

# Hausarbeit in Open Source Software

## CVS – Concurrent Version System

Christoph Korn

Gliederung:

1. Allgemeine Informationen.....	2
2. Beispiel.....	4
2.1 Installieren von CVS unter Suse Linux.....	4
2.2 Arbeitsschritte.....	5
3. Quellen.....	8

# 1. Allgemeine Informationen

CVS steht für Concurrent Version System. Es ist ein Versions-Kontroll-/Überwachungs-Programm, mit dem Dateien von mehreren Personen gleichzeitig bearbeitet werden können. Der Status jeder Datei wird mitprotokolliert. Somit ist ein späteres Prüfen der Veränderungen bzw. Zurückkehren zu einer älteren Versionen von Dateien oder des ganzen Projektes möglich. Die Verwaltung mehrerer Projekte wird in einem Repository (=Verwaltungsarchiv) durchgeführt.

CVS ist keine Software, die auf Lauffähigkeit bzw. Kompilierbarkeit prüft. Die Programmlogik wird somit ebenso nicht kontrolliert. CVS ersetzt auch nicht die Kommunikation zwischen den Entwicklern. CVS darf nicht mit einer Entwicklungsumgebung verwechselt werden; es trifft keinerlei Annahmen über den Inhalt der Dateien. Es kann nur textuelle und keine logischen Konflikte auflösen.

Das Repository ist das CVS-Archiv, welches die einzelnen Projekte enthält. Es wird zwischen einem Lokal- und Remote-Repository unterschieden. Lokal bedeutet dabei, dass es sich auf dem Rechner befindet, von dem man auf ein Projekt zugreift. Bei Remote-Repository wird das Projekt über das Netz geladen. Hierbei bietet CVS eine Vielzahl von Möglichkeiten an. So kann die Remoteverbindung unverschlüsselt, per rsh oder auch mit ssh realisiert werden.

Im Folgenden wird auf die Umgebungsvariablen eingegangen, die bei Remote-Repository gesetzt werden sollen. Das sonstige Variablensetzen ist optional. Es wird jedoch je nach Betriebssystem folgendes empfohlen:

Plattform	Umgebungsvariablen
Unix	CVSROOT, ggf. PATH
Linux	CVSROOT, ggf. PATH

Das Setzen der CVSROOT Variablen ist deshalb dringend zu empfehlen, da sonst bei vielen Aufrufen mit der Option `-d <CVSROOT>` die entsprechende Angabe gemacht werden muss.

Es empfiehlt sich natürlich den Aufrufort des Programmes CVS in die PATH Variable zu übernehmen, um ein einfaches Aufrufen von der Kommandozeile zu ermöglichen.

