
This book is available in English.

Full translation available in

български език,
Español,
Français,
日本語,
한국어,
Nederlands,
Русский,
Українська
简体中文,

Partial translations available in

Čeština,
Indonesian,
Polski,
Српски,
Tagalog,
繁體中文,

Translations started for

Беларуская,
Deutsch,
فارسی,
Ελληνικά,

Italiano,
Македонски,
Bahasa Melayu,
Polski,
Português (Brasil),
Türkçe,
Ўзбекча.

The source of this book is hosted on GitHub.
Patches, suggestions and comments are welcome.

Chapters ▾

1st Edition

.1 Los geht’s - Wozu Versionskontrolle?

Wozu Versionskontrolle?

Was ist die Versionskontrolle, und warum solltest Du Dich dafür interessieren? Versionskontrollsysteme (VCS) protokollieren Änderungen an einer Datei oder einer Anzahl von Dateien über die Zeit hinweg, so dass man zu jedem Zeitpunkt auf Versionen und Änderungen zugreifen kann. Die Beispiele in diesem Buch verwenden Software Quellcode, tatsächlich aber kannst Du Änderungen an praktisch jeder Art von Datei per Versionskontrolle nachverfolgen.

Als ein Grafik- oder Webdesigner zum Beispiel willst Du in der Lage sein, jede Version eines Bildes oder Layouts zurückverfolgen zu können. Es wäre daher sehr ratsam, ein Versionskontrollsystem zu verwenden. Ein solches System erlaubt Dir, einzelne Dateien oder auch ein ganzes Projekt in einen früheren Zustand zurückzuversetzen, nachzuvollziehen, wer zuletzt welche Änderungen vorgenommen hat, die möglicherweise Probleme verursachen, wer eine Änderung ursprünglich vorgenommen hat usw. Ein Versionskontrollsystem für Deine Arbeit zu verwenden, versetzt Dich in die Lage jederzeit zu einem vorherigen, funktionierenden Zustand zurückzugehen, wenn Du vielleicht Mist gebaut oder aus irgendeinem Grunde Dateien verloren hast. All diese Vorteile erhältst Du für einen nur sehr geringen, zusätzlichen Aufwand.

Lokale Versionskontrollsysteme

Viele Leute kontrollieren Versionen ihrer Arbeit, indem sie einfach Dateien in ein anderes Verzeichnis kopieren (wenn sie clever sind: ein Verzeichnis mit einem Zeitstempel im Namen). Diese Vorgehensweise ist üblich, weil sie so einfach ist. Aber sie ist auch unglaublich fehleranfällig. Man vergisst sehr leicht, in welchem Verzeichnis man sich gerade befindet und kopiert die falschen Dateien oder überschreibt Dateien, die man eigentlich nicht überschreiben wollte.

Um diese Arbeit zu erleichtern und sicherer zu machen, haben Programmierer vor langer Zeit Versionskontrollsysteme entwickelt, die alle Änderungen an allen relevanten Dateien in einer lokalen Datenbank verfolgten (siehe Bild 1-1).

