

# Teamwork im Pool

Winzige Einführung in CVS

Nils Knappmeier

8.2.2002

## Inhaltsverzeichnis

<b>1</b>	<b>Was ist CVS?</b>	<b>1</b>
<b>2</b>	<b>Einrichten des CVS im RBG-Pool</b>	<b>1</b>
2.1	Das Repository . . . . .	1
2.1.1	Initialisierung . . . . .	2
2.1.2	Lese und Schreibberechtigung . . . . .	2
2.1.3	Umgebungsvariablen . . . . .	2
2.2	Arbeiten mit CVS . . . . .	2
2.2.1	Projekt auschecken (Lokale Kopie erstellen) . . . . .	2
2.2.2	Update auf den aktuellen Stand . . . . .	2
2.2.3	Dateien hinzufügen . . . . .	3
2.2.4	Dateien löschen . . . . .	3
2.2.5	Änderungen übertragen . . . . .	3
<b>3</b>	<b>Weitere Informationen</b>	<b>3</b>

## 1 Was ist CVS?

CVS ist ein System zum gemeinsamen Arbeiten, an einem Projekt. Die Idee dabei ist, dass es einen gemeinsamen Datenpool gibt (Repository), in dem das Projekt (zum Beispiel die Java-Dateien) lagert. Jedes Team-Mitglied macht sich eine eigene Kopie dieses Datenpools (checkout) und kann damit arbeiten und herumprobieren, ohne die anderen zu stören. Wenn die Änderungen fertig sind und funktionieren, können sie in den gemeinsamen Datenpool zurückübertragen werden (commit).

Das Schöne dabei ist, dass im Repository immer nur die Änderungen gespeichert werden. So ist es möglich, alte Versionen von Dateien wiederherzustellen, wenn mal was schief laufen sollte.

## 2 Einrichten des CVS im RBG-Pool

### 2.1 Das Repository

Zunächst einmal brauchen wir eine Stelle, an der wir den gemeinsamen Datenpool speichern. Dies sollte **nicht** im Verzeichnis `/media/tmp` sein, da diese Dateien regelmäßig gelöscht werden. Einer von euch muss sich also dazu bereiterklären, das Repository in seinem Home-Verzeichnis unterzubringen. Nehmen wir ab jetzt einfach mal an, dieser User hätte den Login `matze`. Dann legt ihr am besten ein Unterverzeichnis `/home/matze/.cvs` an. Dieses Verzeichnis ist unsichtbar und stört dann den täglichen Betrieb nicht weiter.

### 2.1.1 Initialisierung

Damit CVS mit dem Verzeichnis auch was anfangen kann, müssen ein paar Kontrolldateien hineingeschrieben werden. Das erreicht ihr mit dem Befehl:

```
cvs -d /home/matze/.cvs init
```

Ausserdem ist es so, dass ein Repository mehrere Projekte beinhalten kann. Diese Projekte werden Module genannt. Jedes Modul stellt ein Unterverzeichnis von `/home/matze/cvs` dar. Ihr müsst also für euer Projekt ein Unterverzeichnis anlegen (nennen wir es `robo`).

### 2.1.2 Lese und Schreibberechtigung

Natürlich muss jedes Team-Mitglied auf dem Repository lesen und schreiben können. Andererseits soll dies nicht unbedingt jeder dahergelaufene Hannebambel können. Die RBG-Pools bieten mit den Befehlen `setfacl` und `getfacl` die Möglichkeit, Dateien für ganz bestimmte Personen zum Lesen oder Schreiben freizugeben.

Kurz: Um `/home/matze,/home/matze/.cvs` und `/home/matze/.cvs/robo` für einen anderen User (z.B. *minki*) freizugeben müsst ihr folgende Befehle ausführen:

```
setfacl -r -m user:minki:rwX /home/matze
setfacl -r -m user:minki:rwX /home/matze/.cvs
setfacl -r -m user:minki:rwX /home/matze/.cvs/robo
```

Das müsst ihr für alle Teammitglieder machen.

### 2.1.3 Umgebungsvariablen

Damit CVS beim auschecken weiss, wo das Repository liegt, setzt man am besten die Umgebungsvariable `CVSROOT` auf `/home/matze/.cvs`

```
export CVSROOT=/home/matze/.cvs
```

## 2.2 Arbeiten mit CVS

### 2.2.1 Projekt auschecken (Lokale Kopie erstellen)

Mit dem Befehl `cvs co robo` könnt ihr eure lokale Kopie des Projektes erstellen. Wenn ihr den Befehl ausgeführt habt, findet ihr im aktuellen Verzeichnis ein Unterverzeichnis `robo`, in dem die Kopie des Projektes zu finden ist. Mit dieser Kopie könnt ihr nun arbeiten.

### 2.2.2 Update auf den aktuellen Stand

Wenn ihr innerhalb eurer Arbeitskopie den Befehl `cvs update -d` ausführt, werden alle Dateien in diesem Verzeichnis und allen Unterverzeichnissen auf den aktuellen Stand gebracht.

Das heisst:

- alte Dateien werden auf den neuesten Stand gebracht.

- Dateien, die von anderen Leuten neu hinzugefügt wurden, werden in die Kopie aufgenommen.
- Wenn Dateien von anderen Leuten und von euch verändert wurden, versucht CVS alle Veränderungen sinnvoll zusammenzumischen. Wenn das nicht geht, markiert es die Stellen in der Datei, an denen es Konflikte gab.

Der Parameter **-d** bewirkt, dass auch neue Verzeichnisse erzeugt werden.

### 2.2.3 Dateien hinzufügen

Mit dem Befehl `cv`s `add` `<datei1>` `<datei2>` ... könnt ihr neue Dateien zum Repository hinzufügen. Die Dateien werden zunächst nur markiert und erst beim `commit` übertragen.

### 2.2.4 Dateien löschen

Mit dem Befehl `cv`s `remove` `<datei1>` `<datei2>` ... könnt ihr Dateien aus dem Repository löschen. Vorher müsst ihr die Dateien allerdings aus eurer lokalen Kopie gelöscht haben. Auch hier werden die Änderungen erst beim `commit` ausgeführt.

*Hinweis: Wenn ihr Dateien in eurer lokalen Kopie löscht, und nicht mit `cv`s `remove` aus dem Repository entfernt, erscheinen sie beim nächsten `update` wieder bei euch.*

### 2.2.5 Änderungen übertragen

Mit dem `cv`s `commit` übertragt ihr alle Änderungen in dem aktuellen Verzeichnisbaum an das Repository. Wenn ihr nur Änderungen in bestimmten Dateien übertragen wollt, könnt ihr dies mit `cv`s `commit` `<datei1>` `<datei2>` ... tun.

## 3 Weitere Informationen

Weitere Informationen zu CVS findet ihr

1. auf der CVS-Manpage (`man cv`s)
2. in der CVS-Dokumentation unter `/usr/local/doc/cvs/cvs.ps`
3. in der KIF-Doku von Darmstadt (<http://www.fachschaft.informatik.tu-darmstadt.de/kif/doku/> unter dem Arbeitskreis *Nachhaltig Arbeiten und Organisieren*)
4. auf der CVS-Homepage (<http://www.cvshome.org>)

Ausserdem gibt es ein paar nette Frontends:

1. Frontend für Windows <http://www.wincvs.org/>
2. `tkcvs` hier im Pool
3. einfach mal bei <http://freshmeat.net> nach CVS suchen.