

# CVS for Dummies

[Bernd Loechner, Andreas Jaeger](#)

## Inhaltsverzeichnis

- [Einleitung](#)
- [Motivation](#)
  - [Der Ansatz von CVS](#)
  - [Einsatz von CVS](#)
  - [Weitere Dokumentation](#)
- [Die wichtigsten Kommandos](#)
  - [Vorbereitungen](#)
  - [Änderungen synchronisieren](#)
  - [Neue Dateien registrieren](#)
  - [Dateien löschen](#)
  - [Versionsinformationen im Quelltext](#)
  - [Handhabung aus dem Emacs/XEmacs](#)
- [Ergänzungen](#)
  - [Einrichten eines globalen CVS-Verzeichnisses](#)
  - [Einrichten eines Projektes](#)
  - [Lokale und Remote CVS Verzeichnisse](#)
  - [Erstellen von Releases](#)

## Einleitung

Dateien wie Programmquellen oder Dokumentationen unterliegen einer hohen Änderungshäufigkeit. Hinzu kommt, daß oftmals viele Dateien in verschiedenen Versionen parallel von mehreren Entwicklern bearbeitet werden. Das Programm CVS unterstützt diese Aufgabe. Diese kleine Ausarbeitung soll neuen oder gelegentlichen Benutzern ohne tiefere Kenntnisse (sog. *Dummies*) den Umgang mit CVS erleichtern.

Dieser Artikel richtet sich an Anwender von CVS, er bietet keine vollständige Dokumentation, sondern eine Einführung der wichtigsten Konzepte und Befehle.

## Motivation

Ein typisches Phänomen bei Quelldateien zu Programmen oder Dokumentationen ist, daß sie über längere Zeiträume von verschiedenen Entwicklern immer wieder in meist kleinerem Umfang geändert werden. Hierbei hat man verschiedene Anforderungen: So sollen Änderungen, z.B. Löschungen von Textabschnitten, rückgängig gemacht werden können. Oder es soll die Fassung vom 29. April 2000 rekonstruiert werden. Oder es soll der Änderungsverlauf nachvollziehbar sein, warum also wann von wem welche Änderungen durchgeführt wurden.

Es sind also verschiedene Fassungen oder **Versionen** einer Datei und die Zusammenhänge zwischen ihnen zu berücksichtigen. Hierzu können natürlich laufend Kopien der entsprechenden Dateien erstellt werden. Ein besserer Ansatz ist jedoch der Einsatz eines **Versionsverwaltungssystems**, welches für

solche Aufgaben entworfen wurde. Dieses macht sich die inkrementelle Natur des Vorgangs zunutze, indem es nur die Änderungen zwischen den Versionen archiviert. Versionen werden über sogenannte **Versionsnummern** identifiziert.

Die Situation wird verschärft, wenn gleichzeitig mehrere Entwickler an einem Projekt arbeiten, das aus vielen, vielleicht Hunderten von Dateien besteht. Einerseits ist dafür das unabhängige Arbeiten zu gewährleisten, andererseits dürfen Änderungen nicht verloren gehen.

## Der Ansatz von CVS

CVS ("Concurrent Versions System") ist ein solches Versionsverwaltungssystem, welches als GNU-Produkt allgemein verfügbar ist und speziell auch mit großen Projekten und vielen Dateien und Entwicklern zurechtkommt.

Für CVS ist ein Projekt einfach ein UNIX-Dateibaum mit den dazugehörigen Dateien. An zentraler Stelle, dem sog. *Repository* `cvsrc`, werden alle Fassungen der einzelnen Dateien zusammen mit den Änderungsprotokollen archiviert. Üblicherweise werden nur *Quellen* CVS unterstellt, nicht jedoch daraus abgeleitete Dateien, die beispielsweise beim Übersetzen von C (`*.o`), Java (`*.class`) oder LaTeX (`*.dvi`) anfallen.

Einzelne Benutzer besitzen jeweils lokale Kopien, die sie wie gewohnt bearbeiten können, insbesondere unabhängig von anderen Entwicklern. Es werden also keine Locking-Mechanismen benutzt.

