

4. 6.4 [Reescribiendo la historia](#)
5. 6.5 [Depuración con Git](#)
6. 6.6 [Submódulos](#)
7. 6.7 [Fusión de subárboles](#)
8. 6.8 [Recapitulación](#)

## 1. **7. Personalizando Git**

1. 7.1 [Configuración de Git](#)
2. 7.2 [Atributos de Git](#)
3. 7.3 [Puntos de enganche Git](#)
4. 7.4 [Un ejemplo de implantación de una determinada política en Git](#)
5. 7.5 [Recapitulación](#)

## 2. **8. Git y Otros Sistemas**

1. 8.1 [Git y Subversion](#)
2. 8.2 [Migrating to Git](#)
3. 8.3 [Summary](#)

## 3. **9. Los entresijos internos de Git**

1. 9.1 [Fontanería y porcelana](#)
2. 9.2 [Los objetos Git](#)
3. 9.3 [Referencias Git](#)
4. 9.4 [Archivos empaquetadores](#)
5. 9.5 [Las especificaciones para hacer referencia a... \(refspec\)](#)
6. 9.6 [Protocolos de transferencia](#)
7. 9.7 [Mantenimiento y recuperación de datos](#)
8. 9.8 [Recapitulación](#)

1st Edition

# 1.1 Empezando - Acerca del control de versiones

## Acerca del control de versiones

¿Qué es el control de versiones, y por qué debería importarte? El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. A pesar de que los ejemplos de este libro

muestran código fuente como archivos bajo control de versiones, en realidad cualquier tipo de archivo que encuentres en un ordenador puede ponerse bajo control de versiones.

Si eres diseñador gráfico o web, y quieres mantener cada versión de una imagen o diseño (algo que sin duda quieres), un sistema de control de versiones (Version Control System o VCS en inglés) es una elección muy sabia. Te permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más. Usar un VCS también significa generalmente que si fastidias o pierdes archivos, puedes recuperarlos fácilmente. Además, obtienes todos estos beneficios a un coste muy bajo.

## Sistemas de control de versiones locales

Un método de control de versiones usado por mucha gente es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son avispados). Este enfoque es muy común porque es muy simple, pero también tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras, y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para hacer frente a este problema, los programadores desarrollaron hace tiempo VCSs locales que contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre los archivos (véase Figura 1-1).

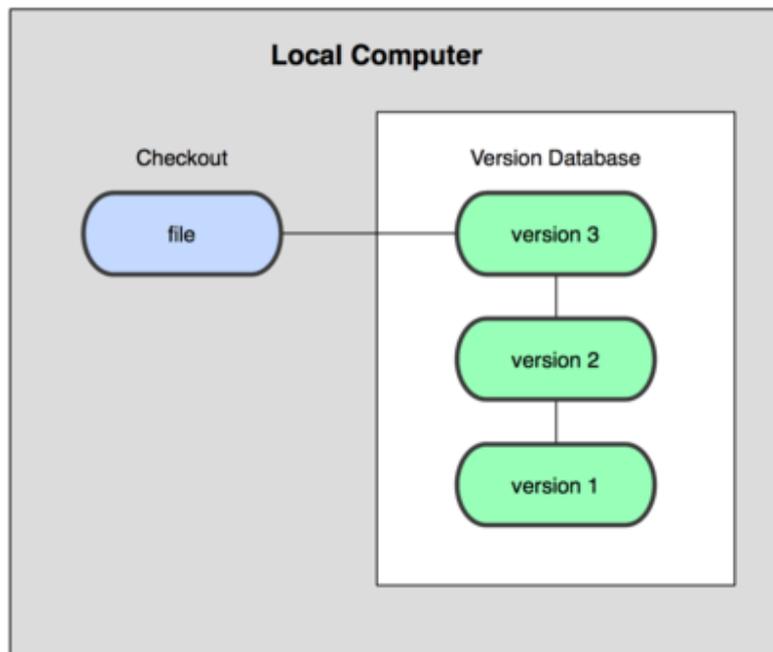


Figura 1-1. Diagrama de control de versiones local.

