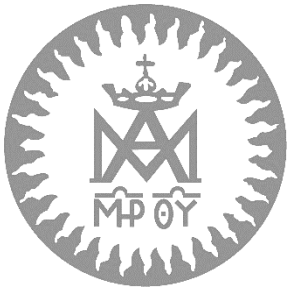


ENTORNOS DE DESARROLLO

TEMA 1

CONTROL DE VERSIONES

Colegio Calasanz 2022-2023

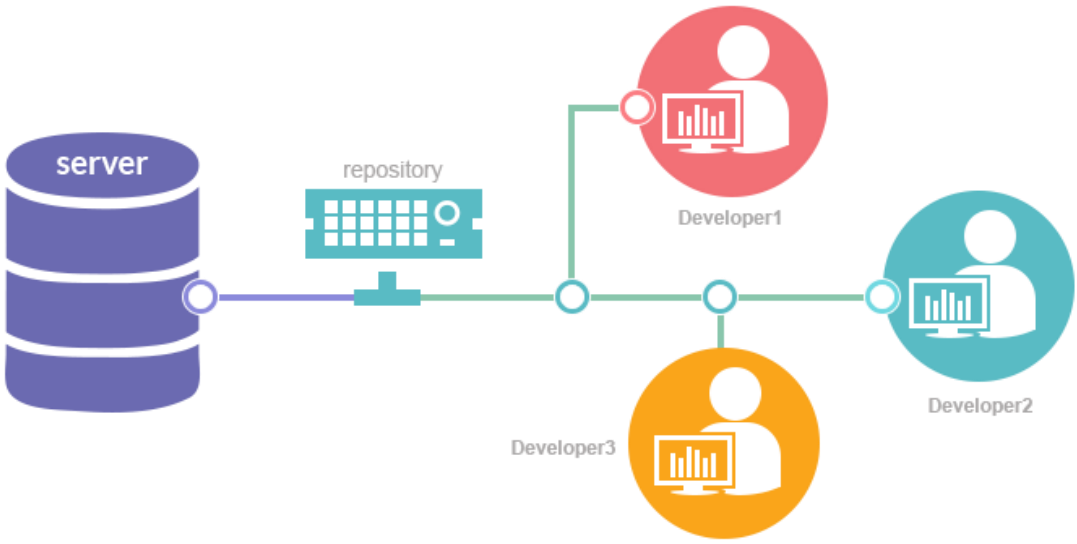
Introducción

El sistema de control de versiones, CVS (Control Version System), se encarga de registrar los cambios realizados sobre un archivo, o conjunto de archivos, a lo largo del tiempo, siendo posible la recuperación de ciertas versiones de los mismos, en cualquier momento. Estos sistemas se usan para registrar cualquier tipo de archivos, y comparar cambios a lo largo del tiempo, revertir a un estado anterior del proyecto, ver en qué momento se introdujo un error, etc.

Estas características convierten a los CVS como una herramienta muy útil en el trabajo de manera cooperativa, puesto que distintas personas pueden trabajar sobre los mismos archivos de una manera simultánea y compartir su trabajo de una forma inminente.

Un sistema de control de versiones debe proporcionar:

