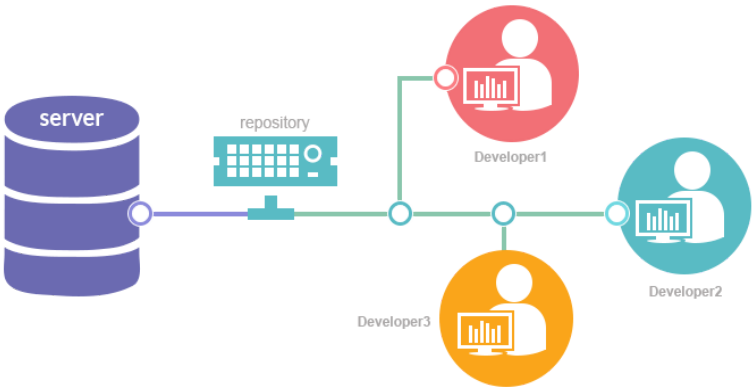
Taller de Programación Web

Versionado

Quién alguna vez no deseó poder volver al pasado para solucionar algún tipo de problema que tengamos en el presente.

Por desgracia, estos viajes temporales no son posibles en la vida real, pero al menos sí que es posible hacerlo en el desarrollo de software gracias al control de versiones.

¿Qué es el control de versiones?



*Se llama **control de versiones** a los métodos y herramientas disponibles para controlar todo lo referente a los cambios en el tiempo de un archivo.*

Entonces, un **sistema de control de versiones (VCS)** es una herramienta capaz de registrar todos los cambios que se realizan en uno o más proyectos, guardando a su vez versiones anteriores del proyecto, versiones a las que podemos acudir en caso de que no funcionen de la forma correcta.

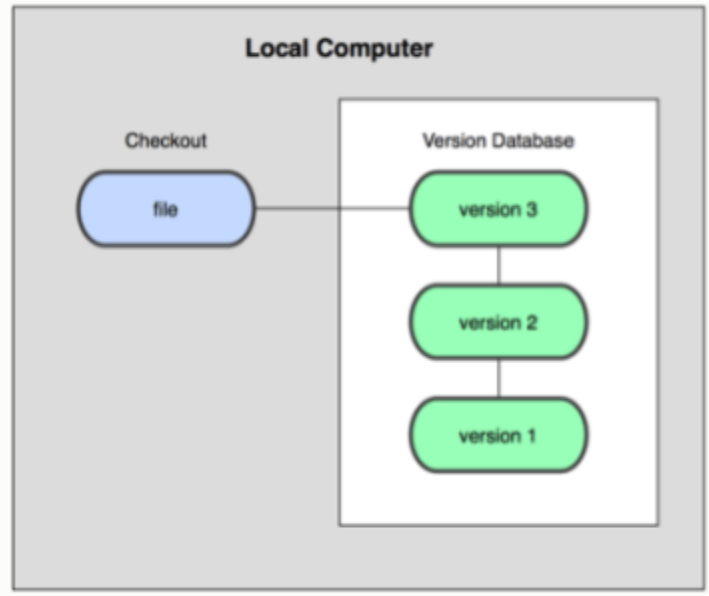
Ventajas de utilizar un VCS

El control de versiones permite que los desarrolladores se muevan más rápido y posibilita que los equipos de software mantengan la eficacia y la agilidad a medida que el equipo se escala para incluir más desarrolladores. Los sistemas de control de versiones (VCS) han experimentado grandes mejoras en las últimas décadas y algunos son mejores que otros, pero entre sus principales ventajas se observa que:

