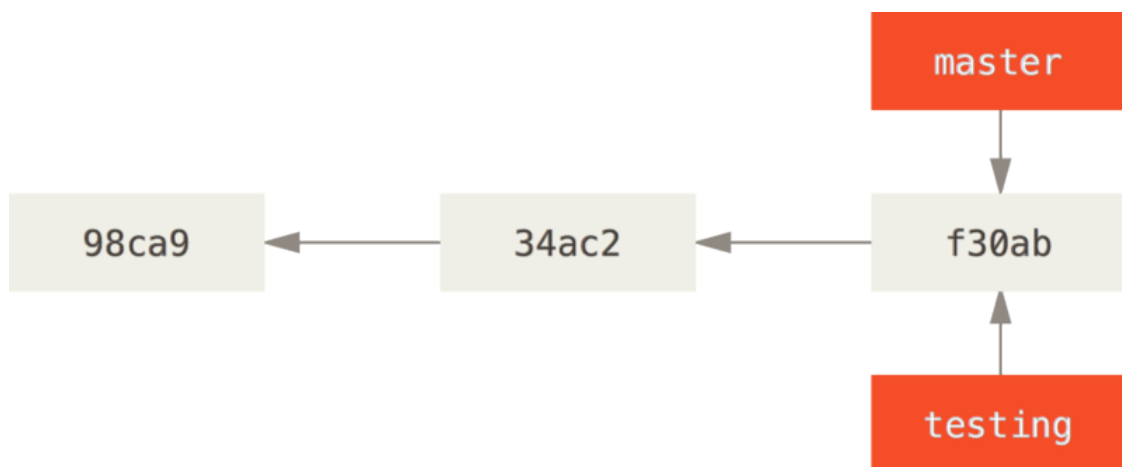


Creación de ramas

git branch

- **git branch <rama>** crea una nueva rama con el nombre <rama> en el repositorio a partir del último commit, es decir, donde apunte HEAD.

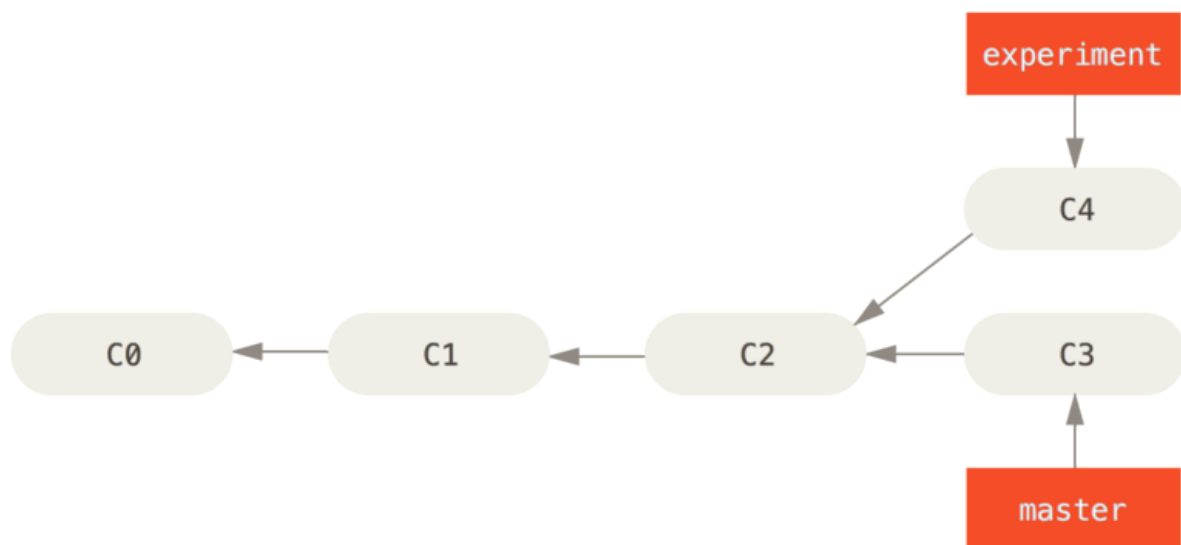
Al crear una rama a partir de un commit, el flujo de commits se bifurca en dos de manera que se pueden desarrollar dos versiones del proyecto en paralelo.



Introducción a GIT - Sistema de control de versiones



Desarrollo en ramas diferentes



Introducción a GIT - Sistema de control de versiones



Listado de ramas

git log

- **git branch** muestra las ramas activas de un repositorio indicando con * la rama activa en ese momento.
- **git log --graph --all --oneline** muestra la historia del repositorio en forma de grafo (--graph) incluyendo todas las ramas (--all).



Cambio de ramas

git checkout

- **git checkout <rama>** actualiza los ficheros del directorio de trabajo a la versión del repositorio correspondiente a la rama <rama>, HEAD pasa a apuntar al último commit de esta rama.