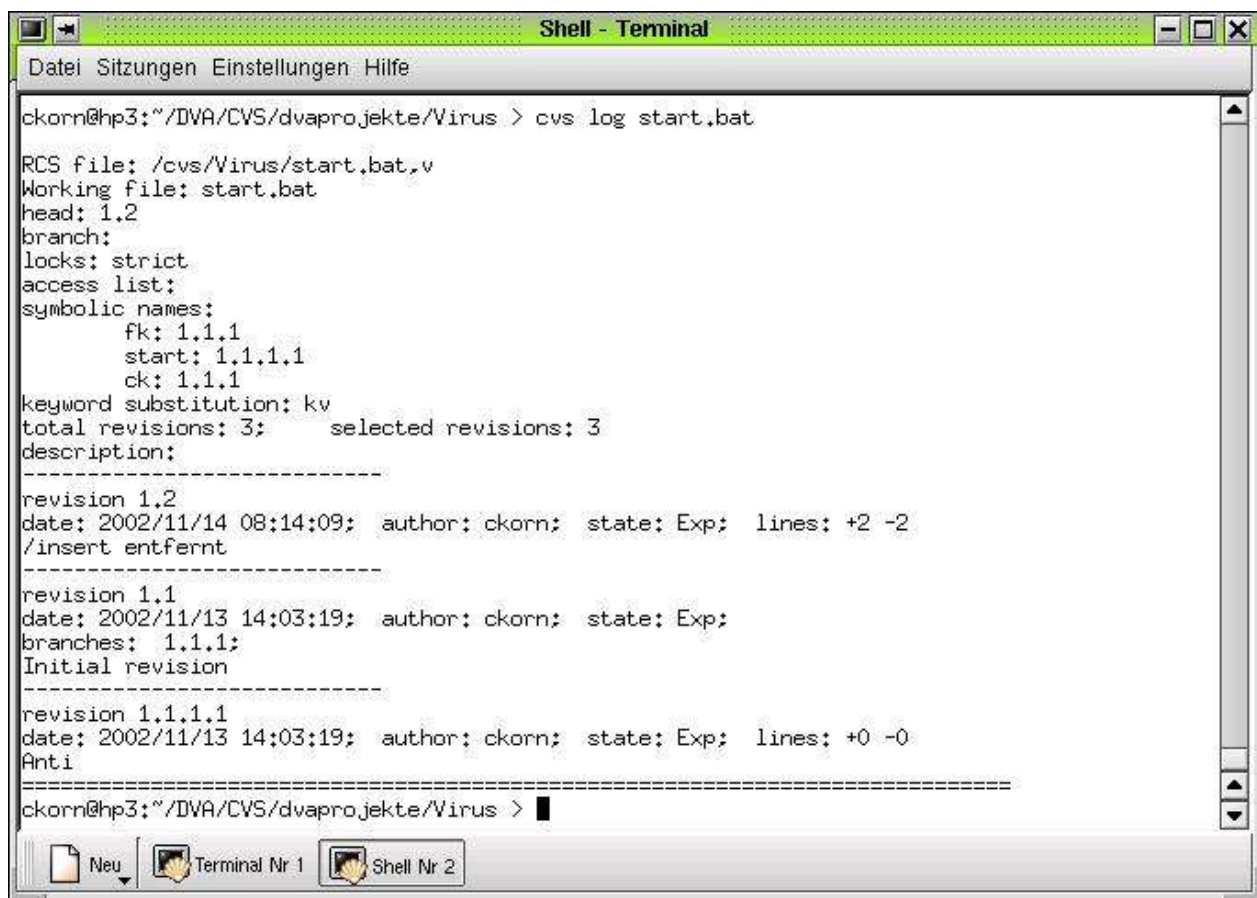
Nachfolgend werden noch einige Begriffe erklärt:

### Revision:

Revisions sind Veränderungen an Dateien des Projektes, die mit COMMIT abgeschlossen sind. Sie entsprechen einer Momentaufnahme des Projektes. Sie sind nicht zu verwechseln mit den einzelnen Versionen eines Programmes.

### Log-Nachrichten:

Log-Nachrichten dienen der genauen Beschreibung der Veränderungen an einzelnen Dateien. Mithilfe der Log-Nachrichten kann ein Entwickler den Veränderungsprozess nachvollziehen.



```
Shell - Terminal
Datei Sitzungen Einstellungen Hilfe
ckorn@hp3:~/DVA/CVS/dvaprojekte/Virus > cvs log start.bat
RCS file: /cvs/Virus/start.bat,v
Working file: start.bat
head: 1.2
branch:
locks: strict
access list:
symbolic names:
    fk: 1.1.1
    start: 1.1.1.1
    ck: 1.1.1
keyword substitution: kv
total revisions: 3;   selected revisions: 3
description:
-----
revision 1.2
date: 2002/11/14 08:14:09;  author: ckorn;  state: Exp;  lines: +2 -2
/insert entfernt
-----
revision 1.1
date: 2002/11/13 14:03:19;  author: ckorn;  state: Exp;
branches: 1.1.1;
Initial revision
-----
revision 1.1.1.1
date: 2002/11/13 14:03:19;  author: ckorn;  state: Exp;  lines: +0 -0
Anti
-----
ckorn@hp3:~/DVA/CVS/dvaprojekte/Virus > █
```

### Tagging:

Tagging (engl. Markieren) unterscheidet sich von einem normalen COMMIT dadurch, dass hier keine Veränderungen am Quell-Text gespeichert werden, sondern lediglich eine Veränderung der Einschätzung der Dateien durch den Entwickler vorgenommen wird.

