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).

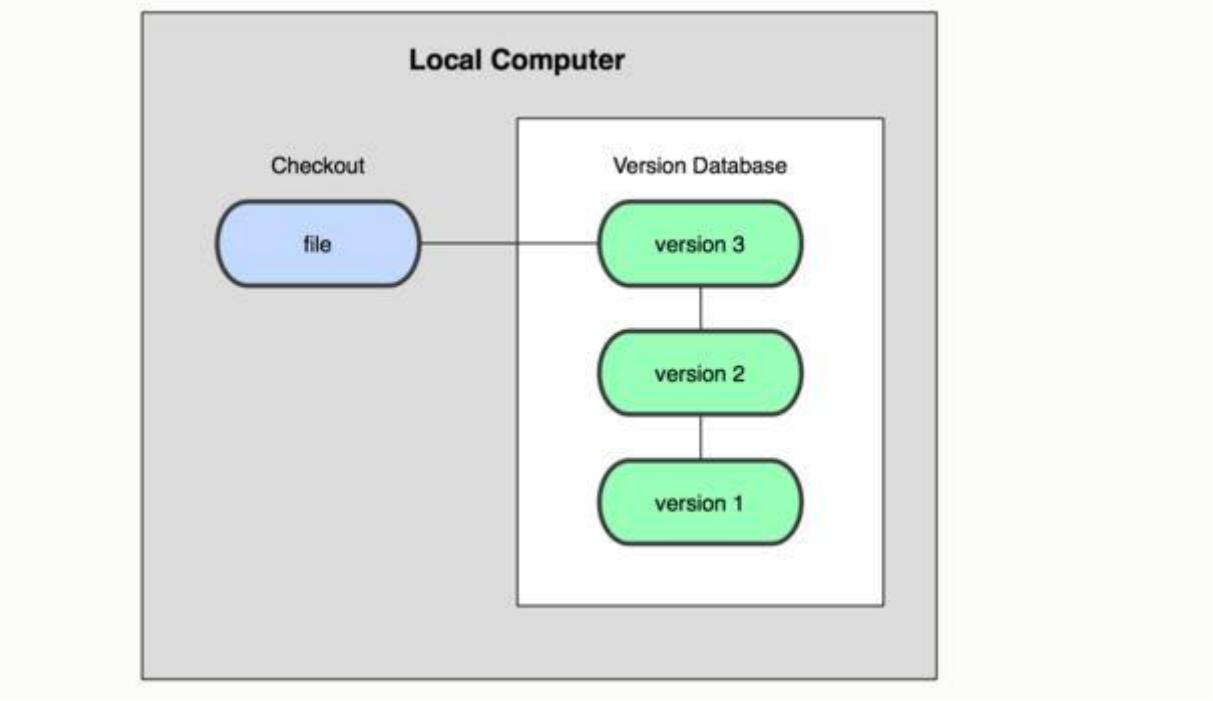

Figura 1-1. Diagrama de control de versiones local.

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).

