Concurrent Versions System

Ausarbeitung zum Vortrag im Rahmen des Proseminars Unix Tools
Technische Universitaet Muenchen
WS2005/2006

Siarhei Trushyn

14. Dezember 2005

Inhaltsverzeichnis

1 Einleitung

Concurrent Versions System (CVS) bezeichnet ein Kommandozeilenprogramm zur Versionsverwaltung von Dateien und insbesondere von Programmquelldateien.

1.1 Geschichte

CVS ist eine Weiterentwicklung von RCS, das ebenfalls eine Versionsverwaltung ermöglichte, diese jedoch nur auf einzelne Dateien beschränkte. Hauptsächlich diese Schwäche sollte mit der Entwicklung von CVS behoben werden.
Erste Entwicklungen an Concurrent Versions System wurden im Jahre 1986 von Dick Brune durchgeführt, als er seine erste Ideen und Konzepte in vielen Shell-Skripten niedergeschrieben hatte, die jedoch nicht als endgültige Implementierung, sondern als Entwürfe anzusehen waren. Die endgültige Entwicklung von CVS ist 1989 von Brian Berliner und Jeff Polk gemacht worden, die die wesentlichen Konzepte und Ideen aus den Shell-Skripten von Dick Brune übernommen haben.
Heute erfreut sich CVS grosser Beliebtheit in der Open Source Welt und wird beispielsweise von solchen grossen Projekten wie Apache, Firefox und Samba benutzt. Aber auch kommerzielle Configuration-Management-Werkzeuge benutzen die Funktionalität von CVS für ihre Zwecke.
Heutzutage wird CVS nach und nach durch eine Weiterentwicklung wie SVN ersetzt, bei der die meisten Schwächen von CVS behoben worden sind.

1.2 Einsatzmöglichkeiten

An dieser Stelle gilt es nun zu klären, was genau CVS einem Entwickler an Einsatzmöglichkeiten bietet:

- **Versionskontrollverwaltung**
  Dies ist die Hauptfunktionalität von CVS, welche einem Entwickler erlaubt alle Versionen eines Softwareprojekts zu verwalten, einzusehen und bei Bedarf die früheren wiederherzustellen. Somit eröffnet sich die Möglichkeit einen Überblick über die Geschichte des Projekts zu behalten.

- **Gemeinsame Entwicklung von vielen Personen**
  CVS erlaubt es einer ganzen Projektgruppe gleichzeitig an einem Projekt zu entwickeln und sich dabei nicht gegenseitig zu behindern. Es existieren in CVS entsprechende Mechanismen, die Konflikte zwischen den Entwicklern aufzeigen oder selbstständig beheben.

- **Speicherplatzersparnis**
  In einem CVS Versionsverwaltungssystem werden nicht alle Versionen eines Projekts einzeln für sich gespeichert. Vielmehr werden an einer zentralen Stelle nur die inhaltlichen Äenderungen protokolliert und gespeichert. Diese Vorgehensweise hat den Vorteil, dass der Speicherplatz effizient genützt wird und dabei alle Versionen trotzdem jederzeit einsehbar und wiederherstellbar sind.

- **Open Source Lizenz**
  CVS unterliegt der Open-Source-Lizenz und darf deswegen von jedem frei benutzt und verbreitet werden.

Es gilt jedoch ebenfalls zu erwähnen was CVS einem Entwickler nicht bietet. Die im Folgenden aufgeführten Punkte sind nicht als Nachteil von CVS zu betrachten, sondern wurden gezielt nicht in CVS integriert.

- **kein Entwicklungssystem**
  CVS schreibt in keiner Weise vor, wie ein Entwickler seine Projekte technisch und inhaltlich aufbauen soll. Es unterstützt einen mehr in der Weise, dass es für einen die komplette Übersicht über das Projekt behält.
- **kein Ersatz für Projektleitung oder Kommunikation zwischen Entwicklern**
  CVS ist nicht in der Lage die Leitung eines Projekts zu ersetzen und die Kommunikation zwischen den Entwicklern wird zwar unterstützt, aber auf keinen Fall ersetzt.
- **keine automatische Behebung von Fehlern oder Verifizierung**
  CVS wurde nicht dafür entwickelt Prgrammierfehler in einem Projekt zu finden oder gar die ganze Software zu verifiezeren. Dieser Aufgabe muss sich der Entwickler widmen oder spezielle Software für diese Zwecke verwenden.

2 Archiv

Das Archiv oder auch Repository genannt, bezeichnet die zentrale Einheit in CVS, in der komplette Kopien von Dateien und Verzeichnissen, die sich unter der Versionsverwaltung befinden, gespeichert werden.

2.1 Benutzung

Um CVS zu benutzen ist es erforderlich, dass man auf ein solches Archiv zugreift. Hierfür sind die entspechenden CVS Befehle zu verwenden, die die Verbindung mit dem Archiv automatisch erledigen und die erforderlichen Anpassungen an dem Archiv durchführen. Im Allgemeinen ist es nicht notwendig sich Gedanken über das Repository zu machen, denn die CVS Befehle verrichten die ganze Arbeit für einen.
In manchen Fällen ist es jedoch unabdingbar, das Achiv auch per Hand zu ändern. Diese Sonderfälle zeichnen auch gleichzeitig auf die Schwächen von CVS hin. So ist beim Umbenennen und Verschieben von Dateien und Verzeichnissen ein manuelles Eingreifen des Benutzers erforderlich.