Nachdem lokale Änderungen ausgetestet sind, werden diese von den Benutzern in das globale Verzeichnis übertragen, wobei CVS Benutzer, Datum etc. geeignet protokolliert. CVS sorgt dafür, daß Änderungen mehrerer Benutzer an der gleichen Datei richtig übernommen werden, sofern sich Änderungen nicht überschneiden. Von CVS kann aber nicht überprüft werden, ob diese Änderungen logisch zusammenpassen, es wird nur syntaktisch auf Kollisionen überprüft. Der einzelne Entwickler übernimmt also bei diesem Vorgang die *Verantwortung* den anderen Entwicklern gegenüber, daß die Änderungen sinnvoll sind und die Dateien sich in einem vernünftigen Zustand befinden (z.B. sich übersetzen lassen).

Man beachte also: "*CVS is not a substitute for developer communication.*" [P. Cederquist]

## Einsatz von CVS

CVS kann zur Verwaltung von beliebigen Dateien benutzt werden, wobei der Schwerpunkt auf Textdateien liegt. Sowohl Texte, die z.B. in HTML oder LaTeX vorliegen, als auch Programmcode in den verschiedensten Sprachen kann somit problemlos verwaltet werden. Die Autoren dieses Artikels benutz(t)en CVS sowohl für umfangreiche Texte, wie Diplomarbeiten, als auch für Webseiten und große Programmpakete. Ein Einsatz von CVS ist sowohl für einzelne Entwickler als auch für große Projekte mit Hunderten von Entwicklern ratsam.

Innerhalb dieser Projekte haben einige Entwickler Schreibrechte, während der interessierte Rest der Welt über einen anonymen Lesezugriff den Entwicklungsstand des Projekts mitverfolgen und auf den entsprechenden Projekt-Mailinglisten mitdiskutieren kann.

Besonders in der Entwicklung von Open Source Projekten ist CVS das Standardwerkzeug. Die meisten GNU Pakete sind mittlerweile per CVS erreichbar, wie die GNU Compiler Collection (GCC) unter <http://gcc.gnu.org>. CVS ist ebenfalls ein Open Source Projekt und wird auch per CVS verwaltet (siehe <http://www.cvshome.org>). Desweiteren werden auf [Sourceforge](http://Sourceforge) mehr als 12000 verschiedene Softwarepakete mit CVS verwaltet. Projekte wie [KDE](#), [Gnome](#) und [Mozilla](#) sind auch unter CVS.

## Weitere Dokumentation

Ausführliche Hilfe zu CVS gibt es in der Info-Dokumentation. Diese ist entweder durch `info cvs` oder im Emacs bzw. XEmacs durch `C-h i` (Einstiegsseite für Info-Dokumentation) erreichbar. Außerdem gibt es noch die Manualpages (`man cvs`).

Eine kurze Hilfe zu CVS gibt es mittels der Aufrufe

```
cvs --help
cvs --help-commands
```

Hilfe zu einzelnen Kommandos gibt es mittels

```
cvs --help kommando
```

Das Buch "Open Source Development with CVS" von Karl Fogel ([Coriolis Verlag](#), ISBN 1-57610-490-7) erläutert nicht nur ausführlich CVS, sondern gibt auch eine Einführung in die Entwicklung freier Software mit zahlreichen Tips für das Projektmanagement.

Die Quellen von CVS und weitere Dokumentation ist über die [Homepage des CVS Projektes](#) verfügbar.

## Die wichtigsten Kommandos

Alle Kommandos von CVS werden als Parameter des UNIX-Programmes `cvs` angegeben. Es kann jeweils nur ein Kommando angegeben werden, wobei die Reihenfolge der Parameter wichtig ist. In fast allen Fällen werden keine zusätzlichen Optionen gebraucht, der Aufruf ist einfach:

```
cvs kommando [<filename>]
```

Wird kein Dateiname übergeben, dann wird das entsprechende Kommando im aktuellen Verzeichnis und in allen Unterverzeichnissen ausgeführt. Der Dateiname darf Pfadinformationen enthalten.

## Vorbereitungen

Um bequem CVS verwenden zu können, richtet man am besten zu Anfang die Arbeitsumgebung entsprechend ein.