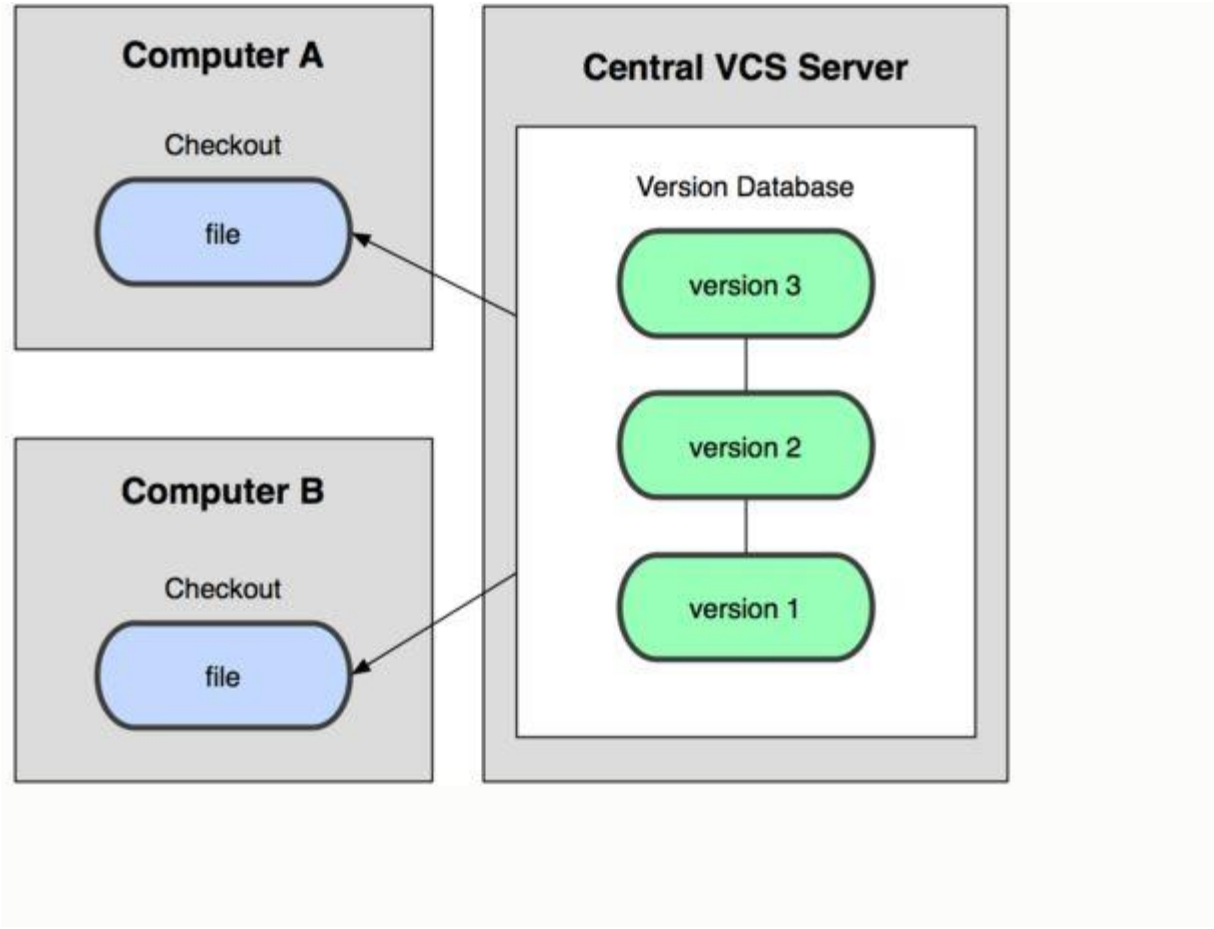

Figura 1-2. Diagrama de control de versiones centralizado.

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).

