CVS

"Bjoern A. Zeeb" <zebj@ba-horb.de>

8. Januar 2002

Inhaltsverzeichnis

1 Einführung

1.1 CVS - Was ist das ?

CVS steht für "Concurrent Versions System". Es ermöglicht die parallele Arbeit mehrerer Programmierer an dem Quellcode eines Projektes. CVS ist ein Open Source-Tool und steht daher jedem zur Verfügung [1].

1.2 Warum CVS ?

Bei CVS steht besonders auch die Versionsverwaltung einzelner Dateien und von Projekten im Vordergrund. Hierzu zwei Beispiele:

- Ein Worddokument "Version3" wird morgens per Email auf die Reise geschickt an zwei Personen. Beide verändern etwas und schicken das Dokument an jemand anderen weiter. Am Abend gibt es statt einer Gesamtdatei 15 verschiedene "Version4"-Dokumente.
- Man arbeitet an einem grösseren Projekt und legt sich dafür ein Verzeichnis `Projekt` an. Nach einer Überarbeitung wird ein weiteres Verzeichnis `Projekt_1` angelegt, dann später `Projekt_1_1`, `Projekt_2`, usw. für jede Version. Am Schluss hat man viele Verzeichnisse, immer mit den gleichen Dateien nur jeweils eine (Teil-)Version weiter. Keiner dieser Namen sagt etwas über den wirklichen Inhalt bzw. Entwicklungsstand oder die Veränderungen des Projektes aus.

Beide obige Beispiele arten bei vielen Personen oder vielen Datein und langen Projekten schnell aus und werden unpraktikabel. CVS versucht die Sache zu entschärfen indem zum einen alle Dateien an einer zentralen Stelle gehalten werden und zum anderen die Versionierung intern vom Programm übernommen wird. Mehrere Benutzer können sich dann eine lokale Kopie aus dem Verzeichnisbaum auschecken, unabhängig voneinander bearbeiten und wieder in das Repository zurückschieben. CVS nutzt einen 'merge'-Ansatz, d.h. wenn Konflikte auftreten sind diese von bzw. unter den Entwicklern zu lösen und solange wird eine Datei nicht in das Repository reingelassen. Andere Tools (später) lösen diese Problematik z.B. dadurch, dass sie nur immer einem Benutzer gleichzeitig das Recht geben an einer Datei zu arbeiten.

1.3 Was kann/ist CVS nicht

- CVS ist in keinster Weise ein "build system", d.h. es ist nur für die Verwaltung der Datein zuständig aber nicht für das, was man mit den Dateien machen will - z.B. Programm aus Quelltexten erstellen, ...
- CVS ist kein Ersatz für ein Projektmanagement oder die Kommunikation zwischen den Entwicklern. Ganz im Gegenteil. Zum einen muss wie im Projektmanagement jemand da sein, der sich um das Repository kümmert und die ganze

Entwicklung koordiniert. Zum anderen ist bei Versionskonflikten o.ä. die Kommunikation zwischen den Entwicklern der einzige Weg, um sinnvoll eine Lösung zu erreichen und nicht kontraproduktiv zu arbeiten.

1.4 Alternativen

Einige mögliche Alternativen seien hier nur der Vollständigkeit halber kurz bzw. namentlich erwähnt.

1.4.1 RCS - Revision Control System

RCS arbeitet nach dem 'locking'-Prinzip, d.h. es kann immer nur eine Person an einer Datei arbeiten. CVS hat RCS unter anderem um die Möglichkeit verteilt über Netzwerke zu arbeiten erweitert. Mehr Informationen zu RCS findet man in den entsprechenden manpages[2].

1.4.2 SCCS - Source Code Control System

1.4.3 PVCS

2 Funktionsweise

2.1 Servertypen

Ein Teil der verschiedenen Servertypen seien an dieser Stelle nur kurz erwähnt. Wir werden wohl hauptsächlich mit Punkt 2.1.4 'local' arbeiten. Das Einrichten eines Servers wird hier nicht beschrieben.

2.1.1 pserver

Dies ist die Standardbetriebsweise eines CVS Servers. Server meint in diesem Zusammenhang den Rechner, der den Dienst zur Verfügung stellt, auf dem das Repository liegt. Jeder Benutzer muss sich hier mit seinem Usernamen und Passwort am Server anmelden bevor er Zugang zum Repository bekommt. Die Zugänge können virtuell sein, d.h. die Benutzer brauchen keinen Login zum Rechner selber (shell Zugriff).

2.1.2 anonymous

Ein *anonymous* oder *public* CVS Server ist ein Sonderfall der pserver-Variante. Hier bekommt ein spezieller Login mit einem öffentlich bekannten Passwort oder keinem Passwort Zugang zum Repository. Meist ist dieser Zugang nur lesend.

