CVS – eine kleine Einführung

Kathleen Krebs
TU-Dresden
Fakultät Informatik

1 Was ist CVS?

CVS steht für *Concurrent Versions System* und basiert auf dem *Revision Control System* – RCS. RCS wurde entwickelt, um mehrere Versionen einer Datei zu verwalten. Die Arbeitsweise von RCS ist simpel. Alle Änderungen an der Datei werden protokolliert. Mit Hilfe dieses Protokolls können automatisch alle älteren Versionen der Datei zurückgeholt werden. Diese Methode spart im Vergleich zu Sicherheitskopien Speicherplatz. Außerdem ist es möglich, verschiedene Versionen der Datei zu vergleichen oder zu einer Version zusammenzuführen.

CVS erweitert die Kontrollstruktur zu einer hierarchischen Sammlung von Verzeichnissen mit Dateien unter Versionskontrolle. Nunmehr ist es also möglich, verschiedene Versionen mehrerer Dateien innerhalb eines Verzeichnisbaums zu verwalten.

CVS ermöglicht das gleichzeitige Editieren einer Datei durch mehrere Nutzer, ohne das Informationen verloren gehen. Dazu arbeitet jeder Nutzer mit einer Arbeitskopie (lokal im Arbeitsverzeichnis). Weiterhin existiert eine Hauptkopie – das *Repository*, welche alle Informationen enthält, um alte Versionen wieder herzustellen. Die einzelnen Nutzer aktualisieren das Repository mittels CVS-Befehlen (genauso bei der Arbeitskopie). Dabei prüft CVS, ob es Versionskonflikte gibt, beseitigt diese automatisch oder meldet sie dem Nutzer (wenn keine automatische Auflösung möglich ist).

Versionen können auch speziell (z.B. durch Schlüsselwörter) gekennzeichnet werden. Vorteil dabei ist es u.a., daß bestimmte Versionen direkt durch ein Schlüsselwort angesprochen werden können. Bereits RCS realisiert diese Funktion für einzelne Dateien. Eine Erstellung von Software-Release aus einer Sammlung von Verzeichnissen und Dateien wird ebenfalls unterstützt.

2 Voraussetzungen

Für das Softwaretechnologie-Praktikum wird ein vorbereitetes CVS-Respository bereitgestellt. Es ist im jeweiligen Gruppenverzeichnis zu finden (`/usr/users/swt2p/swt01-1/cvs` für Gruppe 1). Um damit bequem arbeiten zu können, sollten einige Voraussetzungen erfüllt sein:

CVSROOT . Die Variable CVSROOT muß unbedingt gesetzt werden. Je nachdem, ob man mit einem Entwicklungswerkzeug oder auf der Kommandozeile arbeitet, gibt es verschiedene Möglichkeiten. Bei der Arbeit auf der Kommandozeile kann die Variable in den Pfad eingetragen oder direkt beim ersten Login angegeben werden. Ersteres geschieht unter Unix z.B. mittels 'setenv', entweder auf der Kommandozeile oder aber in der `.cshrc` (wenn man mit der tcsh als Shell arbeitet). Das sieht dann so aus:

```
setenv CVSROOT :ext:<login>@irz601.inf.tu-dresden.de:/usr/users/swt2p/swt02-1/cvs
```

`ext` gibt die Zugriffsmethode auf den CVS-Server an. Es gibt zwei Möglichkeiten, `ext` und `pserver`. Für das Praktikum wird die `ext`-Methode verwendet. Dabei verläuft der Zugriff mittels *Secure Shell* (ssh).

Weiterhin wird das Login und der CVS-Server angegeben. In diesem Fall ist der Server `irz601.inf.tu-dresden.de`. Anschließend wird das CVSROOT-Verzeichnis angegeben.

Diese Zeile kann auch in die `.cshrc` geschrieben werden. Damit erspart man sich, diese Zeile jedesmal neu einzugeben.

Nach dem ersten `'checkout'` (siehe weiter unten), ist das Setzten dieser Variable nicht mehr notwendig, da dieser Wert in einer Datei festgehalten, die CVS kennt (aber dazu später mehr).

Die Angabe der CVSROOT-Variable beim Einloggen wird speziell beim Login erklärt.

CVSROOT als auch andere Parameter müssen ebenfalls in Entwicklungswerkzeugen gesetzt werden. Dazu sollte die jeweilige Hilfe der konsultiert werden.

CVS_RSH . Wie schon erwähnt erfolgt der Zugriff auf den CVS-Server mittels ssh. Voreingestellt ist der Zugriff mit rsh. Um dies zu ändern, muß die Variable CVS_RSH auf ssh gesetzt werden.

```
setenv CVS_RSH ssh
```

Wie auch schon `CVSROOT` kann diese Zeile ebenfalls in die `.cshrc` geschrieben werden.