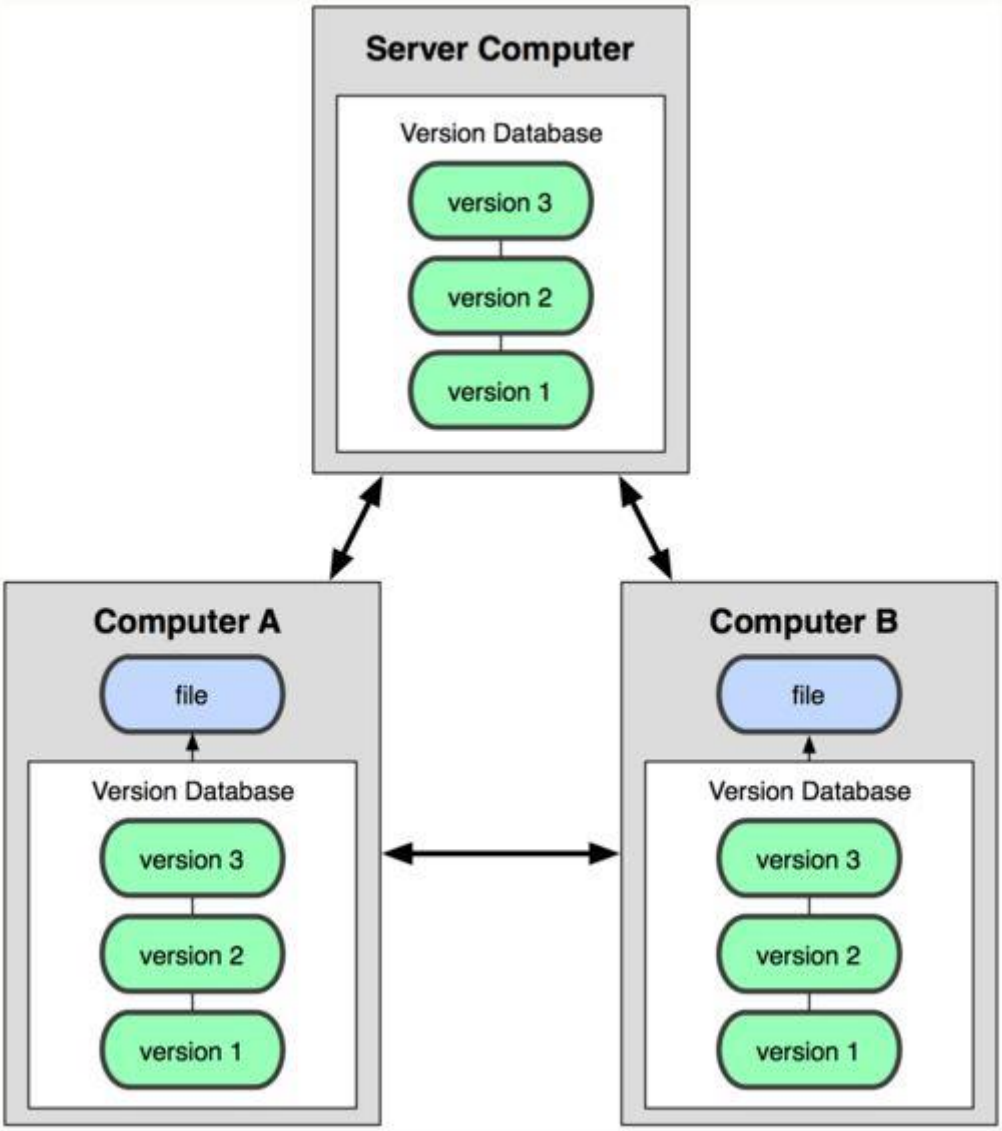

Figura 1-3. Diagrama de control de versiones distribuido.

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.

Terminología:

Repositorio

El **repositorio** es el lugar en el que se almacenan los datos actualizados e históricos de cambios, a menudo en un servidor. A veces se le denomina depósito o depot. Puede ser un sistema de archivos en un disco duro, un banco de datos, etc..

Módulo

Conjunto de directorios y/o archivos dentro del repositorio que pertenecen a un proyecto común.

Revisión ("version")Version Control System

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). A la última versión se le suele identificar de forma especial con el nombre de **HEAD**. Para marcar una revisión concreta se usan los **rótulos** o **tags**

Rotular ("tag")

Darle a alguna versión de cada uno de los ficheros del módulo en desarrollo en un momento preciso un nombre común ("etiqueta" o "rótulo") para asegurarse de reencontrar ese estado de desarrollo posteriormente bajo ese nombre. En la práctica se rotula a todos los archivos en un momento determinado. Para eso el módulo se "congela" durante el rotulado para imponer una versión coherente. Pero bajo ciertas circunstancias puede ser necesario utilizar versiones de algunos ficheros que no coinciden temporalmente con las de los otros ficheros del módulo. 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. En algunos sistemas se considera un tag como una rama en la que los ficheros no evolucionan, están congelados.

Línea base **("Baseline")**

Una revisión aprobada de un documento o fichero fuente, a partir del cual se pueden realizar cambios subsiguientes.

Abrir rama ("branch") o ramificar

Un módulo puede ser **branched** o **bifurcado** en un instante de tiempo de forma que, desde ese momento en adelante se tienen dos copias (ramas) que evolucionan de forma independiente siguiendo su propia línea de desarrollo. El módulo tiene entonces 2 (o más) "ramas". La ventaja es que se puede hacer un "merge" de las modificaciones de ambas ramas, posibilitando la creación de "ramas de prueba" que contengan código para evaluación, si se decide que las modificaciones realizadas en la "rama de prueba" sean preservadas, se hace un "merge" con la rama principal. Son motivos habituales para la creación de ramas la creación de nuevas funcionalidades o la corrección de errores.