Una de las herramientas de control de versiones más popular fue un sistema llamado rcs, que todavía podemos encontrar en muchos de los ordenadores actuales. Hasta el famoso sistema operativo Mac OS X incluye el comando rcs cuando instalas las herramientas de desarrollo. Esta herramienta funciona básicamente guardando conjuntos de parches (es decir, las diferencias entre archivos) de una versión a otra en un formato especial en disco; puede entonces recrear cómo era un archivo en cualquier momento sumando los distintos parches.

## Sistemas de control de versiones centralizados

El siguiente gran problema que se encuentra la gente es que necesitan colaborar con desarrolladores en otros sistemas. Para solventar este problema, se desarrollaron los sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCSs en inglés). 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. Durante muchos años éste ha sido el estándar para el control de versiones (véase Figura 1-2).

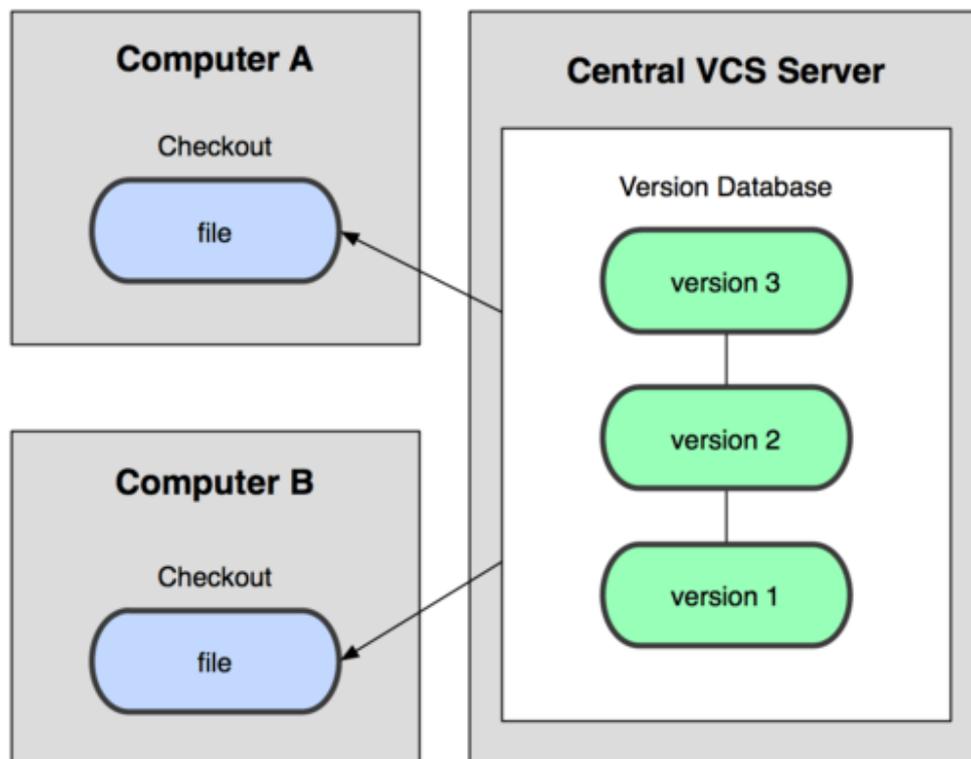


Figura 1-2. Diagrama de control de versiones centralizado.

Esta configuración ofrece muchas ventajas, especialmente frente a VCSs

locales. Por ejemplo, todo el mundo puede saber (hasta cierto punto) en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado de qué puede hacer cada uno; 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 puede colaborar o guardar cambios versionados de aquello en que están trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han llevado copias de seguridad adecuadamente, pierdes absolutamente todo —toda la historia del proyecto salvo aquellas instantáneas que la gente pueda tener en sus máquinas locales. Los VCSs locales sufren de este mismo problema— cuando tienes toda la historia del proyecto en un único lugar, te arriesgas a perderlo todo.