CVSEDITOR ist standartmäßig der vi. Wer diesen nicht nutzen möchte, aber auf der Kommandozeile arbeitet, setzt diesen Wert einfach auf seinen Lieblingseditor, z.B.:

```
setenv CVSEDITOR joe
```

Hier ist es sinnvoll die Variable in der `.cshrc` fest zu setzten.

Arbeitsverzeichnis Da die Arbeit nicht direkt im CVS-Repository stattfindet, wird noch ein Arbeitsverzeichnis benötigt, in dem eine Arbeitskopie des Repositories (bzw. einem Ausschnitt davon) existiert. Es sollte also vorher angelegt werden.

3 Das CVS-Repository

Einzelheiten bzgl. des CVS-Repositories sollen an dieser Stelle nicht betrachtet werden. Wichtig zu wissen ist, daß es Module und Verzeichnisse gibt. *Module* legen eine logische Struktur der Dateien fest und können selber *Verzeichnisse* enthalten. Sinnvoll ist z.B. ein Modul für den Quellcode und eins für die Dokumentation. Wieviel Module man benötigt und wie diese heißen sollen legt man am besten zu Begin der Arbeit fest (später können natürlich ohne Probleme neue Module angelegt werden). Angelegt werden Module mit dem **import**-Kommando, welches in der Kommandoübersicht (Abschnitt 6) erklärt wird.

4 Eine CVS-Sitzung

Wie arbeitet man nun mit CVS? Das ist ganz einfach:

1. Nachdem in das Arbeitsverzeichnis gewechselt wurde, können die benötigten Daten *'ausgecheckt'* werden. Dieser Vorgang erstellt eine Arbeitskopie des Repositories (bzw. dem gewünschten Teil davon) an

   ```
   cvs checkout impl
   ```

   z.B., legt eine Arbeitskopie des kompletten impl-Moduls an Voraussetzung ist natürlich, das vorher ein solches Modul angelegt wurde und bereits Daten existieren). CVS erzeugt im Arbeitsverzeichnis ein Verzeichnis namens `impl` dafür, in dem alles zu finden ist.

   Existiert im Arbeitsverzeichnis bereits eine Arbeitskopie und man möchte die neuesten Änderungen der anderen übernehmen, geschieht dies mittels (um beim Beispiel mit 'impl' zu bleiben):

   ```
   cvs update impl
   ```

   Wenn CVSROOT vorher nicht gesetzt wurde und das die erste Sitzung ist, sieht der Befehl zum *'auschecken'* folgendermaßen aus (am Beispiel von Gruppe 1):

   ```
   cvs -d :ext:<login>@irz601.inf.tu-dresden.de:/usr/users/swt2p/swt02-1/cvs checkout impl
   ```

   Zusätzlich zum impl-Verzeichnis wird ein CVS-Verzeichnis angelegt, hier werden alle - für CVS - wichtigen Daten abgelegt, u.a. auch CVSROOT. Somit ist es bei späteren Sitzungen nicht mehr nötig all das anzugeben. Natürlich nur, so lange man sich in dem Verzeichnis befindet, in dem auch das CVS-Verzeichnis zu finden ist.

2. Nachdem die nötigen Dateien ins Arbeitsverzeichnis kopiert wurden, kann damit gearbeitet werden.

3. CVS übernimmt nur die schon im Repository vorhandenen Dateien. Neue Dateien/Verzeichnisse müssen 'von Hand' hinzugefügt, werden. Dies geschieht mittels:

   ```
   cvs add nocheinedatei.java
   ```

   Wobei man sich an der Stelle im Arbeitsverzeichnis befinden muß, an dem diese Datei steht. Diese wird dann an genau dieser Stelle ins Repository eingefügt.

4. Abschließend überträgt (*'einchecken'*) man die Änderungen ins Repository (auch nach dem hinzufügen von Dateien bzw. Verzeichnissen notwendig, ansonsten werden diese nicht geltend gemacht):

   `cvs commit impl`

   CVS sucht dabei nach veränderten Dateien und versucht diese konfliktfrei ins Repository zu übernehmen.

   Sollte es doch zu Konflikten kommen, die nicht automatisch beseitigt werden können, wird dies mitgeteilt und ein Editor geöffnet. Dort müssen die Konflikte von Hand beseitigt werden.

   Hat man nicht bereits mit der Erweiterung '-m "Kommentar"' ein Statement abgegeben, wird außerdem dazu aufgefordert mit Hilfe des gewählten Editors dies nachzuholen. Der Kommentar dient dazu kurz eine Bemerkung zu den Änderungen abzugeben, z.B.

