Control de versiones con historial de cambios para el buen desarrollo en equipo: Git, Subversion y Mercurial

Cardenal Gardoki, 1
48008 BILBAO (Vizcaya)
Teléfono: 902 012 199
www.hostalia.com

HOSTALIA .com

A la hora de desarrollar algún tipo de software o aplicación web, si queremos que éste sea de calidad, es habitual que varias personas trabajen en él de forma simultánea, cada uno centrado en una determinada tarea como puede ser el frontend, backend, base de datos…

Debido a esto, es importante contar con una herramienta que sea capaz de organizar al equipo y que nos permita disponer en todo momento de los cambios que han realizado el resto del equipo en el proyecto. Es aquí donde entran en juego los sistemas de control de versiones de los cuales hablaremos a lo largo de este libro blanco.

Qué es un sistema de control de versiones

Un sistema de control de versiones (Version Control System o VCS en inglés) es una herramienta que permite controlar los cambios que se van realizando en el proyecto que estamos desarrollando, manteniendo de forma sencilla un historial de los cambios llevados a cabo. Es útil para guardar cualquier documento o proyecto que cambie con frecuencia.

Los sistemas de control de versiones son la mejor herramienta para trabajar en equipo ya que en todo momento tendremos un control completo del proceso de desarrollo, pudiendo identificar en cada momento la fecha en la que se modificó un determinado archivo, quién hizo ese cambio o realizar comparaciones a lo largo del tiempo.

¿Cómo funciona un sistema de control de versiones?

La forma de trabajar con este tipo de herramientas es sencilla. Lo primero que se hace es obtener una copia del proyecto de un repositorio (ver glosario en página 6). El programador realiza los cambios necesarios en el código fuente para la tarea que le han encomendado. Una vez que esos cambios están listos, son enviados al servidor mediante el uso de un programa cliente, de modo que el resto de usuarios que trabajan en el proyecto se los pueden descargar en cualquier momento, además de guardar los cambios realizados por cada usuario.

Puede darse la situación de que varios programadores tengan que trabajar sobre el mismo archivo. Si es así, este tipo de sistemas detectan esta situación, y actúan para evitar posibles problemas, pudiendo darse dos casos.

1. Si los programadores han trabajado en partes de código diferentes, el sistema lo detectará y los fusionará para dar un fichero que incluya todos los cambios.

2. Si los programadores han trabajado en líneas de código comunes, realizando cambios sobre ellas, el sistema lo que hace es señalar que se ha producido un conflicto, creando un archivo intermedio para que puedan ser revisados los cambios de forma simultánea, y que de esta forma se pueda decidir con qué versión quedarse o bien realizar a mano una combinación de los dos.

Ventajas de los sistemas de control de versión

A continuación os pasamos a detallar algunas de las ventajas que ofrece utilizar este tipo de herramientas.

- Ofrece la posibilidad de tener un número ilimitado de personas trabajando sobre un mismo código.
- Retorno a un estado anterior. El uso de sistemas de versiones permite poder volver a un estado anterior de un fichero o de todo el proyecto en caso de que los cambios realizados a posteriori hayan dado problemas.
- Historial de los cambios realizados, pudiendo saber en todo momento quién y en qué momento se hizo un determinado cambio.
- Crear una determinada rama para probar una cosa o solucionar un determinado bug encontrado sin comprometer lo que ya llevas realizado.
- Acceso remoto. Es posible acceder al repositorio donde se encuentra el proyecto desde cualquier equipo capaz de conectarse a la red.
- Mejora de la seguridad, ya que se pueden otorgar diferentes permisos sobre cada rama del proyecto. Así se podría dar acceso de lectura a todos los usuarios, pero sólo a unos pocos permisos de escritura.

Clasificación de los sistemas de control de versiones

El uso de sistemas de control de versiones no es algo nuevo, sino que es algo que se ha utilizado desde hace años. Los usuarios, a través del tiempo, han usado diversos métodos para compartir su código fuente en un ambiente de desarrollo colaborativo.

