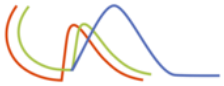
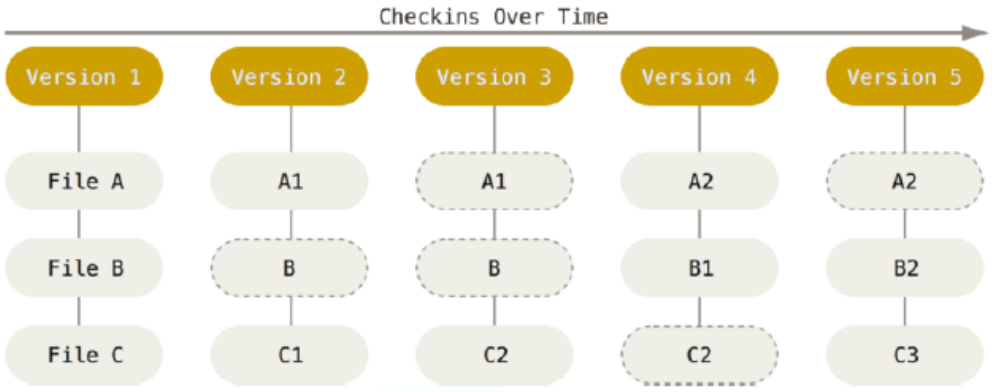


Contenido

Git es un sistema de control de versiones distribuido 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.

Cada vez que el usuario confirma un cambio, o guarda el estado de su proyecto en Git, básicamente toma una foto del aspecto de todos los archivos en ese momento y guarda una referencia a esa copia instantánea. Para ser eficiente, si los archivos no se han modificado Git no almacena el archivo de nuevo, sino un enlace al archivo anterior idéntico que ya tiene almacenado. Git maneja sus datos como una secuencia de copias instantáneas, tal como se muestra en la imagen:



Las particularidades o características distintivas de Git son:

Casi todas las operaciones son locales: la mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar. Por lo general no se necesita información de ningún otro equipo o servidor.

Git tiene integridad: todo en Git es verificado mediante una suma de comprobación (**checksum**) antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma. Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo sepa. Esta funcionalidad está integrada en Git al más bajo nivel y es parte integral de su filosofía. No se puede perder información durante su transmisión o sufrir corrupción de archivos sin que Git sea capaz de detectarlo.

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 y se calcula con base en los contenidos del archivo o estructura del directorio en Git. Un hash SHA-1 se ve de la siguiente forma:

24b9da6552252987aa493b52f8696cd6d3b00373

Estos valores hash son usados con mucha frecuencia. De hecho, Git guarda todo, no por nombre de archivo, sino por el valor hash de sus contenidos.

-**Git generalmente solo añade información**: cuando un usuario realiza acciones en Git, casi todas ellas sólo añaden información a la base de datos de Git.

ESTADOS

Los Tres Estados: Git tiene tres estados principales en los que se pueden encontrar los archivos del usuario:

o **Preparado (staged)**: el archivo ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadido al área de preparación, para que vaya en la próxima confirmación.

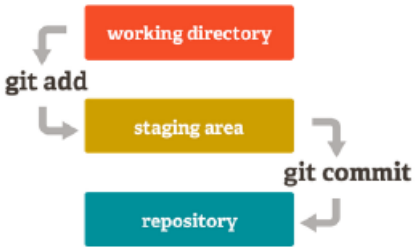
o **Modificado (modified)**: el archivo ha sufrido cambios desde que se obtuvo del repositorio, pero no se está preparado.

o **Confirmado (committed)**: los datos están almacenados de manera segura en el directorio de Git.

SECCIONES

Esto nos lleva a las tres secciones principales de un proyecto de Git:

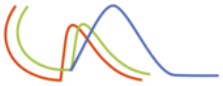
- **El directorio de trabajo (working directory)**: es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que el usuario los pueda usar o modificar.
- **El área de preparación (staging area)**: es un archivo, generalmente contenido en el directorio de Git del usuario, que almacena información acerca de los cambios a consolidar en la próxima confirmación. También se le denomina índice (**Index**).
- **El directorio de Git (Git directory - repository)**: es donde se almacenan los metadatos y la base de datos de objetos del proyecto. Es la parte más importante de Git, y es lo que se copia cuando se clona un repositorio desde otra máquina.