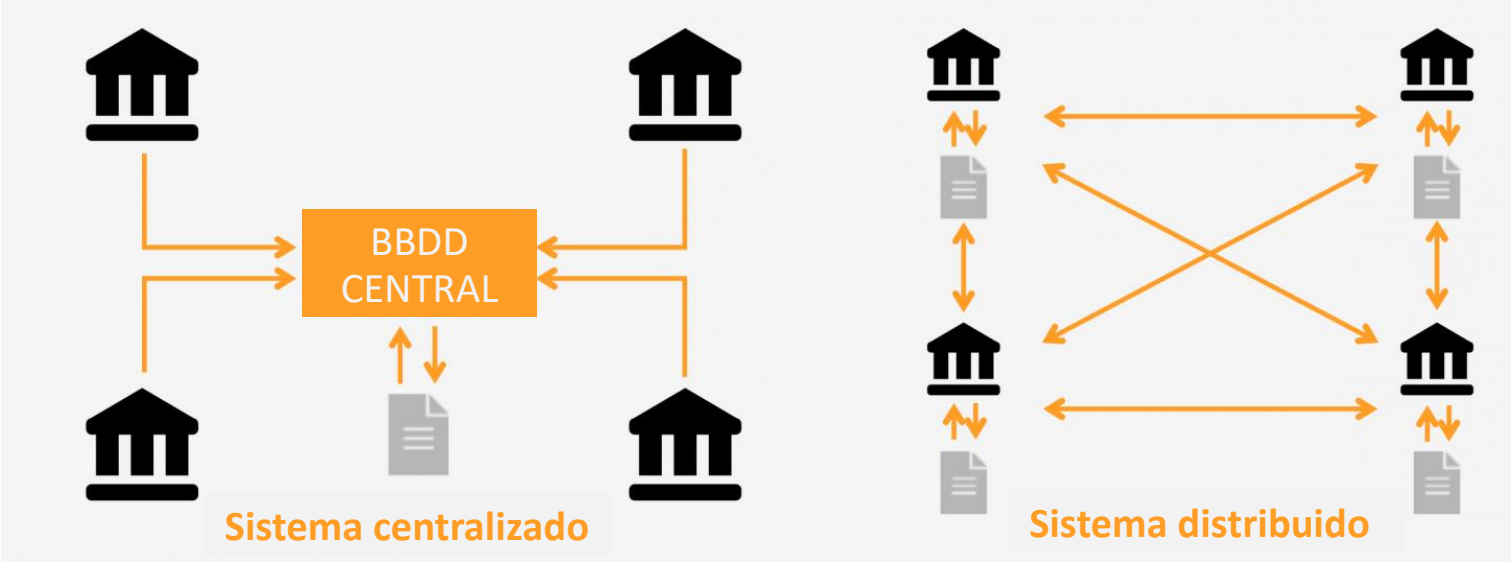
- Almacenamiento de archivos. (ej. archivos de texto, imágenes, documentación...).
- Posibilidad de realizar cambios sobre los elementos almacenados (ej. modificaciones parciales, añadir, borrar, renombrar o mover elementos).
- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).



Para utilizar casi cualquier CVS, hay que conocer y entender ciertos términos:

- **Repositorio**: Lugar donde se almacenan clases, carpetas, proyectos… datos y cambios.
- **Tag**: Etiqueta. Se trata de una instantánea del proyecto en el tiempo, el estado del repositorio completo, dando un nombre concreto con el que identificarla. Se suele crear una etiqueta cada vez que se va a liberar una nueva versión del software.
- **Branch**: Rama. Copia de archivos, carpetas o proyectos. Cuando una rama se crea se está creando una bifurcación del proyecto y pasa a haber dos líneas de desarrollo, facilitando así las pruebas.
- **Merge**: Fusión. Permite unir los diferentes cambios realizados sobre un archivo en una única versión, por ejemplo para unir los cambios realizados en dos ramas.
- **Checkout**: Descargarse una copia de archivos, carpetas o proyectos al equipo local de cualquier versión existente. Cuando se hace checkout, se vincula el proyecto local con el del repositorio.
- **Commit o check-in**: Confirmar. Cuando se quieren integrar los cambios realizados en local al repositorio se ha de hacer un commit.
- **Update**: Actualizar los archivos locales con los cambios del repositorio.
- **Conflicto**: Se produce cuando el sistema no es capaz de realizar la fusión. El usuario deberá resolverlo manualmente combinando los cambios o eligiendo uno de ellos.

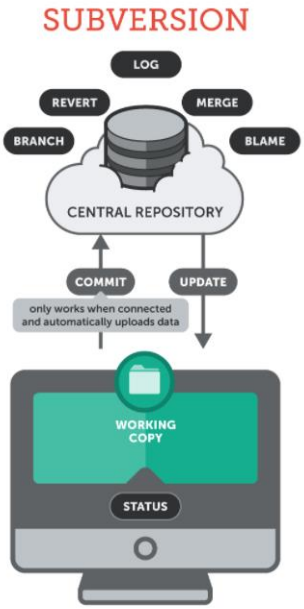
Los CVS los podemos diferenciar, según su arquitectura, en sistemas distribuidos y sistemas centralizados.



Sistema centralizado

Existe un repositorio central al que cada desarrollador del proyecto accederá mediante un cliente instalado en su equipo. Dicho repositorio contendrá todas las versiones de todos los archivos y los clientes se descargarán de ahí los archivos a modificar. Este tipo de arquitectura ha estado (y sigue estando) muy extendida, pero un servidor centralizado puede dar problemas si ese equipo se cae o si pierde datos. Algunos sistemas de este tipo son Subversion SVN de código abierto y Visual SourceSafe propietario.