Hasta llegar a los métodos actuales, se han utilizados ciertos métodos con mejor o peor resultado. Entre estos métodos podemos destacar:

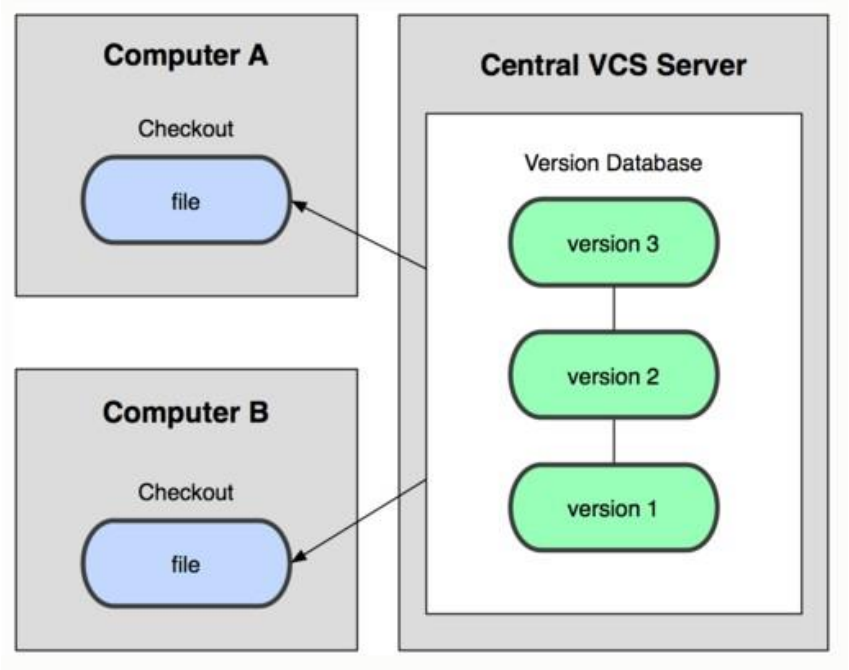
- Mantener una copia en equipos o servidores diferentes. Este tipo de acciones ofrece un buen resultado para mantener a salvo el trabajo, pero ofrece importantes desventajas a la hora de guardar versiones. En cada modificación que hagamos sobre el código, nosotros seremos los encargados de realizar una copia de la versión anterior a los cambios, así como de la nueva versión.
- Desarrollo de aplicaciones propias para la gestión de versiones. Se trata de una técnica totalmente desaconsejada ya que estaríamos reinventado la rueda, cuando podemos hacer uso de varias herramientas Open Source capaz de realizar todas las tareas necesarias para controlar nuestras versiones.
- Guardar los proyectos en sistemas de almacenamiento en la nube. Este tipo de aplicaciones puede funcionar como herramienta de control de versiones, aunque con importantes limitaciones respecto a otros sistemas creados expresamente para este fin como pueden ser Subversion, Git o Mercurial.

Hoy en día, los desarrolladores han dejado de utilizar esos sistemas a favor de otras herramientas que ofrecen muchas más funcionalidades, herramientas que podemos clasificar en dos grupos:

a) Sistemas de control de versiones centralizados

En este tipo de estructura existe un servidor donde es alojado el repositorio del proyecto y al que los distintos usuarios acceden para recuperar o actualizar los cambios realizados. Algunos ejemplos de herramientas de control de versiones centralizadas son CVS y Subversion.

En la imagen siguiente podéis ver el esquema de un sistema centralizado.