-

CONCEPTOS BÁSICOS

Para el manejo de este tipo de sistemas conviene aclarar previamente el vocabulario concreto que utilizan:



Repositorio (repository): es el lugar en el que se almacenan los datos actualizados e históricos de cambios.

Revisión (revision): es una versión determinada de la información que se gestiona. Hay sistemas que identifican las revisiones con un contador. Hay otros sistemas que identifican las revisiones mediante un código de detección de modificaciones.

Etiquetar (tag): permiten identificar de forma fácil revisiones importantes en el proyecto. Por ejemplo, se suelen usar tags para identificar el contenido de las versiones publicadas del proyecto.

Rama (branch): un conjunto de archivos puede ser ramificado o bifurcado en un punto en el tiempo de manera que, a partir de ese momento, dos copias de esos archivos evolucionan de forma independiente.

Cambio (change): un cambio (o diff, o delta) representa una modificación específica de un documento bajo el control de versiones. La granularidad de la modificación que es considerada como un cambio varía entre los sistemas de control de versiones.

Desplegar (checkout): es crear una copia de trabajo local desde el repositorio. Un usuario puede especificar una revisión en concreto u obtener la última. El término checkout también se puede utilizar para describir la copia de trabajo.

Confirmar (commit): es escribir o mezclar los cambios realizados en el repositorio. Los términos commit y checkin también se pueden utilizar como sustantivos para describir la nueva revisión que se crea como resultado de confirmar.

Conflicto (conflict): se produce cuando diferentes usuarios realizan cambios en el mismo documento y el sistema es incapaz de conciliar los cambios. Un usuario debe resolver el conflicto mediante la integración de los cambios, o mediante la selección de un cambio en favor del otro.

Cabeza (head): también a veces se llama tip (punta) y se refiere a la última confirmación, ya sea en el tronco (trunk) o en una rama (branch). El tronco y cada rama tienen su propia cabeza, aunque HEAD se utiliza para referirse al tronco.

Tronco (trunk): La única línea de desarrollo que no es una rama (a veces también llamada línea base, línea principal o máster).

Fusionar, integrar, mezclar (merge): una fusión o integración es una operación en la que se aplican dos tipos de cambios en un archivo o conjunto de archivos. Algunos escenarios de ejemplo son los siguientes:

1. Un usuario, trabajando en un conjunto de archivos, actualiza o sincroniza su copia de trabajo con los cambios realizados y confirmados, por otros usuarios, en el repositorio.
2. Un usuario intenta confirmar archivos que han sido actualizado por otros usuarios desde el último despliegue (checkout) y el software de control de versiones integra automáticamente los archivos.
3. Un conjunto de archivos se bifurca, un problema que existía antes de la ramificación se trabaja en una nueva rama, y la solución se combina luego en la otra rama.
4. Se crea una rama, el código de los archivos es independiente editado, y la rama actualizada se incorpora más tarde en un único tronco unificado.

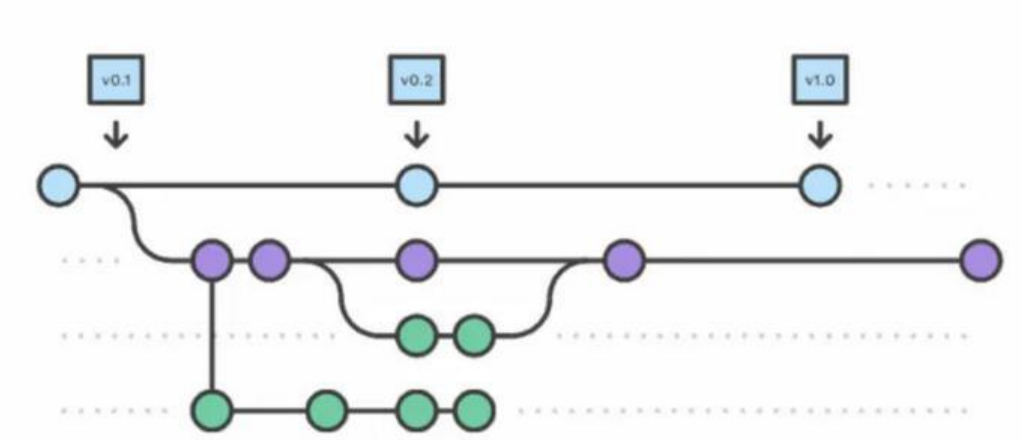
¿QUÉ ES UNA RAMA (BRANCH)?