2.1.3 external (rsh / ssh)

Auf *external* Repositories wird per rsh (remote shell, default) oder ssh (secure shell) zugegriffen. Welchen Dienst man für den remote-Zugriff nutzt wird über die Umgebungsvariable `$CVS_RSH` geregelt. Man meldet sich hierbei (im Hintergrund automatisch) auf dem Rechner mit dem CVS Repository an. Auf dem CVS Server wird ein echter Login dafür benötigt.

2.1.4 local

Ist das CVS repository *lokal* auf dem Rechner bzw. per NFS o.ä. auf den Rechner gemountet, so braucht man die Netzwerkfähigkeit von cvs nicht. Hier entfällt das anmelden z.B. komplett.

2.2 Repository

2.2.1 Was ist ein Repository ?

Bei CVS spricht man von sogenannten *repositories*. Ein Repository ist ganz grob gesprochen eine versionierte, strukturierte Ansammlung von Verzeichnissen und Dateien. Ein CVS Server kann mehrere verschiedene Repositories führen.

Leider wird das Schlagwort 'Repository' je nach Literaturstelle und Sprachgebrauch für zwei verschiedene Dinge bei CVS genutzt. Zum einen wird die Wurzel des CVS-Verzeichnisbaumes (`$CVSROOT` - mehr dazu gleich) so bezeichnet. Zum anderen aber manchmal auch die einzelnen darunterliegenden Verzeichnisse (Module). Dies führt gelegentlich an manchen Stellen zu Verwirrungen.

2.2.2 Repository anlegen

Um eine Repository anlegen zu können muss man sich zuerst überlegen wo in der Verzeichnisstruktur auf der Platte man die Datein ablegen will. Diesen Pfad, die Wurzel des CVS Verzeichnisbaumes nennt man (`$`)`CVSROOT`. Nehmen wir an, unser (`$`)`CVSROOT` soll unter `/import/projekte/cvsroot` liegen.

Als erstes müssen wir diesen Pfad dem Programm bekanntmachen. Dies geht über die Umgebungsvariable `$CVSROOT` mit `export CVSROOT=/import/projekte/cvsroot` für eine sh/bash oder per `setenv CVSROOT /import/projekte/cvsroot` für eine csh. Man kann den Befehl auch in das entsprechende dotfile - also z.B. `.cshrc` eintragen. Die andere Möglichkeit ist, dass man den Pfad dem cvs Befehl mit angibt. Dann lautet der Aufruf `cvs init` bzw. `cvs -d /import/projekte/cvsroot init`.

2.2.3 Wie sieht ein Repository aus ?

Ein Repository ist ein Verzeichnisbaum, der 1:1 das Projekt bzw. mehrere Projekte beinhaltet bzw. abbildet. Nehmen wir z.B. ein Projekt Taschenrechner. Zu dem Projekt soll es einen Kurzüberblick, ein Verzeichnis mit den Quelltexten und ein Verzeichnis für die Dokumentation geben.

```
| CVSROOT
| Taschenrechner
        | README
        | man
                | ...  <Dateien mit der Dokumentation>
        | src
                | Makefile
                | ...  <Dateien mit den Quelltexten>
```

Weiterhin enthält ein Repository Dateien und Verzeichnisse (z.B. CVSROOT - nicht zu verwechseln mit der Umgebungsvariable `$CVSROOT` !) mit Metainformationen bzw. Informationen zu den verschiedenen Versionen und über das Repository selber. Ebenso sind administrative Dateien enthalten die man entsprechend auch per cvs pflegen kann.

Im obigen Beispiel haben wir nur ein Projekt im Repository. Es ist nicht unüblich nur ein grosses Repository für mehrere verschiedene Projekte zu haben. Ein zweites

Projekt könnte sich z.B. mit einem Browser beschäftigen. Man würde dann einfach ein weiteres Verzeichnis namens `Browser` auf der gleichen Ebene wie `Taschenrechner` haben. An dieser Stelle sei auf die oben auch nur genannten 'Module' verwiesen, die in diesem Dokument nicht explizit behandelt werden. Im filesystem würde das dann so aussehen:

```
/import/projekte/cvsroot/
                        | CVSROOT
                        | Taschenrechner
                        |       | README
                        |       | man
                        |       |       | ...  <Dateien mit der Dokumentation>
                        |       | src
                        |       |       | Makefile
                        |       |       | ...  <Dateien mit den Quelltexten>
                        | Browser
                        |       | ...
```

3 Befehlsübersicht

3.1 Auswahl globaler Optionen

- -H oder –help fuer Kommandozeilenhilfe. Als Option kann man dahinter auch noch den Namen eines Befehls angeben um die genaue Syntax und dessen Optionen angezeigt zu bekommen.
- -d root um den Pfad zum `$CVSROOT` anzugeben; überschreibt $CVSROOT.