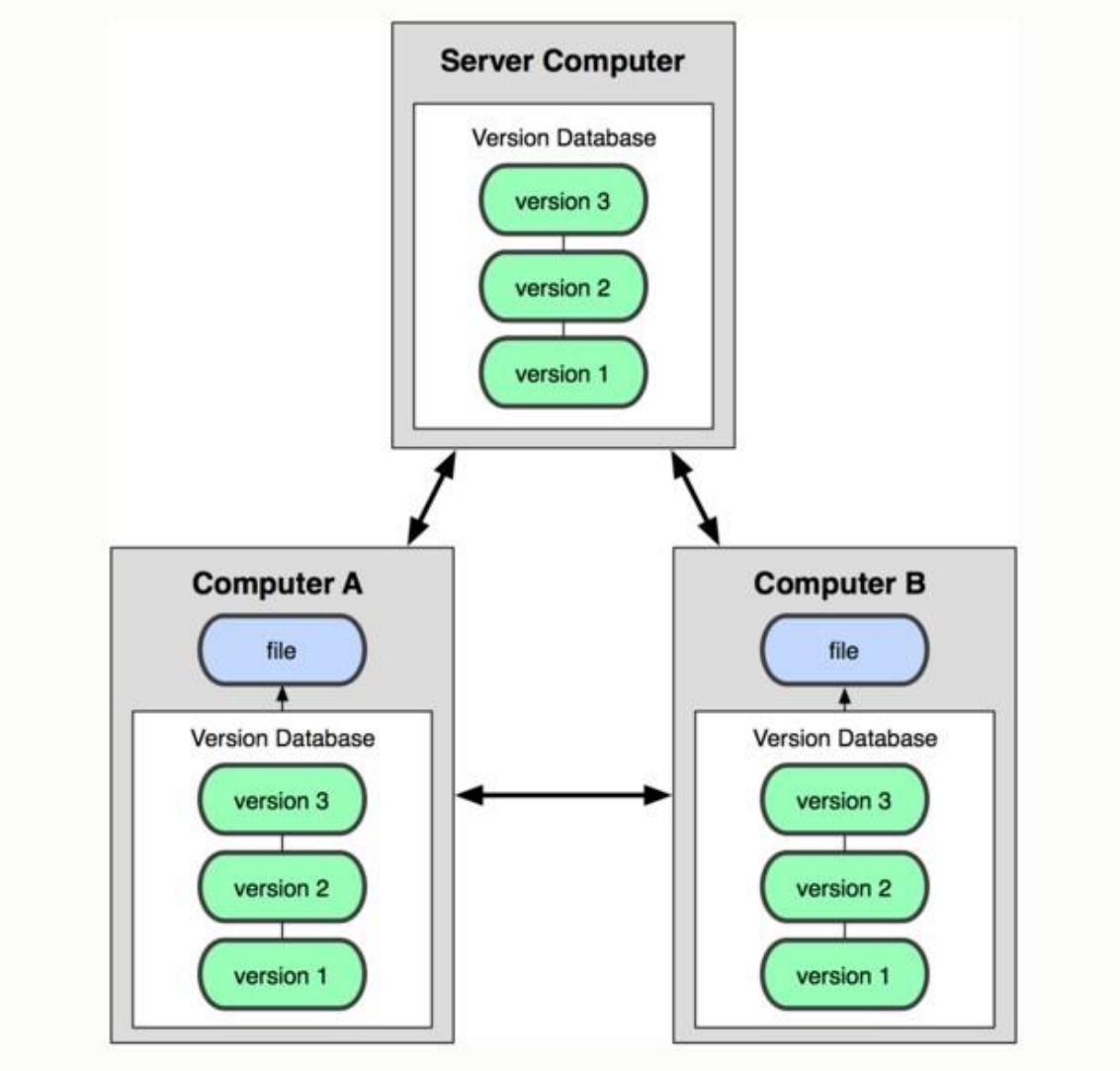
Este sistema tiene la ventaja de ofrecer un mayor control a los administradores, que en todo momento pueden controlar qué es lo que pueden hacer el resto de usuarios.

Por el contrario, tiene una gran desventaja, y es que dependemos al 100% del buen funcionamiento del servidor. Una caída del servidor haría que nadie pudiera descargar o guardar cambios versionados. Además, si el problema fuese mayor y el disco duro se estropeara y no hemos realizado copia de seguridad, perderemos todo el historial del desarrollo, cosa que se podía solucionar con un backup.

b) Sistemas de control de versiones distribuidos

La principal diferencia con el sistema centralizado es que en este caso no hay un único servidor que mantenga una copia del repositorio, sino que cada usuario tiene su propia copia de todo el historial. Además, la disponibilidad de un servidor donde alojar el historial se convierte en algo opcional. Un ejemplo de herramienta de esta estructura es Git.