Una **rama** en Git es un **apuntador a un commit específico** dentro del historial del repositorio. Cada vez que trabajamos en una rama, Git sigue la secuencia de commits que nacen desde ese punto.

Cuando inicializamos un repositorio, Git crea por defecto una rama llamada **main** (anteriormente master).

¿Para qué sirve una rama?
Las ramas permiten:

- Trabajar en **nuevas características o cambios** sin afectar la versión estable del proyecto.
- Desarrollar de manera **paralela y segura**.
- Realizar **experimentos o pruebas** sin alterar el código principal.
- Facilitar la **colaboración en equipo**, ya que cada desarrollador puede trabajar en su propia rama.



Comandos básicos de ramas

Acción	Comando	Descripción
Crear una nueva rama	git branch nombre-rama	Crea la rama sin cambiarte a ella
Cambiar a otra rama	git checkout nombre-rama	Cambia tu HEAD a la rama seleccionada
Crear y cambiarte de inmediato	git checkout -b rama	Crea la rama y te mueve a ella
Eliminar una rama local	git branch -d nombre-rama	Borra la rama local (si ya fue fusionada)
Eliminar rama local forzadamente	git branch -D nombre-rama	Fuerza el borrado aunque no esté fusionada
Eliminar rama remota	git push origin --delete nombre-rama	Elimina la rama del repositorio remoto

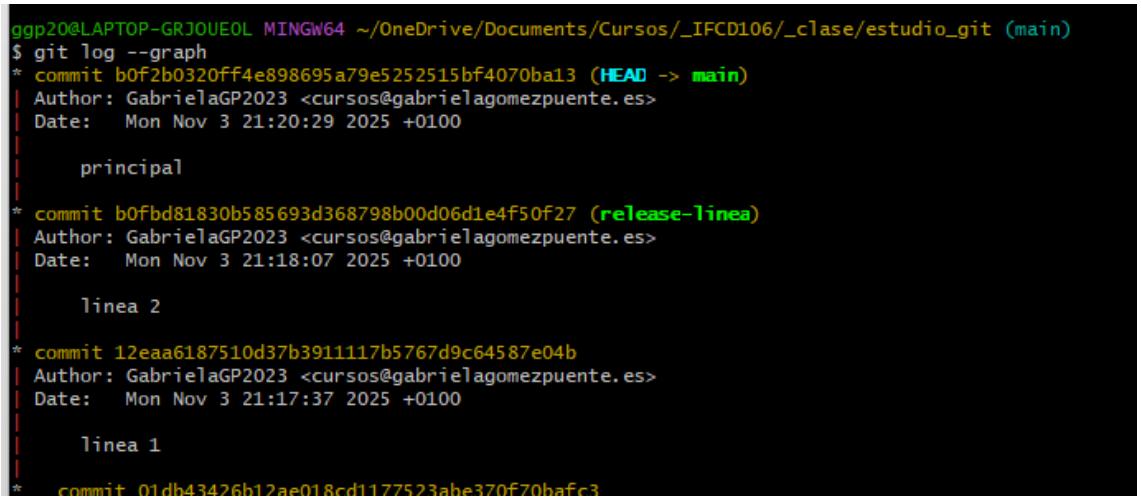
Acción	Comando	Descripción
Listar ramas locales	git branch	Muestra todas las ramas locales
Ramas **no fusionadas** a la actual	git branch --no-merged	Lista ramas con trabajo pendiente de unir
Ramas **ya fusionadas**	git branch --merged	Lista ramas que ya fueron integradas

REBASE ENTRE RAMAS

Permite "reaplicar" los commits de una rama secundaria sobre otra principal, creando un historial lineal.

```
git checkout rama-secundaria
git rebase rama-principal
```

Esto reescribe la historia de la rama secundaria, haciendo que parezca que sus commits fueron creados después de los de la principal.