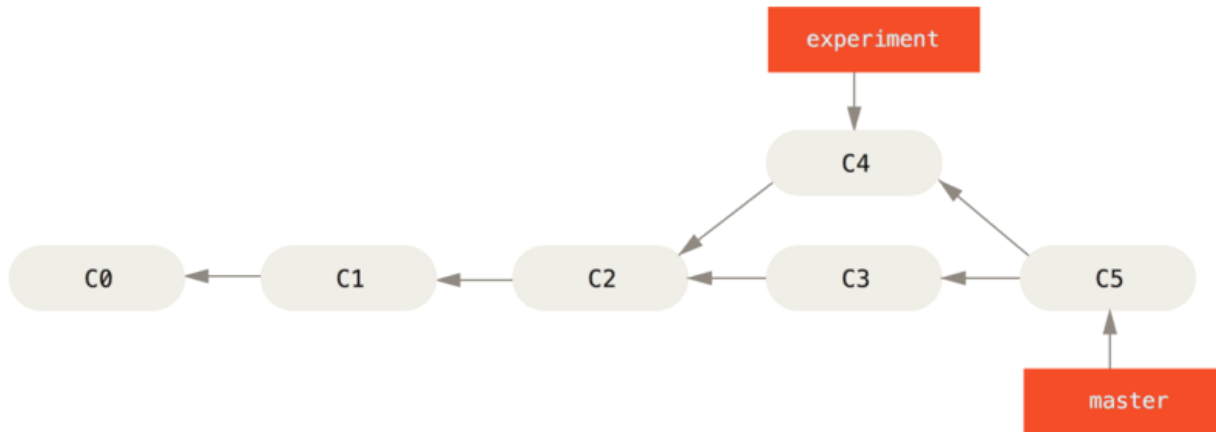




Fusión de ramas

git merge

- `git merge <rama>` integra los cambios de la rama `<rama>` en la rama actual a la que apunta HEAD.



Resolución de conflictos

Para fusionar dos ramas es necesario que no haya conflictos entre los cambios realizados a las dos versiones del proyecto.

Si en ambas versiones se han hecho cambios sobre la misma parte de un fichero, entonces se produce un conflicto y es necesario resolverlo antes de poder fusionar las ramas.

La resolución debe hacerse manualmente observando los cambios que interfieren y decidiendo cuales deben prevalecer, aunque existen herramientas como KDif3 o meld que facilitan el proceso.





Respositorios remotos

La otra característica de Git, que unida a las ramas, facilita la colaboración entre distintos usuarios en un proyecto son los repositorios remotos.

Git permite la creación de una copia del repositorio en un servidor git en internet. La principal ventaja de tener una copia remota del repositorio, a parte de servir como copia de seguridad, es que otros usuarios pueden acceder a ella y hacer también cambios.

Existen muchos proveedores de alojamiento para repositorios Git pero el más usado es GitHub.



Introducción a GIT - Sistema de control de versiones



¿Qué es GitHub?

GitHub es el proveedor de alojamiento en la nube para repositorios gestionados con git más usado y el que actualmente tiene alojados más proyectos de desarrollo de software de código abierto en el mundo.

La principal ventaja de GitHub es que permite albergar un número ilimitado de repositorios tanto públicos como privados, y que además ofrece servicios de registro de errores, solicitud de nuevas funcionalidades, gestión de tareas, wikis o publicación de páginas web, para cada proyecto, incluso con el plan básico que es gratuito.

GitHub



Introducción a GIT - Sistema de control de versiones



Añadir un repositorio remoto

git remote add

- **git remote add** <repositorio-remoto> <url> crea un enlace con el nombre <repositorio-remoto> a un repositorio remoto ubicado en la dirección <url>.

Cuando se añade un repositorio remoto a un repositorio, Git seguirá también los cambios del repositorio remoto de manera que se pueden descargar los cambios del repositorio remoto al local y se pueden subir los cambios del repositorio local al remoto.



Lista de repositorios remotos

git remote

- **git remote** muestra un listado con todos los enlaces a repositorios remotos definidos en un repositorio local.
- **git remote -v** muestra además las direcciones url para cada repositorio remoto.





Descargar cambios desde un repositorio remoto

git pull

- **git pull <remoto> <rama>** descarga los cambios de la rama **<rama>** del repositorio remoto **<remoto>** y los integra en la última versión del repositorio local, es decir, en el HEAD.
- **git fetch <remoto>** descarga los cambios del repositorio remoto **<remoto>** pero no los integra en la última versión del repositorio local.



Introducción a GIT - Sistema de control de versiones



Subir cambios a un repositorio remoto

git push

- **git push <remoto> <rama>** sube al repositorio remoto **<remoto>** los cambios de la rama **<rama>** en el repositorio local.



Introducción a GIT - Sistema de control de versiones



Referencias

- [Git](#). Sitio web de Git.
- [GitHub](#). Sitio web de GitHub.
- [Pro Git](#). Libro oficial de Git.
- [Ry's Git Tutorial](#). Tutorial de Git gratuito.
- [Gitcheats](#). Página de ayuda sobre los comandos de Git.
- [Trucos y consejos de Git](#) Resumen de los comandos más importantes y consejos sobre el uso de Git.
- [Chuleta de comandos de Git](#) Resumen de los comandos de Git más habituales.
- [Flujos de trabajo con Git](#) Esquema de los flujos de trabajo más habituales con Git.