1) **Se cuenta con un historial de cambios a largo plazo de todos los archivos.** Esto quiere decir todos los cambios realizados por muchas personas a lo largo de los años. Los cambios incluyen la creación y la eliminación de los archivos, así como los cambios de sus contenidos. Tener el historial completo permite volver a las versiones anteriores para ayudar a analizar la causa raíz de los errores y es crucial cuando se tiene que solucionar problemas en las versiones anteriores del software.
2) **Creación de ramas y fusiones.** Si se tiene a integrantes del equipo trabajando al mismo tiempo, es algo evidente; pero incluso las personas que trabajan solas pueden beneficiarse de la capacidad de trabajar en flujos independientes de cambios. La creación de una "rama" en las herramientas de VCS mantiene múltiples flujos de trabajo independientes los unos de los otros al tiempo que ofrece la facilidad de volver a fusionar ese trabajo, lo que permite que desarrolladores verifiquen que los cambios de cada rama no entran en conflicto.
3) **Trazabilidad.** Ser capaz de seguir cada cambio que se hace en el software y conectarlo con un software de gestión de proyectos y seguimiento de errores, además de ser capaz de anotar cada cambio con un mensaje que describa el propósito y el objetivo del cambio, no solo te ayuda con el análisis de la causa raíz y la recopilación de información.

Aunque se puede desarrollar software sin utilizar ningún control de versiones, hacerlo somete al proyecto a un gran riesgo que ningún equipo profesional debería aceptar. ***Así que la pregunta no es si utilizar el control de versiones, sino qué sistema de control de versiones usar.***

Clasificación de los sistemas de control de versiones



Sistemas de control de versiones locales

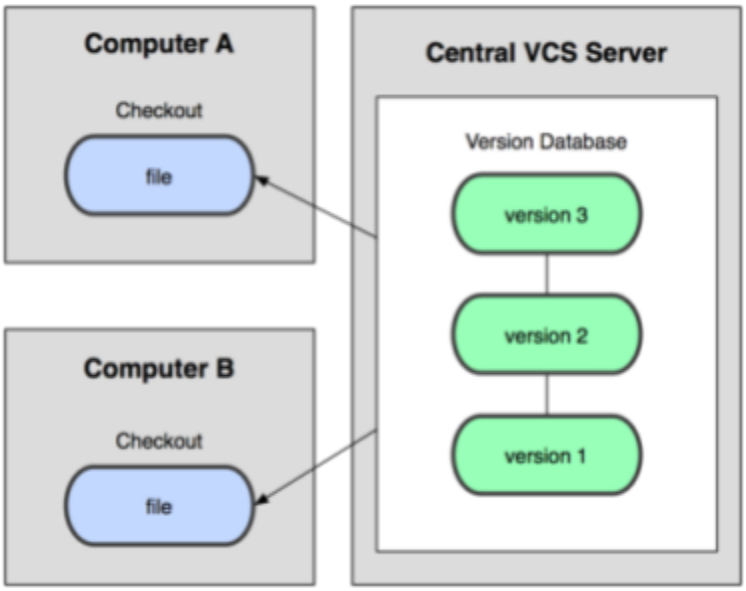
Uno de los métodos más utilizados a la hora de realizar algún tipo de control de versión de sus cambios, consiste en copiar en un **directorio de su equipo local el archivo** que iba a ser modificado indicando la fecha de modificación, para que en caso de error se supiese cuál era la última versión guardada.

Este sistema puede ser útil para el desarrollo de una aplicación pequeña, pero ofrece ciertos problemas como el de no recordar dónde hemos guardado la copia o simplemente olvidarnos de hacerla.

Para hacer frente a estos problemas, los equipos de software desarrollaron los conocidos como VCSs locales, que consistían en una base de datos donde se llevaba un registro de los cambios realizados sobre los archivos.