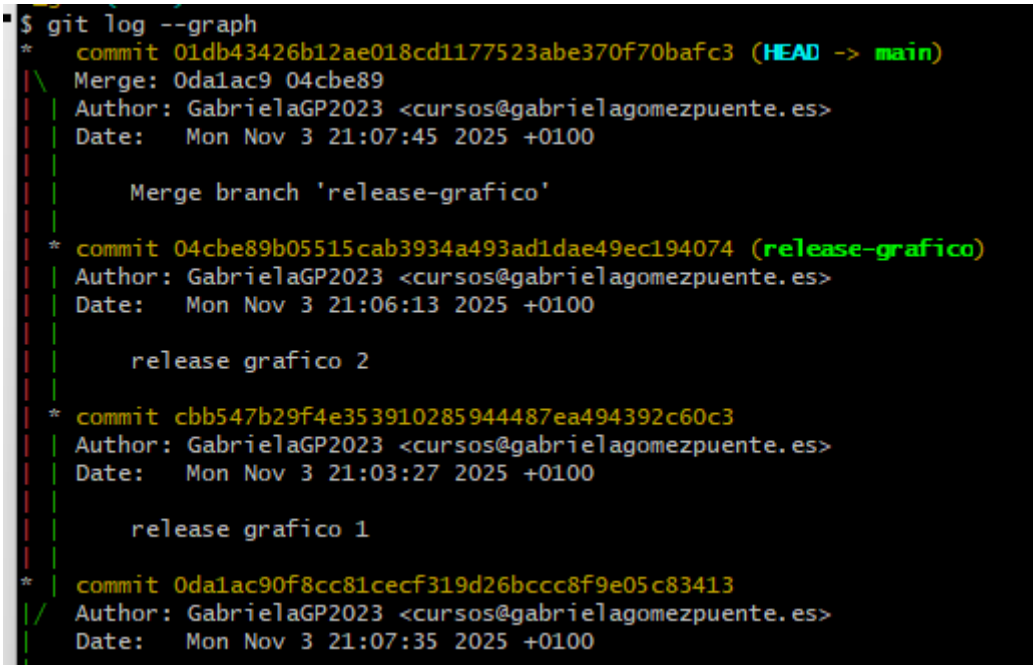
TIPOS DE FUSIONES (MERGE)

Existen dos algoritmos principales para fusionar ramas:

Fast-forward merge: Ocurre cuando la rama secundaria no ha creado commits que interfieran con la principal. Git simplemente mueve el puntero de la rama principal hacia adelante. No se crea un commit de merge adicional.

Merge con divergencia Sucede cuando ambas ramas tienen commits nuevos (es decir, sus historias se bifurcaron). Git combina las instantáneas (commits) de ambas ramas usando su ancestro común más reciente. Crea un commit de merge que une las dos historias.

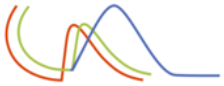
```
# Cambiarte a la rama principal
git checkout rama-principal

# Fusionar la rama secundaria
git merge rama-secundaria
```



ETIQUETAS (TAGS)

Las **etiquetas** sirven para marcar puntos específicos en la historia del repositorio, por ejemplo, una versión estable.

Acción	**Comando**
Crear una etiqueta	git tag numero-versión
Listar etiquetas existentes	git tag
Mostrar detalles de una etiqueta	git show numero-versión
Eliminar una etiqueta	git tag -d numero-versión
Subir una etiqueta al repositorio remoto	git push origin numero-versión



```
git add .
git tag v1.0.0
git commit -m "v1.0.0"
git push origin v1.0.0
```

Nomenclatura de versiones - https://semver.org/

IGNORAR ARCHIVOS (.GITIGNORE)

Permite indicarle a Git qué archivos o carpetas **no deben ser rastreados**.

