MEDAC
Instituto Oficial de Formación Profesional

# Tarea Tema 18

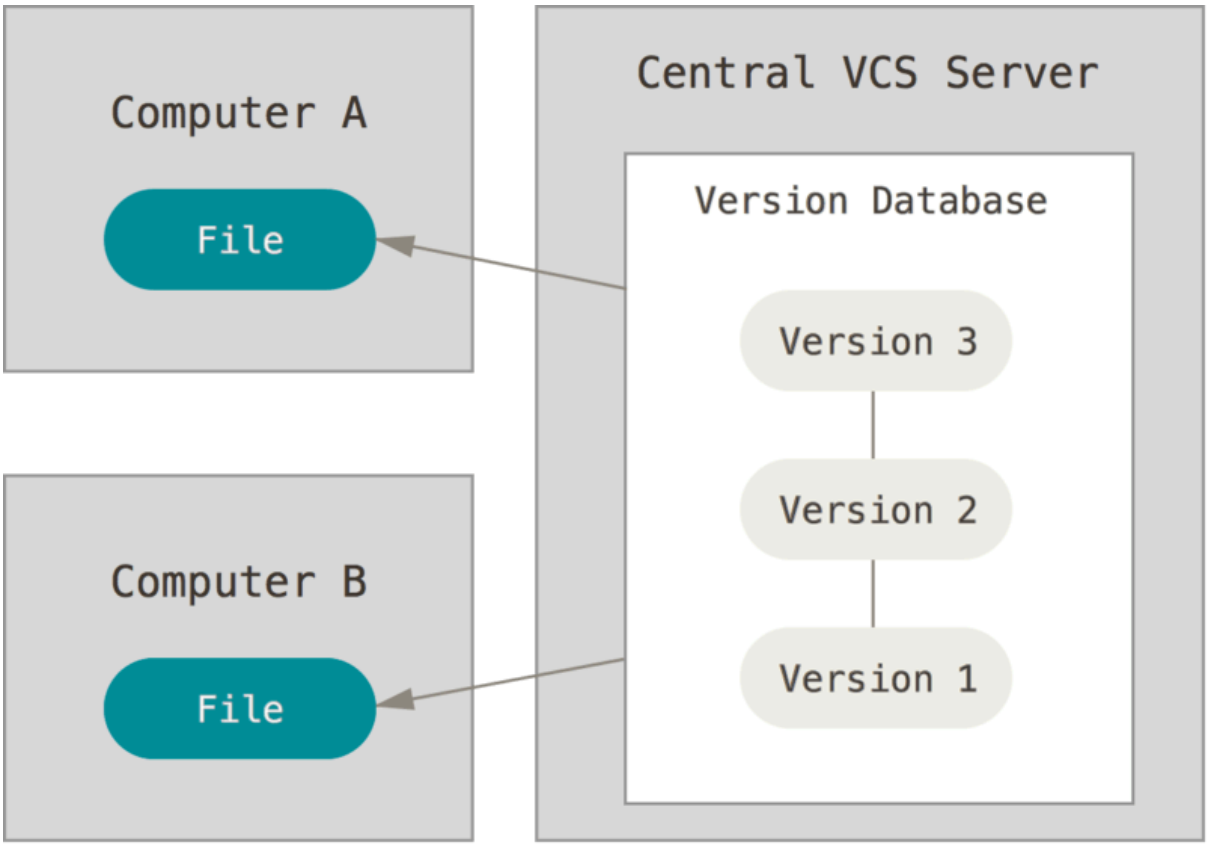
## Entornos de desarrollo

## Control de versiones:

Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. Dicho sistema te permite regresar a versiones anteriores de tus archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un VCS también significa generalmente que si arruinas o pierdes archivos, será posible recuperarlos fácilmente. Adicionalmente, obtendrás todos estos beneficios a un costo muy bajo.

## Sistema centralizado:

Los sistemas de Control de Versiones Centralizados (CVCS por sus siglas en inglés) fueron desarrollados para solucionar este problema. Estos sistemas, como CVS, Subversion y Perforce, tienen un único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos desde ese lugar central. Este ha sido el estándar para el control de versiones por muchos años.




Esta configuración ofrece muchas ventajas, especialmente frente a VCS locales. Por ejemplo, todas las personas saben hasta cierto punto en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado sobre qué puede hacer cada usuario, y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han realizado copias de seguridad adecuadamente, se perderá toda la información del proyecto, con excepción de las copias instantáneas que las personas tengan en sus máquinas locales. Los VCS locales sufren de este mismo problema: Cuando tienes toda la historia del proyecto en un mismo lugar, te arriesgas a perderlo todo.

***Ventajas:***

**Simplicidad:** Suelen ser más fáciles de entender y utilizar, especialmente para equipos pequeños o no técnicos.

**Control de acceso:** El control de acceso es más fácil de gestionar, ya que todas las versiones están en un solo lugar.

**Coherencia de versión:** Todos los miembros del equipo trabajan en la misma versión principal del código.

**Historial centralizado:** El historial de cambios se encuentra en un único repositorio, lo que facilita la administración y la referencia.