3.2 Einige wichtige Befehle für Benutzer

Befehl	Beschreibung
add	Fügt ein Verzeichnis oder eine Datei einem Repository hinzu
checkout	Lokale Kopie aus dem Repository auschecken um Quellcode zu bearbeiten
commit	Änderungen im Repository wirksam machen
diff	Unterschiede zwischen verschiedenen Versionen von Dateien zeigen
help	Für Hilfe (zeigt eine komplette Liste aller Befehle mit kurzer Erklärung)
import	Vorhandene Quelltexte z.B. eines ganzen Projektes initial ins CVS einbuchen
log	Zeigt die Änderungen und die dazugehörigen Kommentare
login	Meldet den Benutzer am entfernten CVS Server an und merkt sich das Passwort
logout	Meldet einen vom CVS Server ab, d.h. löscht das gemerkte Passwort
remove	Löscht eine Datei oder ein Verzeichnis vom Repository
status	Zeigt den Status des ausgecheckten files an
tag	Gibt der aktuellen Version eines (Teils eines) Repositories einen Namen
update	Gleicht die Arbeitsdateien mit denen im Repository ab

3.3 Ein Beispiel

Betrachten wir nocheinmal unser Projekt Taschenrechner. Nehmen wir an, dass ein gewisser Arbeitsstand vorliegt und ab sofort mit cvs auf einem lokalen Server weiterentwickelt werden soll. Folgendes ist zu tun:

3.3.1 Aktuelle Quelltexte importieren

Als erstes begeben wir uns in das Verzeichnis in dem unsere Quelltexte liegen - in unserem Beispiel also das README und die zwei Verzeichnisse `man` und `src`. Dort rufen wir `cvs import` auf.

```
cvs import -m "Import des Projektes Taschenrechner" Taschenrechner MAIN start
```

`-m` gibt dabei eine Beschreibung an, `Taschenrechner` ist das Verzeichnis, welches unterhalb des `$CVSROOT`-Verzeichnisses für dieses Projekt angelegt wird, in das die Dateien hineingeschrieben werden, `MAIN` ist ein sogenannter frei gewählter vendor-tag, eine Beschreibung für den ganzen Zweig und `start` ist ein symbolischer Name für diese Version.

`cvs import` ändert nichts am lokalen Verzeichnis, d.h. man kann nicht mit dem lokalen Verzeichnis weiterarbeiten.

3.3.2 Arbeitskopie das erste Mal besorgen

Um in Zukunft mit den Quelltexten aus dem cvs arbeiten zu können besorgen wir uns eine lokale Kopie aus dem cvs. Dazu sichern wir zuerst unser Ursprungsverzeichnis mit `mv Taschenrechner Taschenrechner.orig`. Nun rufen wir das Kommando

```
cvs checkout Taschenrechner
```

auf. Es wird ein Verzeichnis Taschenrechner angelegt in dem man in Zukunft arbeiten kann. Es werden darin auch Verzeichnisse namens `CVS` angelegt. Darin nichts bearbeiten - sie enthalten Dateien mit Metainformationen.

3.3.3 Dateien bearbeiten

Hmm, ja. Nimm Deinen Lieblingseditor und mach `;-)`

Bearbeiten wir z.B. das file README.

3.3.4 Dateien hinzufügen (löschen)

Sollte man beim Bearbeiten neue Dateien hinzugefügt haben müssen diese mit

```
cvs add filename
```

dem Repository hinzugefügt werden. Die Dateien werden "vorbereitet", sind aber noch nicht im Repository aktiv.

Hätten wir im Verzeichnis `man` eine Datei `taschenrechner.8` erstellt würde diese im Verzeichnis `man` mit

```
cvs add taschenrechner.8
```

hinzugefügt.

Löschen - `cvs remove` - funktioniert analog zu `cvs add`.

3.3.5 Dateien aktualisieren und Änderungen wirksam machen

Es ist gute Praxis nach längerem Arbeiten vor dem Einchecken vom Basisverzeichnis aus, d.h. bei uns im Verzeichnis `Taschenrechner` ein

```
cvs update
```

aufzurufen um ggf. Konflikte vorab zu erkennen. Dieser Befehl gleicht die lokale Kopie nochmals mit dem Repository ab. Keine Angst - die selber geänderten Dateien werden nicht überschrieben. Diesen Befehl sollte man z.B. auch ausführen bevor man anfängt zu arbeiten, damit man immer mit einer aktuellen Version arbeitet.