   `Joystick-Unterstuetzung eingebaut`

   Die bereits oben erwähnte zweite Möglichkeit dies zu tun sieht z.B. wie folgt aus:

   `cvs commit -m ''Joystick-Unterstuetzung eingebaut'' impl`

Zum Schluß noch ein Hinweis. Ein Einchecken ist nur möglich, wenn man vorher ausgecheckt hat!!! Außerdem bitte immer den Aufwand des Ein- und Auscheckens betreiben. Einfaches Rein und Rauskopieren erfüllt nicht den gewünschten Zweck.

5 Binäre Daten

Ein Problem gibt es noch mit CVS: Die Verwaltung von binären Daten. Das sind z.B. gif-Bilder, Klassen-Dateien etc. CVS geht bei den eingecheckten Dateien generell davon aus, daß es sich um Textdateien handelt, und speichert Versionsinformationen darin ab.

Wie bringt man CVS nun bei, daß es sich bei der entsprechenden Datei um eine binäre Datei handelt? Dies geschieht durch einen zusätzlichen Parameter (`-kb`) beim Hinzufügen von Dateien:

`cvs add -kb impl/pics/buntesItem.gif`

6 Kommandoübersicht

`cvs add [-k kflag] [-m message] files`

Stellt die angegebenen Dateien im Arbeitsverzeichnis unter CVS-Kontrolle. Beim nächsten `'commit'` werden diese Änderungen mit in das Repository übernommen.

`kflag` legt die Art & Weise der Schlüsselexpansion fest. Wichtig ist dies in Bezug auf binäre Dateien, diese werden mit der Angabe '`-kb`' dem Repository hinzugefügt. (Schlüssel sind z.B. `$Id$`, `$Log$`, die dann von CVS mit entsprechenden Werten gefüllt werden, was bei binären Daten nicht erwünscht ist.)

Die `message` entspricht der Änderungsbeschreibung bei einem `'commit'`.

cvs checkout [options] modules

Sollte nur im Arbeitsverzeichnis ausgeführt werden. Erzeugt Kopien der entsprechenden CVS-Dateien aus den angeführten Modulen. Diese können dann bearbeitet werden.

`checkout` bietet jede Menge Optionen, die jedoch im Normalfall nicht von größerem Nutzen sind (für weitere Informationen siehe auch Abschnitt 7).

cvs commit [options] files

Übernimmt die Änderungen an den aufgeführten Dateien im Arbeitsverzeichnis in das Repository. Neu erstellte bzw. umbenannte Dateien werden nicht berücksichtigt (siehe `'add'` und `'remove'`).

Die wichtigste Option dürfte '`-m message`' sein, sie gibt die Änderungsbeschreibung für alle angegebenen Dateien an.

cvs import [options]

Wird verwendet um viele Dateien oder Verzeichnisse auf einem Schlag einem Projekt hinzuzufügen oder um ein neues Projekt zu kreieren. Um einzelne Dateien hinzuzufügen sollte aber besser `add` verwendet werden.

Die wichtigsten Optionen sollen an einem Beispiel erklärt werden:

```
cvs import -m ''Neues Modul angelegt'' impl paul start
```

Mit `-m message` wird eine kurze Beschreibung angegeben, danach kommt der Modulname. Im Repository wird nun ein Verzeichnis mit diesem Namen angelegt, über den dann alle `checkout`-Befehle gehen. Die beiden anderen Parameter geben den 'Ersteller' und die Version an, was aber nicht weiter relevant für die weitere Arbeit ist, sondern nur Verwaltungsinformationen für das CVS sind.

Das angelegte Modul enthält dann alle Dateien und Verzeichnisse, die sich im aktuellen Pfad befinden. Soll das Modul leer sein, sollte man sich vorher in ein leeres Verzeichnis begeben. Ansonsten den `import`-Befehl von da aus aufrufen, wo sich die Dateien befinden, die das Modul enthalten soll.

cvs remove [options] files

Entfernt die angegebenen Dateien aus dem Repository. Beim nächsten `'commit'` werden diese Änderungen in das Repository übernommen.

Mit der Option `'-f'` werden die angegebenen Dateien auch in der Arbeitskopie gelöscht.

`cvs update [options] files`

Aktualisiert die angegebenen Dateien im Arbeitsverzeichnis.

Mit der Option `'-d'` werden in der Arbeitskopie noch nicht vorhandene Verzeichnisse angelegt.

7 Weitere Informationsquellen

Wenn gerade niemanden da ist, der als Datenquelle benutzen werden kann, bedeutet das noch nicht das Ende.

Eine Möglichkeit sind `man`- und `info`-Seiten unter Unix. Eine weitere (für die, die besser mit Links etc. klar kommen) ist auch das WWW, z.B.

`http://www.cvshome.org/`

Dort gibt es ein Manual, in dem ebenfalls alles erklärt ist.