Ejemplos de sintaxis:

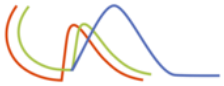
Patrón	**Significado**
archivo.ext	Ignora un archivo específico
/archivo_desde_raiz.ext	Ignora un archivo desde la raíz del proyecto
*.log	Ignora todos los archivos que terminen en .log
!production.log	Excepción: **no** ignorar este archivo
doc/*.txt	Ignora los .txt dentro de doc/, pero **no** en subcarpetas
doc/**/*.txt	Ignora todos los .txt en doc/ y **todas** sus subcarpetas

REPOSITORIOS REMOTOS

Permiten conectar tu repositorio local con uno alojado en la nube (como GitHub o GitLab).

Acción	**Comando**
Ver orígenes remotos	git remote
Ver orígenes con detalle	git remote -v
Agregar un remoto	git remote add nombre https://github.com/usuario/repositorio.git
Renombrar un remoto	git remote rename nombre-viejo nombre-nuevo
Eliminar un remoto	git remote remove nombre
Descargar una rama remota a local	git checkout --track -b rama origin/rama

GESTIÓN DE CAMBIOS



Acción	Comando
Modificar el último commit (sin cambiar mensaje)	git commit --amend --no-edit
Modificar el último commit (con nuevo mensaje)	git commit --amend -m "nuevo mensaje"
Eliminar el último commit por completo	git reset --hard HEAD~1

REGISTRO DEL HISTORIAL

Acción	Comando
Ver historial completo	git log
Ver historial resumido	git log --oneline
Guardar historial en archivo	git log > commits.txt
Personalizar formato	git log --pretty=format:"%h - %an, %ar : %s"
Ver últimos *n* commits	git log -n
Ver diagrama gráfico del historial	git log --graph --oneline

COMANDOS AVANZADOS

Comando	Explicación
git commit -a	Salta el área de *staging* y hace commit directo de archivos modificados
git log -p	Muestra los commits con las diferencias (patches) entre versiones
git log --stat	Muestra un resumen de los cambios (archivos modificados y líneas cambiadas)
git show	Muestra detalles del commit actual (líneas agregadas o eliminadas)
git diff	Compara los cambios entre el último commit y el directorio de trabajo
git diff --staged	Compara los cambios preparados para commit con la última versión confirmada
git add -p	Permite revisar interactivamente los cambios antes de agregarlos al *staging area*

Comando	Explicación
git mv <viejo> <nuevo>	Mueve o renombra un archivo
git rm <archivo>	Elimina un archivo del proyecto y del seguimiento de Git

COMANDOS MÁS USADOS

COMANDOS PARA CONFIGURAR

git --version

git config --global user.name "xxx"

git config --global user.email xxxx

git config --list

#asignando visual studio code como editor de configuración de git

git config --global core.editor "code --wait"

git config --global -e

#saltos de linea

git config --global core.autocrlf true

INICIALIZACIÓN DE REPOSITORIO

mkdir carpeta

cd carpeta

touch README.md

touch .gitignore

git init

code

FLUJO BÁSICO

agregar los cambios de un archivo al staged

git add archivo/directorio

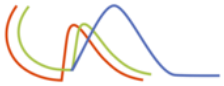
agregar todos los cambios de todos los archivos al staged

git add .

git commit -m "mensaje"

git status

git diff



AYUDA

git config -h

ver todas las opciones de la configuración en el navegador

git help config

REPOSITORIO EN GITHUB

se agrega el origen remoto de tu repositorio de GitHub

git remote add origin https://github.com/usuario/repositorio.git

la primera vez que vinculamos el repositorio remoto con el local

git push -u origin master

para las subsecuentes actualizaciones, sino cambias de rama

git push

#para descargar los cambios del repositorio remoto al local

git pull

ENLACES DE INTERÉS

- Documentación Git https://git-scm.com/doc
- Descargar git para Windows - https://gitforwindows.org/
- Nomenclatura de versiones - https://semver.org/