## Umgebungsvariablen

Von den vielen von CVS verwendeten Umgebungsvariablen sind im allgemeinen nur drei wichtig: `CVSROOT`, `CVSUMASK` und `CVSEEDITOR`. Der Pfad zum globalen CVS-Verzeichnis befindet sich in `CVSROOT`. Wenn diese Variable nicht gesetzt ist, können einige CVS-Kommandos, insbesondere `import` und `checkout` und `init`, nicht funktionieren! Alternativ kann auch den Kommandos mit Option `-d <verzeichnis>` das Verzeichnis mitgeteilt werden. `CVSUMASK` bestimmt die Maske für die Zugriffsberechtigungen der einzelnen Dateien im globalen Verzeichnis und sollte den Zugriff für die "Gruppe" beinhalten. Mit `CVSEEDITOR` wird schließlich der bei `cvs commit` aufzurufende Editor festgelegt. Diese Variablen können zum Beispiel in der Datei `.bashrc` folgendermaßen definiert werden:

```
export CVSROOT      /cvsroot
export CVSUMASK     003
export CVSEEDITOR   'emacs -nw'
```

## Konfigurationsdateien

In der Datei `.cvsignore` werden Muster für die von CVS zu ignorierende Dateinamen aufgenommen. Für Java bieten sich beispielsweise an, die `.class`-Dateien zu ignorieren. Dies kann entweder über eine globale Datei namens `.cvsignore` im Homeverzeichnis, oder eine Datei `.cvsignore` im entsprechenden CVS-Verzeichnis erfolgen.

Für eine globale Datei, legt man in seinem Home-Verzeichnis die Datei `.cvsignore` mit folgendem Inhalt an:

```
*.class
```

Für bestimmte Kommandos ist es sinnvoll, Standardaufrufparameter in die Datei `~/ .cvsrc` aufzunehmen. Es empfiehlt sich zumindest die Zeile:

```
update -d
```

Somit werden neue Verzeichnisse im Repository beim Update nicht ignoriert, sondern in das aktuelle Verzeichnis übernommen.

## Arbeitskopie erstellen

Um mit CVS an einem Projekt arbeiten zu können, muß zu Anfang eine Arbeitskopie erstellt werden. Dies ist notwendig, damit CVS später die Zusammenhänge mit der globalen Fassung herstellen kann. Mit dem Kommando

```
cvsc checkout <modulname>
```

wird im momentanen Verzeichnis eine aktuelle Kopie des entsprechenden Projektes in einem gleichnamigen Unterverzeichnis angelegt (neudeutsch: *ausgecheckt*).

Dazu muß das Projekt bereits CVS unterstellt worden sein (vgl. [Abschnitt "Einrichten eines Projektes"](#)).

## Änderungen synchronisieren

Arbeiten mehrere Entwickler zusammen, sind die Änderungen regelmäßig abzugleichen. Um dies bei eigenen Dateien mit den Änderungen anderer zu tun, verwendet man das Kommando

```
cvsc update
```

Damit werden die global erfolgten Änderungen in die eigenen lokalen Kopien übernommen. Hatte man die lokale Kopie nicht geändert, kann dies problemlos geschehen. Andernfalls werden im Falle nicht-überlappender Änderungen die globalen Änderungen integriert, ohne die lokalen zu überschreiben. Kommt es zu Konflikten, weil die gleichen Stellen verändert wurden, werden in der Datei beide Fassungen in folgender Form aufgenommen:

```
...
<<<<<<< filename
... lokaler Text ...
=====
... globaler Text ...
>>>>>>> 1.3
...
```

Das Programm gibt hierbei eine entsprechende Warnung aus:

```
cvs update: Updating .  
RCS file: /home/aj/cvsroot/Test/filename,v  
retrieving revision 1.2  
retrieving revision 1.3
```

```
Merging differences between 1.2 and 1.3 into filename  
rcsmerge: warning: conflicts during merge  
cvs update: conflicts found in filename
```

Dann sind die entsprechenden Stellen von Hand in Ordnung zu bringen.

Um eigene Änderungen global verfügbar zu machen (neudeutsch: *einzuchecken*), verwendet man das Kommando `commit`. Hierbei ist zu unterscheiden, ob dies bei einer einzelnen Datei, einer Dateigruppe oder allen geänderten Dateien erfolgen soll, da alle Änderungen mit der gleichen Änderungsnachricht versehen werden und diese natürlich inhaltlich sinnvoll sein sollen. Um eine einzelne Datei einzuchecken verwendet man folgenden Aufruf:

```
cvs commit <filename>
```

Bei mehreren Dateien sind einfach die verschiedenen Namen aufzuführen. Will man hingegen *alle* Änderungen vornehmen, so verwendet man

```
cvs commit
```

Ein Einchecken ist nicht möglich, wenn die globalen Änderungen nicht lokal übernommen und Konflikte nicht entsprechend aufgelöst wurden. Deshalb empfiehlt sich direkt vor einem `cvs commit` ein `cvs update`. CVS gibt aber auch entsprechende Meldungen aus.