Esta arquitectura necesita de un cliente y un servidor por separado para su funcionamiento.



Ventajas:

- Mayor control de la seguridad y protección, puesto que sólo hay que controlar un punto.
- Fácil de mantener.
- La curva de aprendizaje es menor.
- Manejo eficiente de archivos binarios grandes.

Inconvenientes:

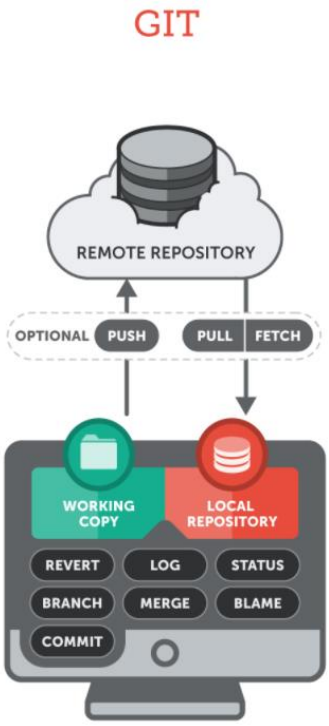
- Interfaz poco llamativa. Se evita tener pantalla con imágenes para controlar el ancho de banda de la red.
- La velocidad puede ser más lenta, puesto que depende de la conexión de la red a la central.
- Debe haber un mecanismo de copia o respaldo en caso de emergencia, puesto que si muere el sistema central, muere todo el sistema.
- El crecimiento es más complicado debido a la dependencia del sistema central.
- Menos utilizado (Entorno al 20% de los repositorios).

Sistema distribuido

Se define como una colección de computadoras separadas físicamente y conectadas entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software que el usuario percibe como un solo sistema.

En este tipo de sistemas los clientes tienen un repositorio local completo. Cada vez que se descarguen una versión del proyecto se está haciendo una copia de seguridad completa de todos los datos. Por lo tanto habrá tantas copias de seguridad como repositorios locales se hayan creado. Sistemas de este tipo de código abierto son GIT, cada vez más utilizado, y Mercurial.

Esta arquitectura cada máquina puede actuar como cliente y servidor.



Ventajas:

- Aumenta la confiabilidad al sistema. Esta arquitectura tiene redundancia, al fallar uno de los sistemas, las demás siguen funcionando.
- El crecimiento es más sencillo, puesto que se crean tantas copias como nuevos lugares tengamos.
- La toma de decisiones es local. La toma de decisiones en cada lugar es independiente de los otros.
- Permite una interfaz más amigable, puesto que solo se consume el ancho de banda de la red local.
- La velocidad de respuesta es más rápida. Tenemos lo datos en nuestra red local.
- La mayoría de sus funciones y características están disponible off-line.

- Más utilizado por la comunidad. (Aproximadamente 70% de los sistemas son distribuidos)

Inconvenientes:

- Soporte de la arquitectura. En cada lugar tenemos que tener el soporte tecnológico para esta arquitectura.
- Si la distribución de los datos es mala, es peor que un sistema centralizado.
- Coste y complejidad del Software.
- La integridad de los datos es más difícil de controlar.