En la imagen siguiente podéis como sería el esquema de esta estructura.



En el caso de que este servidor fallara, no se perdería la información como ocurre en el caso del sistema centralizado, ya que cualquier repositorio de los clientes puede copiarse en el servidor para restaurarlo. Además, mientras el servidor está caído, los usuarios podrán seguir trabajando sin problemas.

Terminología utilizada en los sistemas de control de versiones

Dependiendo de la herramienta de control de versiones que utilicemos, la terminología utilizada puede variar, pero hay ciertos términos que son comunes a todas las herramientas. Veamos los más importantes:

1. **Repositorio**
   Se trata del lugar donde se almacenan los datos actualizados y todos los históricos que forman parte del proyecto. Normalmente está alojado en un servidor.

2. **Commit**
   Se trata de la acción que realiza un cambio en el proyecto y que es almacenado en el repositorio, para que el resto de colaboradores pueda acceder a él.

3. **Mensaje de registro**

Se trata de un mensaje descriptivo que acompaña a cada commit realizado y que sirve para explicar los cambios que se han realizado en esa actualización. De esta forma todos los usuarios sabrán mediante ese mensaje qué es lo que se ha cambiado.

4. **Update**
   Se trata de la acción que permite solicitar los cambios (commits) que han realizado otros en la copia local del proyecto. Esto actualizará la copia a la última versión.

5. **Checkout**
   Es el proceso de obtener una copia del proyecto desde el repositorio. Se puede especificar una determinada copia. Por defecto se suele obtener la última.

6. **Diff**
   Este término hace referencia a cambios. Mediante un diff se pueden ver las líneas de código que han sido modificadas de forma resaltada.

7. **Branch o rama**
   Es una copia del proyecto, bajo el control de versiones, pero aislado, de forma que los cambios realizados en esta rama no afecten al resto del proyecto y viceversa.

Herramientas de control de versiones

1.- Git

Git es un software de control de versiones diseñado por Linus Tolvards. Está encuadrado dentro de los sistemas de gestión de versiones distribuidas y en un principio fue desarrollado para controlar el desarrollo del kernel del sistema operativo Linux, aunque posteriormente fue lanzado para que cualquier usuario pudiera utilizarlo para la gestión de sus proyectos.

Entre sus ventajas podemos destacar que se trata de un software que permite la creación de un proyecto nuevo a partir de una ramificación, además de una gestión rápida y sencilla de las versiones de un proyecto.

En su contra podemos destacar que su aprendizaje puede ser un poco tedioso, más que algunas aplicaciones similares existentes en el mercado. Algunos proyectos que hacen uso de Git son Linux, Eclipse, Android o Debian.

2.- Subversion

Subversion es un software de gestión de versiones pertenecientes al grupo de los centralizados y que es desarrollado como un proyecto de Apache Software Foundation, por lo que cuenta con una gran comunidad detrás de él.

Entre las ventajas que ofrece esta herramienta destaca su fácil implementación en un servidor Apache y su bajo coste a la hora de crear nuevas ramas. En su contra podemos destacar la complejidad de gestionar un proyecto que dispone de muchas ramas.

Entre los proyectos que hacen uso de Subversion podemos destacar Momo, Ruby o FreeBSD.

3.- Mercurial

Al igual que Git, se trata de un software de gestión de versiones del tipo distribuido. Está implementado en su mayor parte mediante el lenguaje de programación Python, aunque la implementación de “diff” está hecha en C para obtener un mayor rendimiento.

Las principales características de esta herramienta son que ofrece un gran rendimiento y escalabilidad, además de ser muy sencillo su mantenimiento y disponer de una interfaz web. Por otro lado, la comunidad de desarrollo que tiene detrás de sí es pequeña, si la comparamos con otros proyectos similares como Subversion.

Entre los proyectos que utilizan Mercurial están Mozilla o Python.

Si nunca has utilizado un sistema de control de versión, empezar con ello puede parecerte algo tedioso, pero con el paso del tiempo irás descubriendo las ventajas y beneficios que ofrecen a la hora de desarrollar aplicaciones.