Um die Unterschiede zwischen lokaler und globaler Fassung zu bestimmen, kann man den Aufruf

```
cvs diff <filename>
```

verwenden. Mit der zusätzlichen Angabe von Versionsnummern lassen sich die Differenzen zwischen beliebigen Fassungen ausgeben:

```
cvs diff -r1.1 -r1.5 <filename>
```

Das Kommando `cvs commit` ruft den in `CVSEEDITOR` spezifizierten Editor auf, mit dem man eine Änderungsnachricht erstellt. Dies ist ein wichtiges Feature für die Verfolgbarkeit von Änderungen: Warum wurde damals dies und das von wem gemacht? Alternativ kann man mit dem Argument `-m` "Nachricht" eine Kurznachricht direkt über die Kommandozeile übergeben. Die Liste aller dieser Nachrichten kann durch

```
cvs log <filename>
```

ausgegeben werden.

## Neue Dateien registrieren

Soll in ein bestehendes Projekt eine neue Datei integriert werden, so legt man sie zunächst lokal im vorgesehenen (und bereits ausgecheckten!) Verzeichnis an. Dann merkt man sie mittels

```
cvs add <filename>
```

vor. Um sie dann global zu registrieren, erfolgt ein:

```
 cvs commit <filename>
```

Neue Unterverzeichnisse werden auf die gleiche Weise angelegt.

## Dateien löschen

Muß man wirklich Dateien löschen, weil z.B. eine Datei aufgeteilt wurde, so entfernt man sie zunächst lokal mittels `rm`. Danach ruft man

```
 cvs delete <filename>
```

zum Vormerken dieser Operation auf. Schließlich erfolgt noch ein

```
 cvs commit <filename>
```

womit die Aktion auch global ausgeführt wird, die Datei also in diesem Projekt auch unter `CVSROOT` gelöscht wird. CVS hält aber dennoch zur Sicherheit die alte Fassung in einem speziellen Verzeichnis unter `CVSROOT` vor.

## Versionsinformationen im Quelltext

CVS gestattet es, Versionsinformationen in der Datei explizit zu machen. Dies geschieht durch besondere Schlüsselwörter im Text, die dann durch das System beim Aus- oder Einchecken geeignet ersetzt werden. Die wichtigsten sind `$Author$` für den Autor, `$Date$` für das Datum der letzten Version, `$Revision$` für die aktuelle Versionsnummer und `$Log$` für das komplette Änderungsprotokoll. Eine Zusammenfassung der wichtigsten Informationen bietet `$Id$`

Für Programme empfiehlt es sich, die gewünschten Informationen in Kommentaren oder - wenn sie auch im übersetzten Programm noch vorhanden sein sollen - Strings anzulegen. Unter LaTeX bietet der `rsc.sty` (zu finden unter <ftp://ftp.dante.de/tex-archive/macros/latex/contrib/supported/rsc/>) einen komfortablen Zugriff auf diese Informationen.

## Handhabung aus dem Emacs/XEmacs heraus

Emacs und XEmacs unterstützen seit den neueren Versionen über VC (version control) direkt auch CVS, allerdings vorzugsweise auf Dateiebene. Das wichtigste Kommando, das eigentlich immer das "Richtige" macht, ist `C-x v v`. Ansonsten kann man VC auch über das Menü `Tools` ansprechen.

## Ergänzungen

Die in diesem Abschnitt erwähnten Kommandos sind für den täglichen Gebrauch von CVS nicht so wichtig.

## Einrichten eines globalen CVS-Verzeichnisses

Das globale CVS-Verzeichnis für das Repository wird nach Setzen der Umgebungsvariable `CVSROOT` durch

```
cv$ init
```

angelegt und initialisiert. Dies muß vor allen anderen Aktionen erfolgen.