## **Sistemas de control de versiones distribuidos**

Es aquí donde entran los sistemas de control de versiones distribuidos (Distributed Version Control Systems o DVCSs en inglés). En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no sólo descargan la última instantánea de los archivos: replican completamente el repositorio. Así, si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los datos (véase Figura 1-3).

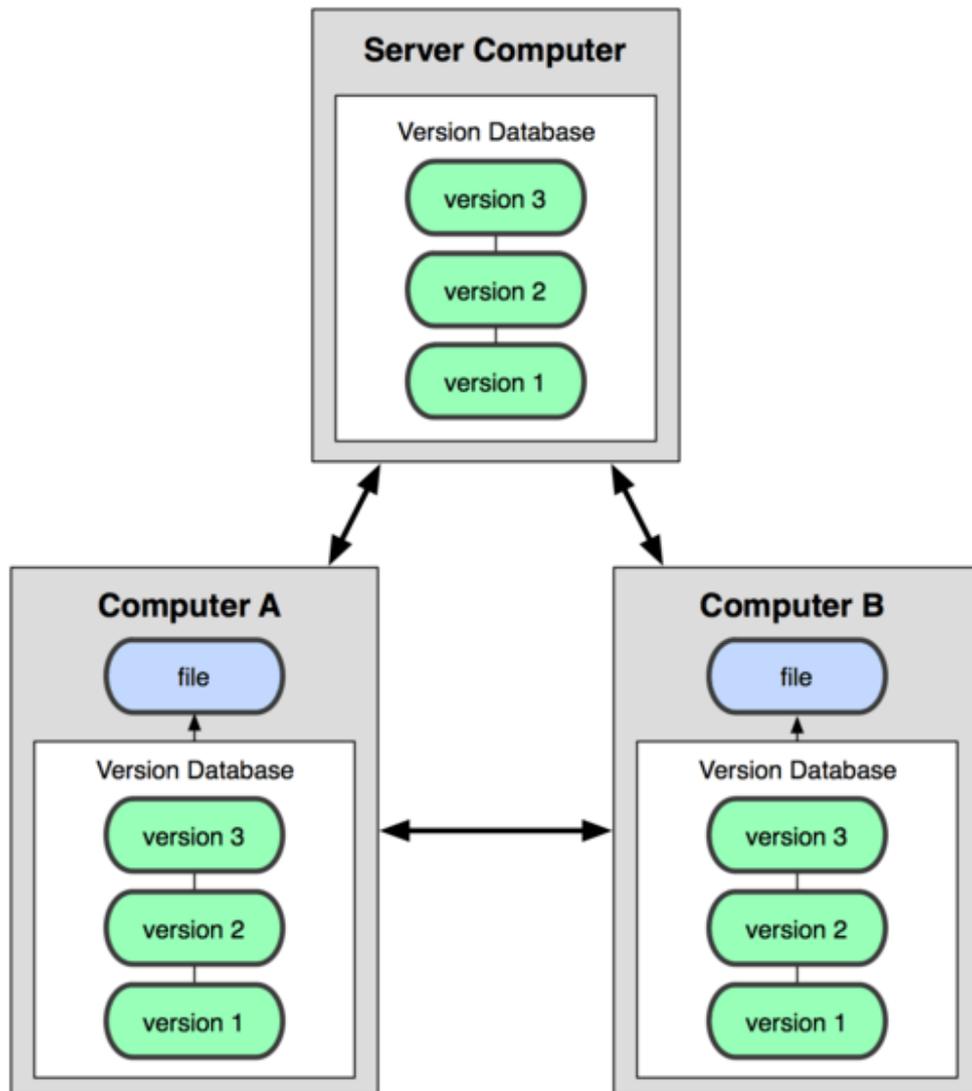


Figura 1-3. Diagrama de control de versiones distribuido.

Es más, muchos de estos sistemas se las arreglan bastante bien teniendo varios repositorios con los que trabajar, por lo que puedes colaborar con distintos grupos de gente simultáneamente dentro del mismo proyecto. Esto te permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

[prev](#) | [next](#)

This [open sourced](#) site is [hosted on GitHub](#).

Patches, suggestions and comments are welcome.

Git is a member of [Software Freedom Conservancy](#)