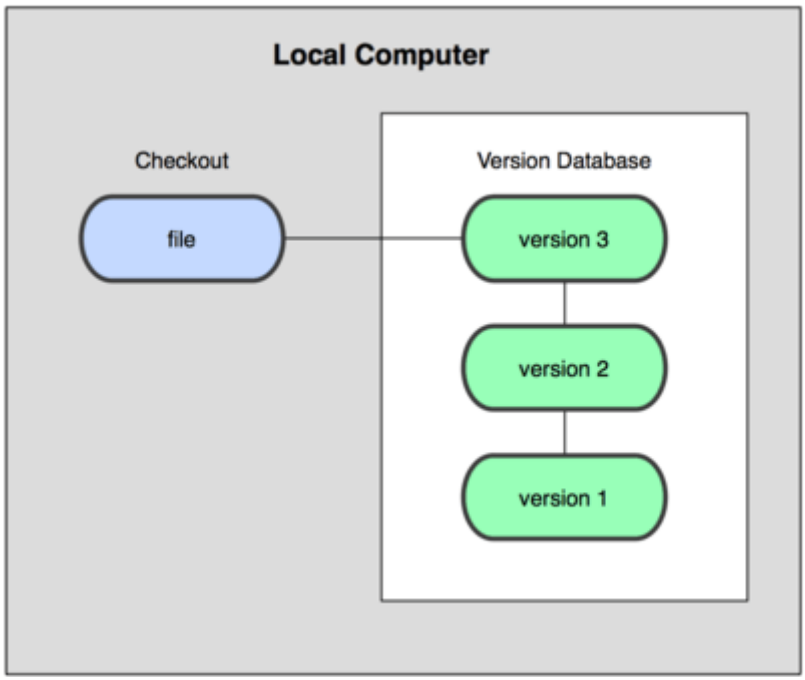

Bild 1-1. Diagramm: Lokale Versionskontrolle

Eines der populärsten Versionskontrollsysteme war rcs, und es wird heute immer noch mit vielen Computern ausgeliefert. Z.B. umfasst auch das Betriebssystem Mac OS X den Befehl rcs, wenn Du die Developer Tools installierst. Dieser Befehl arbeitet nach dem Prinzip, dass er für jede Änderung einen Patch (d.h. eine Kodierung der Unterschiede, die eine Änderung an einer oder mehreren Dateien umfasst) in einem speziellen Format in einer Datei auf der Festplatte speichert.

Zentralisierte Versionskontrollsysteme

Das nächste Problem, mit dem Programmierer sich dann konfrontiert sahen, bestand in der Zusammenarbeit mit anderen: Änderungen an dem gleichen Projekt mussten auf verschiedenen Computern, möglicherweise verschiedenen Betriebssystemen vorgenommen werden können. Um dieses Problem zu lösen, wurden Zentralisierte Versionskontrollsysteme (CVCS) entwickelt. Diese Systeme, beispielsweise CVS, Subversion und Perforce, basieren auf einem zentralen Server, der alle versionierten Dateien verwaltet. Wer an diesen Dateien arbeiten will, kann sie von diesem Server abholen („check out“ xxx), auf seinem eigenen Computer bearbeiten und dann wieder auf dem Server abliefern. Diese Art von System war über viele Jahre hinweg der Standard für Versionskontrollsysteme (siehe Bild 1-2).