Desplegar ("Check-out", "checkout", "co")

Un despliegue crea una copia de trabajo local desde el repositorio. Se puede especificar una revisión concreta, y predeterminadamente se suele obtener la última.

"Publicar" o "Enviar" ("commit", "check-in", "ci", "install", "submit")

Un **commit** sucede cuando una copia de los cambios hechos a una copia local es escrita o integrada sobre el repositorio.

Conflicto

Un conflicto ocurre en las siguientes circunstancias:

1. Los usuarios *X* e *Y* despliegan versiones del *archivo A* en que las líneas *n1* hasta *n2* son comunes.
2. El usuario *X* envía cambios entre las líneas *n1* y *n2* al *archivo A*.
3. El usuario *Y* no actualiza el *archivo A* tras el envío del usuario *X*.
4. El usuario *Y* realiza cambios entre las líneas *n1* y *n2*.
5. El usuario *Y* intenta posteriormente enviar esos cambios al *archivo A*.

El sistema es incapaz de fusionar los cambios. El usuario *Y* debe *resolver* el conflicto combinando los cambios, o eligiendo uno de ellos para descartar el otro.

Resolver

El acto de la intervención del usuario para atender un conflicto entre diferentes cambios al mismo archivo.

Cambio ("change", "diff", "delta")

Un **cambio** representa una modificación específica a un archivo bajo control de versiones. La granularidad de la modificación considerada un cambio varía entre diferentes sistemas de control de versiones.

Lista de cambios ("changelist", "change set", "patch")

En muchos sistemas de control de versiones con commits multi-cambio atómicos, una lista de cambios identifica el conjunto de cambios hechos en un único commit. Esto también puede representar una vista secuencial del código fuente, permitiendo que el fuente sea examinado a partir de cualquier identificador de lista de cambios particular.

Exportación ("export")

Una exportación es similar a un **check-out**, salvo porque crea un árbol de directorios limpio sin los metadatos de control de versiones presentes en la copia de trabajo. Se utiliza a menudo de forma previa a la publicación de los contenidos.

Importación ("import")

Una importación es la acción de copia un árbol de directorios local (que no es en ese momento una copia de trabajo) en el repositorio por primera vez.

Integración o fusión ("merge")

Una **integración** o **fusión** une dos conjuntos de cambios sobre un fichero o un conjunto de ficheros en una revisión unificada de dicho fichero o ficheros.

- Esto puede suceder cuando un usuario, trabajando en esos ficheros, **actualiza** su copia local con los cambios realizados, y añadidos al repositorio, por otros usuarios. Análogamente, este mismo proceso puede ocurrir en el repositorio cuando un usuario intenta **check-in** sus cambios.
- Puede suceder después de que el código haya sido **branched**, y un problema anterior al branching sea arreglado en una rama, y se necesite incorporar dicho arreglo en la otra.
- Puede suceder después de que los ficheros hayan sido **branched**, desarrollados de forma independiente por un tiempo, y que entonces se haya requerido que fueran fundidos de nuevo en un único trunk unificado.

Integración inversa

El proceso de fundir ramas de diferentes equipos en el trunk principal del sistema de versiones.

Actualización (“sync” ó “update”)

Una **actualización** integra los cambios que han sido hechos en el repositorio (por ejemplo por otras personas) en la **copia de trabajo** local.

Copia de trabajo (“workspace”)

La **copia de trabajo** es la copia local de los ficheros de un repositorio, en un momento del tiempo o revisión específicos. Todo el trabajo realizado sobre los ficheros en un repositorio se realiza inicialmente sobre una copia de trabajo, de ahí su nombre. Conceptualmente, es un **cajón de arena** o **sandbox**.

Congelar

Congelar significa permitir los últimos cambios (commits) para solucionar las fallas a resolver en una entrega (release) y suspender cualquier otro cambio antes de una entrega, con el fin de obtener una versión consistente. Si no se congela el repositorio, un desarrollador podría comenzar a resolver una falla cuya resolución no está prevista y cuya solución dé lugar a efectos colaterales imprevistos.