Sistemas de control de versiones centralizados

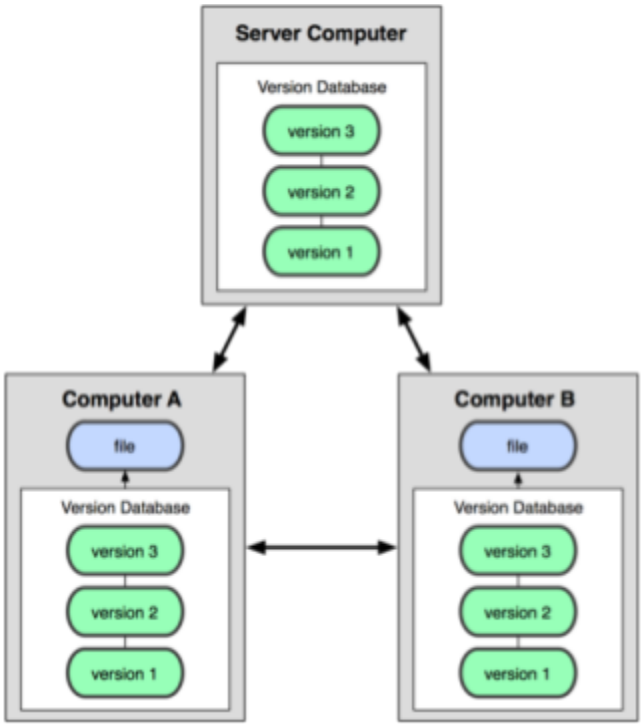
El sistema comentado anteriormente nos puede valer en el caso de que estemos trabajando solos, pero en un proyecto suelen intervenir varias personas y cada una de ellas se encarga de realizar una tarea determinada. En este caso, es necesario poder contar con sistemas de control de versiones centralizados.



En estos sistemas nos encontramos un único servidor que contiene todos los archivos versionados, y los usuarios que forman parte del proyecto se los pueden descargar desde ese servidor centralizado.

Este sistema tiene un problema muy claro, y es que al utilizar un único servidor centralizado, en caso de problema en ese servidor toda la información se podría perder.

Un ejemplo de este tipo de VCS es Subversion.



Sistemas de control de versiones distribuidos

En estos sistemas no tenemos un único servidor que mantenga la información del proyecto, sino que cada usuario contiene una copia completa del proyecto de forma local.
De esta forma, si un servidor cae, cualquiera de los repositorios de los clientes se podría utilizar para restaurar el servidor.

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 VCS, 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.

Términos que debes conocer antes de avanzar

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

Revisión ("revision") Una revisión es una versión determinada de la información que se gestiona. Hay sistemas que identifican las revisiones con un contador (Ej. subversion). Hay otros sistemas que identifican las revisiones mediante un código de detección de modificaciones (Ej. git usa SHA1).

Etiqueta ("tag") Los tags 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 se pueden desarrollar a velocidades diferentes o en formas diferentes de forma independiente el uno del otro.

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 como un sustantivo para describir la copia de trabajo.

Confirmar ("commit") Confirmar es escribir o mezclar los cambios realizados en la copia de trabajo del 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") Un conflicto se produce cuando diferentes partes 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 a veces libremente 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 quese aplican dos tipos decambiosen un archivo o conjunto de archivos. Algunos escenarios de ejemplo son los siguientes:

• 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.