So kann dann ein bestimmter Zustand, in dem sich ein Projekt befand, zu einem späteren Zeitpunkt wieder hergestellt werden, denn eine Markierung zeichnet eine Gruppe von Revisionen (Arbeitskopien des Entwicklers) aus.

## Branch:

Branches werden zum Aufspalten des Versionspfades eingesetzt. Somit können parallel zwei gleiche Versionen des gleichen Projekts vorhanden sein. Dies kann sinnvoll sein, wenn beispielsweise dem Kunden eine Version zugesandt werden soll. Diese sollte normalerweise nicht mehr verändert werden, damit eine gleiche Basis bei Kundengesprächen gegeben ist. Der Versionsbaum wird an dieser Stelle also „eingefroren“ und es werden nur Änderungen seitens des Supports vorgenommen. In einem solchen Fall kann der Branch-Mechanismus sinnvoll sein. Branches werden zudem eingesetzt, wenn die Entwickler in einer Applikation mehrere unterschiedliche Ideen für die Fortsetzung des Projekts ausprobieren möchten. Auch hier kann Branch sinnvoll sein, da damit dann die schlechtere Implementierung von der anderen komplett getrennt ist und somit leicht gelöscht werden kann.

## 2. Beispiel

### ***2.1 Installieren von CVS unter Suse Linux***

Zur Installation unter Suse Linux verwenden wir die Quellen von der CVS-Homepage (<http://www.cvshome.org>) in der Version 1.11.2. Über das Kommando `~$ ./configure` wird automatisch das Makefile erstellt, das dann mit dem Kommando `~$ make` die Dateien kompiliert. Für den Installationsvorgang (`~$ make install`) werden root-Rechte benötigt, da die ausführbare CVS Datei in `/usr/local/bin` kopiert werden soll.

Als CVSROOT legen wir ein neues Verzeichnis `/cvs` im Rootdirectory an. In diesem bekommen alle User Lese/Schreibrechte (ggf. könnte man an dieser Stelle eine eigene Usergroup `cvs` einrichten und nur dieser die Rechte vergeben).

Als nächstes konfigurieren wir den Remotezugang auf den Rechner. Zuerst muss in der Datei `/etc/services` der Eintrag

```
cvspserver 2401/tcp
```

gemacht werden. Dadurch setzt das Linuxsystem die Portnummer in einen Dienstnamen (hier `cvspserver`) um.

In die Datei `/etc/inetd.conf` wird

```
cvspserver stream tcp nowait root /usr/local/bin/cvs \  
--allow-root=/cvs pserver
```

eingetragen. Durch Neustarten des Daemons wird die Änderung aktiv. Somit werden nun alle Anfragen, die über den Port 2401 kommen, direkt an das lokale Programm `cvs` weitergeleitet, dessen CVSROOT als `/cvs` angegeben ist. Die Passwortverwaltung übernimmt in unserem Fall das Linuxsystem. Man kann jedoch explizit Nutzer in der

.cvspass Datei anlegen, die im CVSROOT liegt. Dies muss nach folgenden Muster

```
<BENUTZERNAME>:<VERSCHLÜSSELTES PASSWORT>:<OPTIONALER SYSTEM BENUTZERNAME>
```

erfolgen. So können auch reine CVS Benutzer hinzugefügt werden. Das hierfür benötigte verschlüsselte Passwort erhält man bei existierenden Usern entweder als Root aus der Datei /etc/passwd oder bei CVS Benutzern über ein Perlskript das unter [1], Kapitel 4 zu finden ist.

Als letzten Installationschritt wird noch das CVS Archiv erstellt. Dies erfolgt über den Befehl

```
~$ cvs -d /cvs init.
```

Damit entsteht im Verzeichnis /cvs ein Unterverzeichnis /CVSROOT, das die von CVS benötigten Dateien enthält.