Introducción a Git

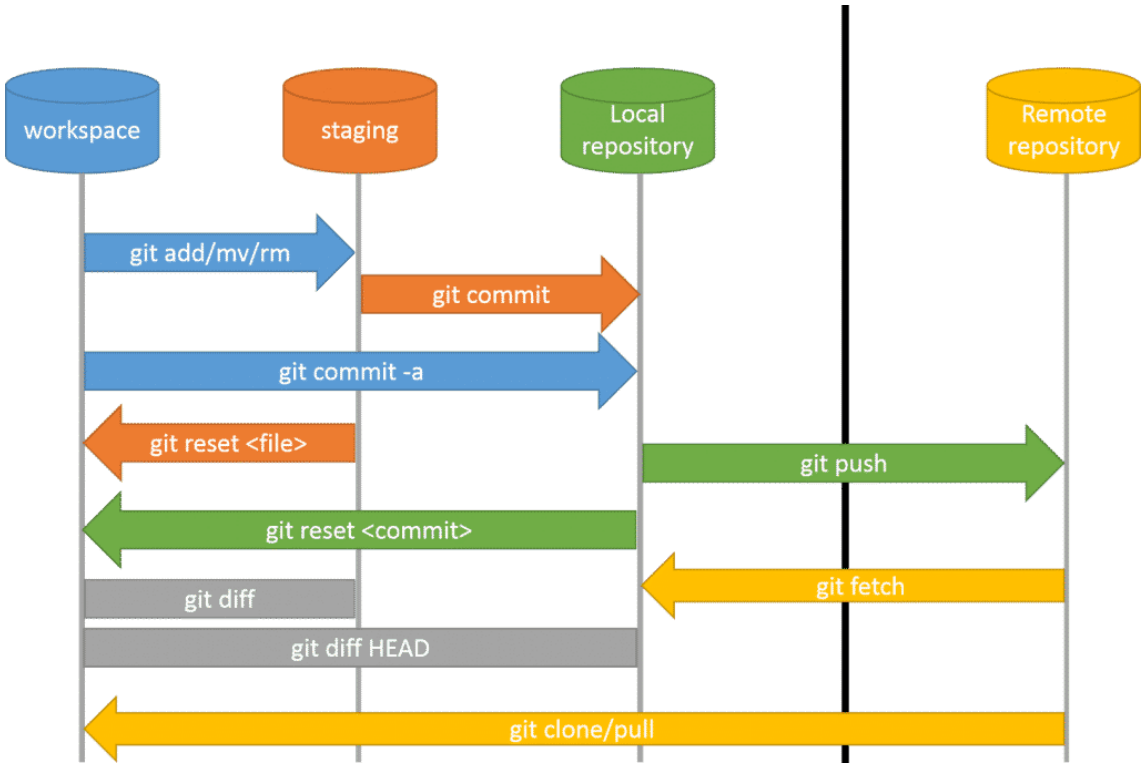
Git es un sistema de control de versiones distribuido creado por Linus Torvalds durante el desarrollo del núcleo de Linux.

Como ya se ha indicado para los sistemas distribuidos, trabajando con Git se tendrá una copia local completa del repositorio (repositorio local) y posteriormente se puede sincronizar con un repositorio remoto.

Flujo de trabajo

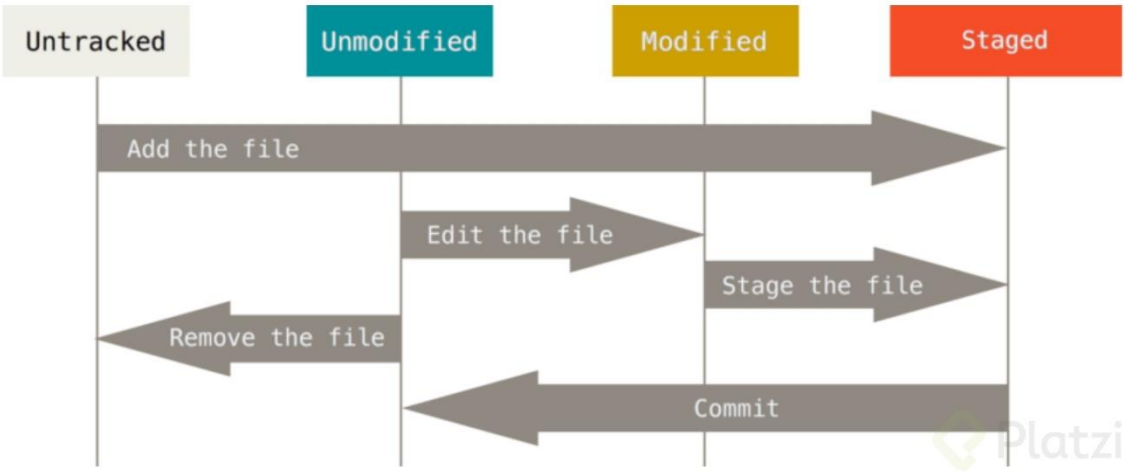
El flujo de trabajo habitual en un proyecto Git está compuesto por los siguientes elementos:

- Worksapace (directorio de trabajo)
- Staging area o index area (área de ensayo)
- Local repository (repositorio local)
- Remote repository (repositorio remoto)



Igualmente, los archivos del proyecto pueden estar en alguno de los siguientes estados:

- **Fuera de seguimiento (untracked):** Cuando se ha creado un fichero nuevo y no se ha incluido aún en el repositorio.
- **Modificado (modified):** Cuando se han detectado cambios en un archivo pero estos no se han consolidado, es decir, no se han persistido en el repositorio.
- **Preparado (staged):** Cuando se han marcado las modificaciones realizadas para un futuro commit, se conoce también como añadir al índice. Se considerará que el archivo estará en el "staging area". Los cambios que no hayan pasado al estado staged no se almacenarán en el siguiente commit.
- **Confirmado (commited/unmodified):** Cuando el archivo y su información han sido guardados en la base de datos de Git.