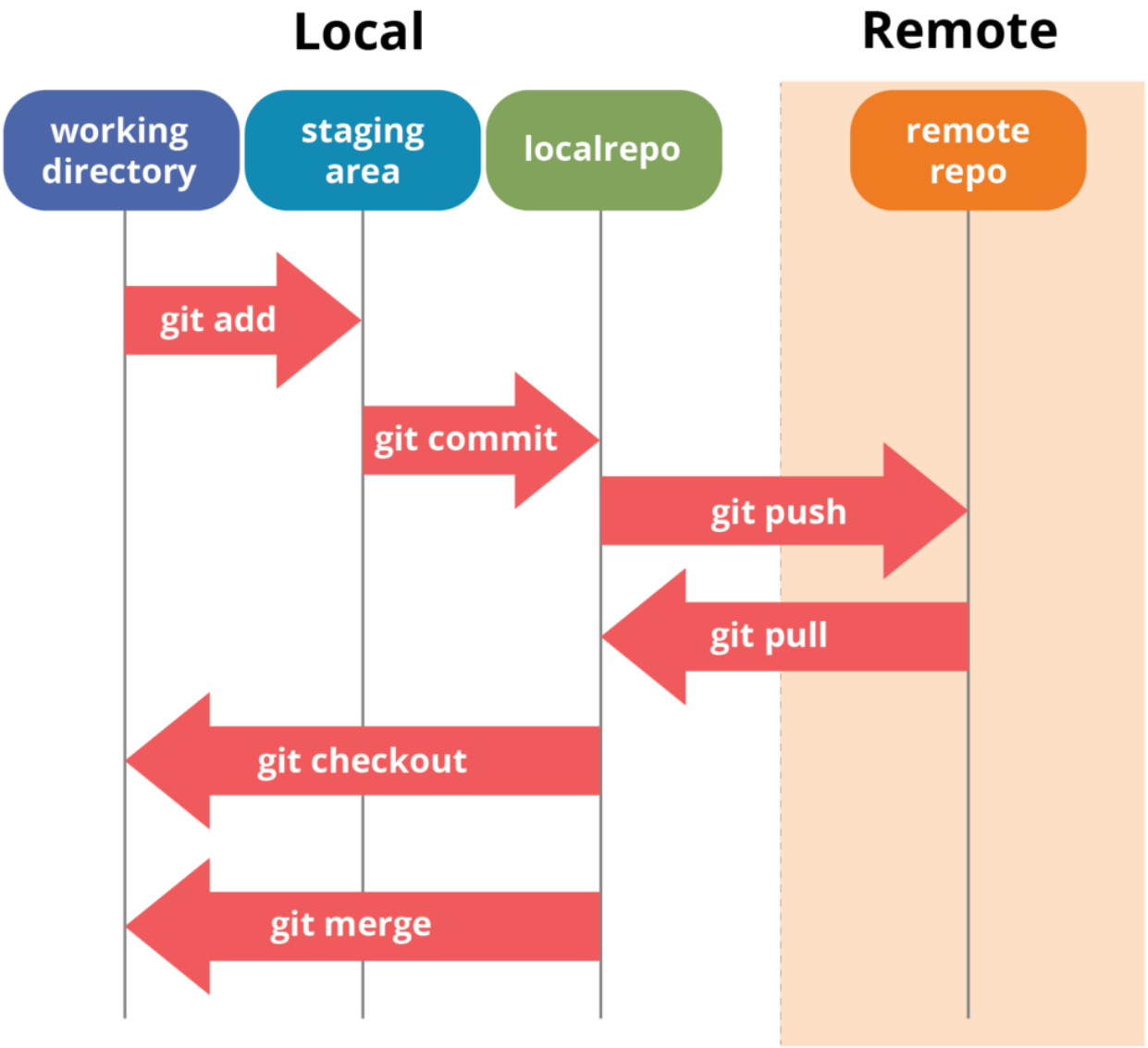
• Un usuario intenta confirmar archivos que han sido actualizado por otros usuarios desde el último despliegue ('checkout'), y el software de control de versiones integra automáticamente los archivos (por lo general, después de preguntarle al usuario si se debe proceder con la integración automática, y en algunos casos sólo se hace si la fusión puede ser clara y razonablemente resuelta).

• 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.

• 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.

Los estados y las áreas de trabajo de GIT

Git tiene **tres estados principales** en los que se pueden encontrar tus archivos: 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 ha 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.



- **Working dir/Área de trabajo:** Es el estado actual de nuestros ficheros, aquellos en los que programamos y que podemos visualizar en nuestro navegador de archivos.
- **Staging Area/Index/Área de ensayo:** Es una zona intermedia donde vamos almacenando los cambios que van a conformar un commit. Pueden ser ficheros completos o solo porciones concretas. Para "pasar" ficheros a esta zona usamos el comando add. Necesariamente hay que enviar los cambios al staging area antes de poder realizar un commit.
- **El repositorio local:** Es donde se almacenan los commits mediante el comando commit, una vez hemos seleccionado los cambios en el index. Toda su información se guarda en el directorio .git.
- **Los repositorios remotos:** Puede ser uno solo (por defecto se llama origin), o podríamos tener varios. La comunicación entre el repositorio local y los remotos se realiza mediante los comandos push(enviar) y pull(descargar).
- ***Stash:** Es la zona de guardado rápido, una zona auxiliar para guardar los cambios en los que estamos trabajando cuando por algún motivo nos interrumpen y tenemos que cambiar de rama, pero aún no queremos hacer un commit porque es un commit a medias, sin acabar. Puede almacenar n estados y funciona como una pila, colocando siempre el primero los últimos cambios que salvemos.

No es posible hacer un push a un remoto de un commit que se encuentre en la zona de stash.
Por lo que esta zona es únicamente de uso privado.