## Einrichten eines Projektes

Unter dem globalen Verzeichnis können mehrere Projekte untergebracht werden. In diesem Zusammenhang ist für CVS ein Projekt ein Verzeichnisbaum. Um einen bestehenden Verzeichnisbaum als Projekt unter die Kontrolle von CVS zu stellen, wechselt man in die oberste Ebene des bestehenden Verzeichnisbaumes und führt

```
cv$ import <modulname> <vendor-tag> <release-tag>
```

aus. Hierbei gibt *<modulname>* den Namen des Wurzelverzeichnisses des neuen Projektes an, darunter wird der komplette Baum im globalen Verzeichnis kopiert und bleibt lokal unverändert. Soll ein neues Projekt gestartet werden, zu dem noch keine Dateien existieren, legt man zunächst ein neues, leeres Verzeichnis an, wechselt hinein und führt dann `cv$ import` aus. *<vendor-tag>* und *<release-tag>* sind Namen, die diesen speziellen Import kennzeichnen und können normalerweise auf Dummy-Werte gesetzt werden.

**Achtung:** Um hinterher das Projekt unter CVS zu bearbeiten, ist es unbedingt notwendig, es mit:

```
cv$ checkout <modulname>
```

neu an geeigneter Stelle auszuchecken.

## Lokale und Remote CVS Verzeichnisse

Für das Aufsetzen von CVS gibt es verschiedene Möglichkeiten. Im allgemeinen hat die `CVSROOT`-Variable folgendes Format: `:METHODE:BENUTZER@RECHNER:/cvs/wurzel/verzeichnis`, wobei entsprechende Defaults benutzt werden, falls `METHODE`, `BENUTZER@` oder `BENUTZER@RECHNER` ausgelassen werden.

Die erste Methode (als "local" bezeichnet) ist ein Zugriff auf ein lokales Verzeichnis, auf das alle Entwickler zugreifen können (evtl. per NFS im lokalen Netz). Die `CVSROOT`-Variable wird so beispielsweise auf den Wert `:local:/cvsroot` oder alternativ (das `:local:` ist optional) auf `/cvsroot` gesetzt.

Für Zugriff auf CVS Verzeichnisse auf anderen Rechnern kann mittels eines speziellen CVS-Servers oder per `rsh` bzw. `ssh` zugegriffen werden. Bei wenigen Entwicklern oder bei einem anonymen Zugriff wird der CVS-Server bevorzugt. Als Methode wird `pserver` benutzt, so daß `CVSROOT` z.B. auf `:pserver:anoncvs@anoncvs.cygnus.com:/cvs/gcc` gesetzt wird.

Beim Zugriff per `rsh` wird als Methode `ext` benutzt. Wird anstelle von `rsh` ein verschlüsselter Zugang mit `ssh` gewünscht (dies ist die normale Konfiguration für Entwickler von Open-Source Projekten mit Schreibzugriff), so muß die Umgebungsvariable `CVS_RSH` auf `ssh` gesetzt werden. `CVSROOT` hat dann beispielsweise den Wert `:ext:cvs.x86-64.org:/cvsroot`.

## Erstellen von Releases

Ein Release ist ein (konsistenter) Schnappschuß eines kompletten Projektes, der später jederzeit rekonstruierbar sein soll. Es bietet sich an, zumindest nach den einzelnen abgeschlossenen Entwicklungsphasen eines Projektes jeweils einen Release zu erstellen. Dies ist deshalb nicht trivial, da

zu dem Zeitpunkt beispielsweise Datei-1 die aktuelle Versionsnummer 1.17 besitzt, während Datei-2 vielleicht 1.78 hat. Hinzu kommt, daß zu einem späteren Zeitpunkt einzelne Dateien hinzugefügt oder gelöscht werden können. Releases werden nicht durch ein Nummernschema, sondern durch symbolische Namen bezeichnet, wie alpha beta, Spezielle-Fassung-fuer-Bernd etc.

Ein Release wird erstellt durch:


```
cv s tag <release>
```

Um einen gegebenen Release eines Projektes auszuchecken, führt man ein

```
cv s checkout -r <release> <modulname>
```

aus.

*Bernd Löchner promoviert an der Universität Kaiserslautern. Seine Spezialgebiete sind automatisches Beweisen und Programmtransformation. Er ärgert sich jedesmal, wenn er mal wieder vergessen hat, ein noch so kleines Projekt **rechtzeitig** unter CVS zu stellen.*

*Andreas Jaeger hat mit Bernd an einigen Projekten zusammen gearbeitet und arbeitet mittlerweile in den SuSE Labs. Dort benutzt er CVS täglich bei Projekten wie der GNU C Library und GCC.* 

**Linux auf dem Server 22.12.2000**