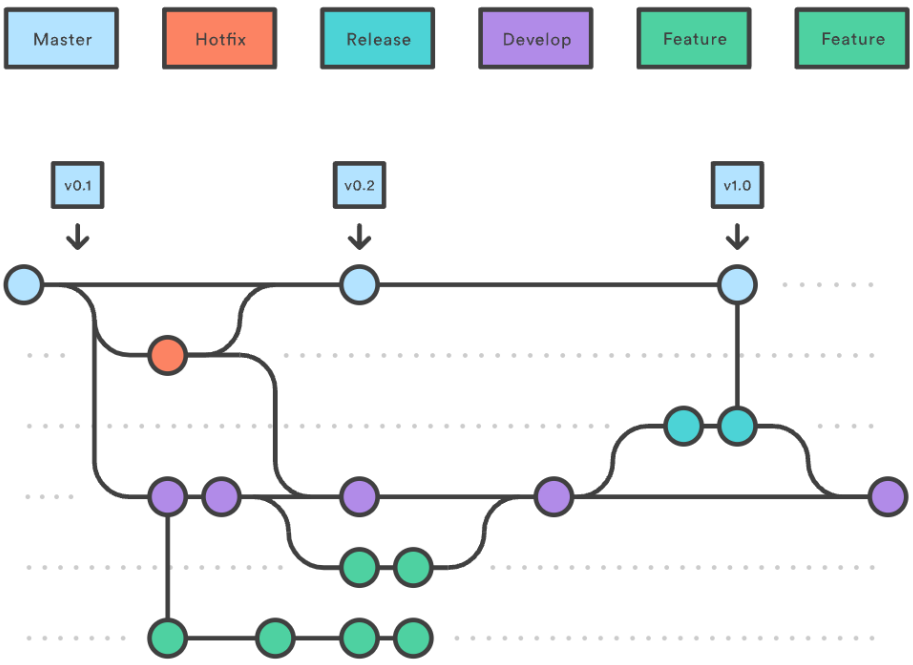


Servidores de repositorios

Para almacenar las versiones de los documentos con Git es necesario tener un servidor de control de versiones que esté basado, claro está, en Git. Existen muchas opciones gratuitas que, aunque tengan algunas limitaciones, son más que suficientes para proyectos de pequeña envergadura. Algunas opciones son GitHub, BitBucket, GitLab, etc.

Ciclo de vida de un proyecto Git

El ciclo de vida, a grandes rasgos, de un proyecto con Git como CVS sería:



Cada una de las ramas tendrá un sentido funcional, como vamos a ver a continuación:

- **Master:** En la rama máster se encuentran las *releases* (lanzamientos) estables de nuestro software. Esta es la rama que un usuario típico se descargará para usar nuestro software, por lo que todo lo que hay en esta rama debería ser funcional. Sin embargo, puede que las últimas mejoras introducidas en el software no estén disponibles todavía en esta rama.
- **Develop**: En esta rama surge de la última *release* de master. En ella se van integrando todas las nuevas características hasta la siguiente *release*.
- **Feature**: Cada nueva mejora o característica que vayamos a introducir en nuestro software tendrá una rama que contendrá su desarrollo. Las ramas de *feature* salen de la rama develop y una vez completado el desarrollo de la mejora, se vuelven a integrar en develop.
- **Release**: Las ramas de *release* se crean cuando se va a publicar la siguiente versión del software y surgen de la rama *develop*. En estas ramas, el desarrollo de nuevas características se congela, y se trabaja en arreglar bugs y generar documentación. Una vez listo para la publicación, se integra en master y se etiqueta con el número de versión correspondiente. Se integran también con *develop*, ya que su contenido ha podido cambiar debido a nuevas mejoras.
- **Hotfix**: Si nuestro código contiene bugs críticos que es necesario parchear de manera inmediata, es posible crear una rama *hotfix* a partir de la publicación correspondiente en la rama master. Esta rama contendrá únicamente los cambios que haya que realizar para parchear el bug. Una vez arreglado, se integrará en master, con su etiqueta de versión correspondiente y en *develop*.