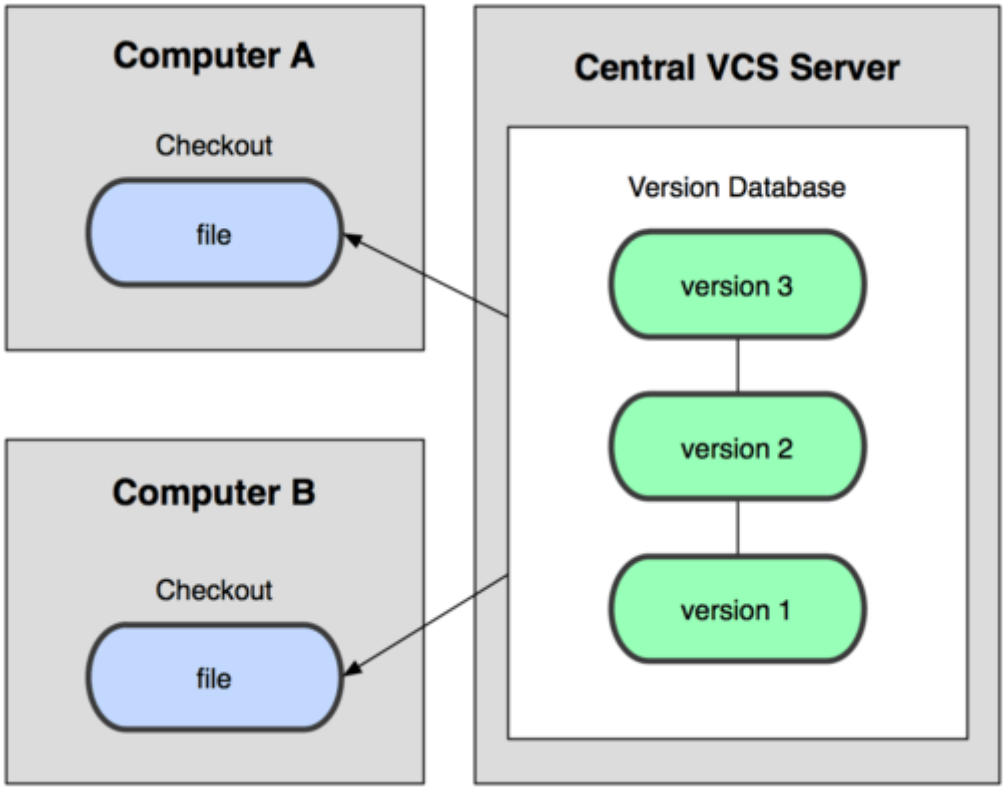

Bild 1-2. Diagramm: Zentralisierte Versionskontrollsysteme

Dieser Aufbau hat viele Vorteile gegenüber Lokalen Versionskontrollsystemen. Zum Beispiel weiß jeder mehr oder weniger genau darüber Bescheid, was andere, an einem Projekt Beteiligte gerade tun. Administratoren haben die Möglichkeit, detailliert festzulegen, wer was tun kann. Und es ist sehr viel einfacher, ein CVCS zu administrieren als lokale Datenbanken auf jedem einzelnen Anwenderrechner zu verwalten.

Allerdings hat dieser Aufbau auch einige erhebliche Nachteile. Der offensichtlichste Nachteil ist der „Single Point of Failure“, den der zentralisierte Server darstellt. Wenn dieser Server für nur eine Stunde nicht verfügbar ist, dann kann in dieser Stunde niemand in irgendeiner Form mit anderen arbeiten oder versionierte Änderungen an den Dateien speichern, an denen sie momentan arbeiten. Wenn die auf dem zentralen Server verwendete Festplatte beschädigt wird und keine Sicherheitskopien erstellt wurden, dann sind all diese Daten unwiederbringlich verloren – die komplette Historie des Projektes, abgesehen natürlich von dem jeweiligen Zustand, den Mitarbeiter gerade zufällig auf ihrem Rechner haben. Lokale Versionskontrollsysteme haben natürlich dasselbe Problem: wenn man die Historie eines Projektes an einer einzigen, zentralen Stelle verwaltet, riskiert man, sie vollständig zu verlieren, wenn irgendetwas an dieser zentralen Stelle ernsthaft schief läuft.

Verteilte Versionskontrollsysteme

Und an dieser Stelle kommen verteilte Versionskontrollsysteme (DVCS) ins Spiel. In einem DVCS (wie z.B. Git, Mercurial, Bazaar oder Darcs) erhalten Anwender nicht einfach den jeweils letzten Snapshot des

Projektes von einem Server: sie erhalten statt dessen eine vollständige Kopie des Repositories. Auf diese Weise kann, wenn ein Server beschädigt wird, jedes beliebige Repository von jedem beliebigen Anwenderrechner zurück kopiert werden und der Server so wieder hergestellt werden.