Um nun die Änderungen wirksam zu machen führt man den Befehl

```
cvs commit
```

aus. Sollten Änderungen gemacht worden sein wird ein Editor gestartet. Dort kann man eine Beschreibung der Änderungen eingeben. Dies geschieht alternativ auch wieder mit der Option `-m`. Bei längeren Beschreibungen bietet sich auch die Option `-F` *`logfile`* an, der ein Dateiname als Paramter folgt. Die Datei sollte den entsprechenden Beschreibungstext enthalten.

Wir könnten sinnvoller Weise z.B. `-m "Dokumentation updated"` angeben.

3.3.6 Unterschiede betrachten

Um die Änderungen einer Datei zu verfolgen kann man

```
cvs log filename
```

angeben. Dort sieht man Änderungsdatum, Login des Bearbeiters, die Beschreibungstext, Versionsnummer, u.a.. Um nun die Änderungen der aktuellen Version zu einer vorigen, z.B. zur Version 1.1 anzusehen ruft man

```
cvs diff -r1.1 filename
```

auf. Die Datei README würde sich in unserem Beispiel hierfür anbieten.

3.3.7 Version "markieren" (tag)

Hat man einen gewissen Entwicklungsstand erreicht, z.B. eine Version 2.0 die man abgeben oder zum Verkauf ausliefern möchte empfielt es sich diesen Stand für später mit einer "Marke" zu versehen, einem symbolischen Namen für diese Version. Dies geschieht für das komplette Verzeichnis und alle Unterverzeichnisse mit

```
cvs tag name .
```

`name` könnte hier z.B. `version_2_0` sein. Alternativ können auch nur einzelne Dateien angegeben werden.

3.4 .cvsignore Dateien

Möchte man sicherstellen, dass bestimmte Dateien nicht in ein Repository eingechecked werden, kann man im aktuellen Verzeichnis eine `.cvsignore` -Datei ablegen. `.cvsignore` -Dateien werden im Normalfall auch nicht in das Repository eingechecked. Als Inhalt kann man "patterns" schreiben, pro Zeile oder per Leerzeichen getrennt. Ein Pattern kann ein Dateiname oder Teile mit z.B. '*' und '?' sein. '?' steht für genau ein beliebiges Zeichen, '*' für Null oder mehrere beliebige Zeichen. Weitere Patterns können in `sh` Style Syntax angegeben werden. Ein Beispiel für eine solche Datei wäre:

```
config.cache config.log
config.status
*~ .#* *.bak
*.o *.class
.make.state
RCS
```

4 Programme

4.1 UNIX/Linux

Unter UNIX/Linux ist bei den cvs sourcen auch ein Kommandozeilenprogramm namens `cvs` (oben einführend erklärt) dabei. Sowohl von KDE (Cervisia) als auch vom gnome Desktop-Projekt (Pharmacy) gibt es entsprechende grafische Frontends. Des

weiteren existiert ein in Qt geschriebenes Programm namens LinCVS. Der GNU Editor Emacs hat einen eingebauten CVS Modus namens “VC Mode”.

4.2 Windows

Für Windows gibt es z.B. unter http://www.wincvs.org/ einen CVS Client.

5 Literatur

5.1 info / manpages

- info cvs
- [2] manpages zu rcs: rcsintro(1), rcs(1), co(1), ci (2), rcsdiff(1), rlog(1)

5.2 Bücher

- CVS Procket Reference, Gregor N. Purdy, 78 Seiten, 2000, O’Reilly, ISBN 0-596-00003-0 *(engl.)*
- CVS kurz und gut. Source Code Management, Gregor N. Purdy; Ahmet Ertem, 88 Seiten, 2000, O’Reilly, ISBN 3-897-21229-3 *(dt.)*
- Open Source Development with CVS - 2nd Edition, Karl Fogel; Moshe Bar, 368 Seiten, 2001, Coriolis, ISBN 1-588-80173-X *(engl.)*
- Open Source-Projekte mit CVS, Karl Fogel, 442 Seiten, 2000, mitp-Verlag, ISBN 3-8266-0628-0 *(dt.)*

5.3 Links

- [1] http://www.cvshome.org/ , die CVS Homepage; enhält u.a. den “Cederqvist” - DIE CVS Dokumentation neben den info pages.
- http://www.loria.fr/~molli/cvs-index.html
- http://cvsbook.red-bean.com/[1]

[1] Alle Angaben ohne Gewaehr. Keine Gewaehr auf Vollstaendigkeit und Richtigkeit der Angaben. Tippfehler enthalten ;)