***Desventajas:***

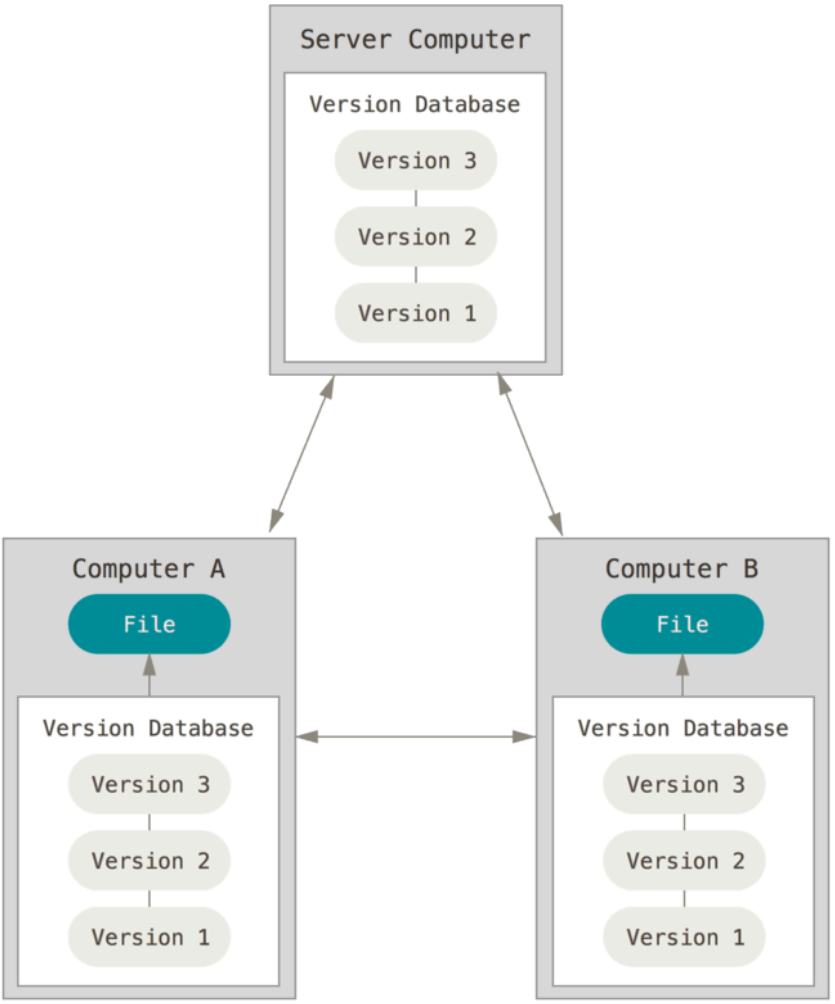
**Dependencia del servidor:** Si el servidor central falla, puede paralizar el trabajo de todo el equipo.

**Colaboración limitada:** La colaboración entre desarrolladores puede ser más complicada, especialmente si están trabajando en ramas separadas.

**Necesidad de conexión constante:** Los desarrolladores suelen necesitar una conexión constante al servidor central para realizar operaciones de control de versiones.

## Sistema distribuido:

Los sistemas de Control de Versiones Distribuidos (DVCS por sus siglas en inglés) ofrecen soluciones para los problemas que han sido mencionados. En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos.



Además, muchos de estos sistemas se encargan de manejar numerosos repositorios remotos con los cuales pueden trabajar, de tal forma que puedes colaborar simultáneamente con diferentes grupos de personas en distintas maneras dentro del mismo proyecto. Esto permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

***Ventajas:***

**Desconexión:** Los desarrolladores pueden trabajar de forma independiente y realizar cambios sin estar conectados al repositorio central.

**Ramas locales:** Los desarrolladores pueden crear ramas locales fácilmente para probar nuevas características sin afectar al código principal.

**Respaldo local:** Cada desarrollador tiene una copia completa del repositorio, lo que proporciona una capa adicional de respaldo.

**Rendimiento:** Al distribuir la carga entre diferentes repositorios, los sistemas distribuidos pueden ofrecer mejor rendimiento en entornos con muchos usuarios.

***Desventajas:***

**Mayor complejidad:** Pueden ser más complejos de entender y utilizar, especialmente para equipos nuevos en el control de versiones.

**Conflicto de sincronización:** Pueden surgir conflictos durante la sincronización entre repositorios, especialmente en equipos grandes o con cambios frecuentes.

**Gestión de ramas:** La gestión de ramas distribuidas puede ser más complicada, ya que cada desarrollador puede tener su propia rama local.

**Más propenso a bifurcaciones:** Debido a la facilidad para crear ramas locales, puede haber una proliferación de bifurcaciones que pueden dificultar la integración de cambios.

## *Conclusión:*

La elección entre uno u otro depende en gran medida de las necesidades y preferencias del equipo de desarrollo. Los **sistemas centralizados** son más simples y adecuados para equipos pequeños con requisitos básicos de control de versiones, ofreciendo una gestión más centralizada y un control de acceso más sencillo. Por otro lado, los **sistemas distribuidos** proporcionan mayor flexibilidad y capacidad de trabajo descentralizado, permitiendo a los desarrolladores trabajar de forma independiente y realizar cambios sin conexión. Sin embargo, esta flexibilidad puede venir acompañada de una mayor complejidad en la gestión y sincronización de los repositorios. En última instancia, la elección entre sistemas centralizados y distribuidos dependerá de factores como el tamaño del equipo, la naturaleza del proyecto y las preferencias de los desarrolladores en términos de flujo de trabajo y control de versiones.