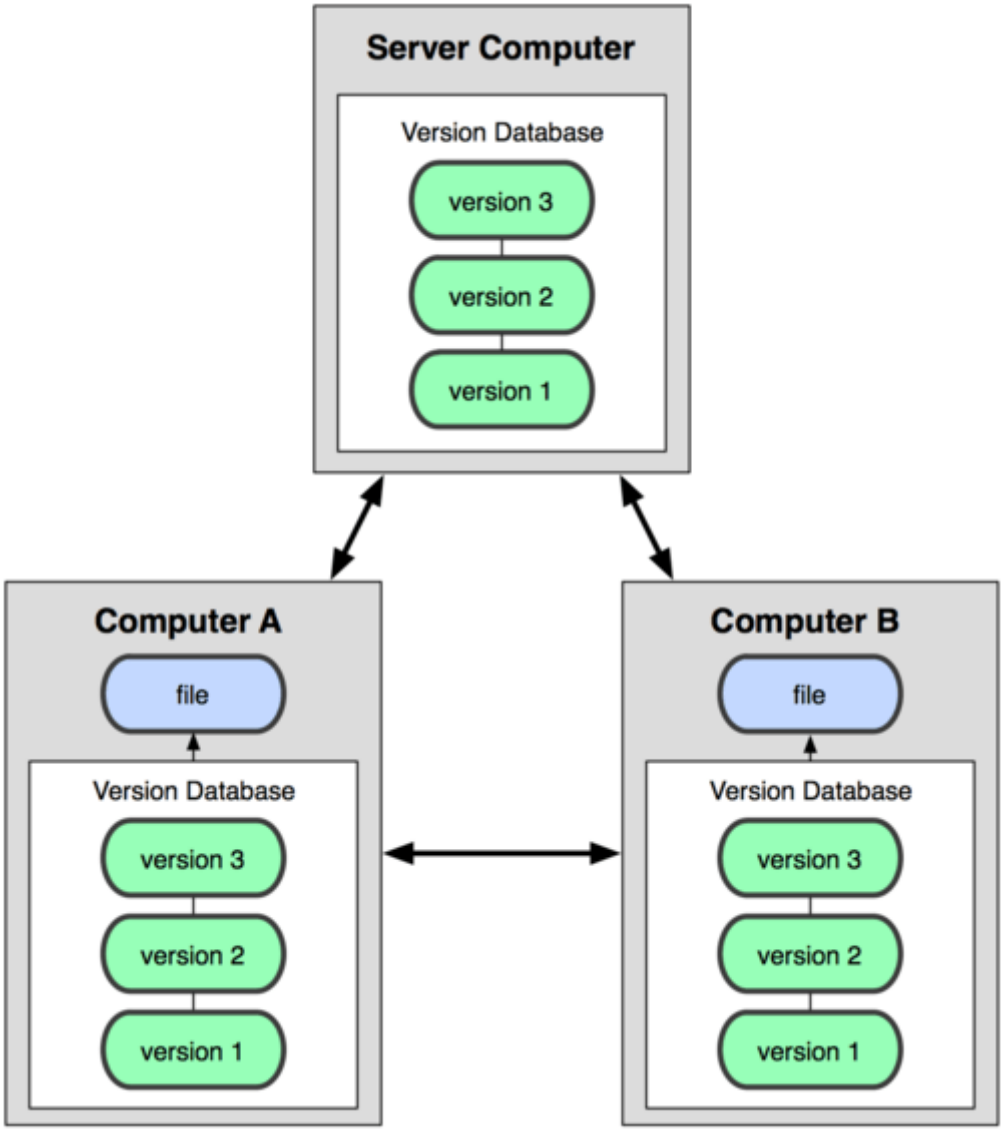

Bild 1-3. Diagramm: Distribuierte Versionskontrolle

Darüber hinaus können derartige Systeme hervorragend mit verschiedenen externen („remote“) Repositories umgehen, sodass man mit verschiedenen Gruppen von Leuten simultan in verschiedenen Weisen zusammenarbeiten kann. Das macht es möglich, verschiedene Arten von Arbeitsabläufen (wie Hierarchien) zu integrieren, was mit zentralisierten Systemen nicht möglich ist.

prev | next
About this site
Patches, suggestions, and comments are welcome.
Git is a member of Software Freedom Conservancy