## 2.2 Arbeitsschritte

Bevor wir mit dem eigentlichen Arbeiten beginnen können, müssen wir uns erst in CVS einloggen. Hierzu wird die Zugriffsmethode (in unserem Fall pserver) sowie der Benutzer (hier: ckorn) und der Rechnername (HP4) angegeben. "/cvs" ist das Verzeichnis, das das Archiv (Repository) enthält. Alle Informationen werden mit einem Doppelpunkt getrennt. Die allgemeine Syntax lautet:

```
cvs -d <Loginart>:<Username>@<Rechnername>:<CVSRoot> login
```

Anschließend wird nur noch die Umgebungsvariable CVSROOT für den Benutzer gesetzt, damit man bei allen weiteren Befehlen auf deren erneute Angabe verzichten kann.



```
Shell - Terminal
Datei Sitzungen Einstellungen Hilfe
ckorn@hp3:~$ > cvs -d ;pserver;ckorn@HP4:/cvs log
Logging in to ;pserver;ckorn@hp4:2401/cvs
CVS password:
ckorn@hp3:~$ >
ckorn@hp3:~$ > CVSROOT=;pserver;ckorn@HP4:/cvs
ckorn@hp3:~$ > export CVSROOT
ckorn@hp3:~$ >
```

Jetzt können Projekte über den Importbefehl in das Archiv aufgenommen werden:

```
~$ cvs import -m "log nachr." projname hersteller-marke versions-marke
```

Als Beispiel:

```
~$ cvs import -m "Test eingecheck" Test ck start
```

Damit wird ein Projekt Test angelegt, mit der Lognachricht „Test eingecheckt“, dem Hersteller ck und der Versionsmarke 1.1.

Zuerst sollten die Benutzer A und B das gleiche Projekt auschecken. Hierbei gab es - wie zu erwarten - keine Probleme.

Dies geschieht über den Aufruf:

```
~$ cvs checkout Test
```

Anschließend verändert A die Datei x und führt anschließend den COMMIT aus.

Der Befehl hierfür ist einfach:

```
~$ cvs commit
```

bzw.

```
~$ cvs commit -m "Lognachricht"
```

wodurch die Lognachricht direkt eingegeben werden kann, ohne einen Editor zu öffnen. Diesen Vorgang wiederholt A noch einmal. Nun ist die aktuelle Version der Datei x im Archiv und die Version wurde um 2 Nummern erhöht. Die Folge ist, dass der Benutzer B keine aktuelle Arbeitskopie mehr besitzt.

Nun kann Benutzer C den CHECKOUT des Projektes durchführen. Dieser erfolgt ohne Probleme; C erhält die aktuelle Version (1.3) der Datei x.

Jetzt verändert A die Datei x und führt erneut COMMIT aus. Dieser COMMIT läuft ebenfalls ohne Problem ab. Die Benutzer B und C haben jedoch beide keine aktuellen Arbeitskopien mehr.

Benutzer B verändert nun die Datei x und führt COMMIT aus. Dies führt zu einer Fehlermeldung, da ein Konflikt auftritt: Er hat die Datei an derselben Stelle geändert wie Benutzer A. Benutzer B führt daraufhin einen UPDATE aus und öffnet anschließend die entsprechende Datei.

In dieser wird der aktuelle Konflikt beschrieben. Nun kann B den Konflikt - möglicherweise mit Hilfe eines anderen Entwicklers (A) - lösen. Danach läuft der COMMIT erfolgreich ab.

Als weitere Hilfestellung für den Entwickler gibt es das diff-Kommando. Es vergleicht die möglicherweise modifizierten Dateien der Arbeitskopie mit den entsprechenden Gegenständen im Archiv und zeigt jegliche Unterschiede auf:

```
$ cvs diff
```

Etwas ausführlicher und besser leserlich ist die Angabe der Optionen `-Q`, sodass nicht angezeigt wird in welchem Verzeichnis CVS gerade arbeitet.

Die meisten empfinden das »Kontext«-Diff-Format als leichter zu lesen, da es ein paar Kontextzeilen zu beiden Seiten einer Veränderung mit anzeigt. Kontext Diffs werden durch die zusätzliche Option `-c` zu `diff` erzeugt:

```
~$ cvs -Q diff -c
```

Die Ausgabe kann wie folgt gelesen werden:

„+“ steht vor Zeilen, die ergänzt wurden, „-“ vor entfernten Zeilen. Ein „!“ gibt an, dass in dieser Zeile eine Änderung vorgenommen wurde.

### 3. Quellen

[1]: <http://cvsbook.red-bean.com/translations/german>

[2]: <http://www.cvshome.org>

[3]: <http://www.wincvs.org>