2.2 Aufbau

Auf der Dateisystemebene betrachtet, ist ein CVS Archiv nichts anderes als ein Verzeichnis, in dem ein `CVSROOT` Verzeichnis und die Verzeichnisse für die jeweiligen Projekte angelegt sind. Das `CVSROOT` Verzeichnis beinhaltet Verwaltugsinformationen, die das gesamte Archiv betreffen. In den jeweiligen Projektverzeichnissen befinden sich ',v'-Dateien, die alle Änderungsinformationen

der Dateien enthalten. Jede Datei, die sich unter Versionsverwaltung befindet, besitzt eine ',v'-Datei. Da alle Änderungen gespeichert werden, sind sie dementsprechend reproduzierbar.

2.3 Anlegen eines Archivs

Um ein Archiv anzulegen, ist es nur erforderlich, das Verzeichnis, in dem das Archiv liegen soll, als solches zu deklarieren. Das geschieht mit dem entsprechenden CVS Befehl. Soll beispielsweise das Archiv in `/usr/local/cvsroot` liegen, würde der Aufruf folgendermassen aussehen:

```
> cvs -d /usr/local/cvsroot init
```

Es ist auffallend, dass CVS sofort nach diesem Aufruf ein Verzeichnis `CVSROOT` in dem Archiv anlegt.
Um das Archiv lokal auf dem gleichen Rechner zu benutzen, ist nichts weiter notwendig. Soll jedoch CVS auf Anfragen aus dem Internet antworten, ist eine weitere Konfiguration notwendig, die ein interessierter Leser der offiziellen CVS Dokumentation entnimmt.

2.4 Zugriff

Um auf ein Archiv zuzugreifen, muss zuerst der Pfad zu dem Archiv, auf das zugegriffen werden soll, festgelegt werden. Der Pfad kann entweder in eine globale Shell-Variable eingetragen werden

```
> export CVSROOT=<Pfad zum Archiv>
```

oder muss bei bestimmten Befehlen immer angegeben

```
> cvs -d <Pfad zum Archiv> <Kommando>
```

werden.
Es gibt drei wichtigste Zugriffsmethoden:

- **Lokales Archiv**
  In diesem Fall befindet sich das Archiv lokal auf dem Rechner. Hier muss nur der Pfad zum Archiv gesetzt werden:

  ```
  > export CVSROOT=/usr/local/cvsroot
  ```

- **Entferntes Archiv - PSERVER**
  Hier handelt es sich um einen passwortauthentisierten Server, bei dem eine einmalige Anmeldung mit dem Passwort zur Authentisierung ausreicht.

  ```
  > export CVSROOT=:pserver:alex@srv:/usr/local/cvsroot
  > cvs login #nur einmalig durchzufuehren
  ```

  Hier wird der Benutzer `alex` am Server `srv` angemeldet.

- **Entferntes Archiv - SSH**
  Bei dieser Zugriffsmethode baut man einen gesicherten Tunnel zum Server auf, über den alle Daten übertragen werden.

```
> export CVSROOT=:ext:alex@srv:/usr/local/cvsroot
> export CVS_RSH=ssh
```

Hier wird ebenfalls der Benutzer `alex` am Server `srv` angemeldet. Zusätzlich legt man über die globale Variable `CVS_RSH` fest, welches Programm zur gesicherten Übertragung genutzt werden soll. In diesem Fall ist es `ssh`.

3 Projekt mit CVS

In diesem Kapitel wir dem Leser erläutert, wie man ein Projekt mit CVS verwalten und benutzen kann.

3.1 Anlegen eines neuen CVS Projekts

Um ein Projekt unter die Verwaltung von CVS zu stellen, muss man nur in das Verzeichnis wechseln und den ensprechenden Befehl `import` auszuführen.

```
> cd projekt
> cvs import -m "Projekt01" projekt01 unix-tools start
N projekt01/Makefile
N projekt01/main.c
No conflicts created by this import
```

CVS durchsucht hierbei das ganze Verzeichnis rekursiv und übernimmt alle Dateien und Unterverzeichnisse. Mit der Option `-m` gibt man ein Initialkommentar für das Projekt an. `Projekt01` bezeichnet den Namen des Projekts, das allerdings mit dem Projektverzeichnisnamen nicht übereinstimmen muss. `Unix-tools` und `start` bezeichnen die Hersteller- und Verkauftags, die jedoch an dieser Stelle nicht von Bedeutung sind.
Ist ein Projektverzeichnis noch nicht vorhanden, so gilt es stets dieses anzulegen und nach der oberen Beschreibung in das Archiv zu übernehmen.

3.2 Auschecken einer Arbeitskopie

Um mit einem Projekt aus dem CVS Archiv zu arbeiten, ist es notwendig sich zuerst eine Arbeitskopie aus dem Archiv zu holen. Dies geschieht mit dem Befehl `checkout`.

```
> cvs checkout projekt01
cvs checkout: Updating projekt01
U projekt01/Makefile
U projekt01/main.c
> cd projekt01
> ls
CVS Makefile main.c
```

Nach dem Befehl `checkout` muss der Name des Projekts angegeben werden, welches 'ausgecheckt' werden soll. Es ist ebenfalls auffallend, dass in der bezogenen Arbeitskopie ein Verzeichnis `CVS` dazugekommen ist. In diesem sind Verwaltungsinformationen wie z.B. das Archiv, dem die Arbeitskopie entstammt, gespeichert.

3.3 Änderungen anschauen

Um festzustellen, welche Dateien in der Arbeitskopie geändert wurden, ist der Befehl `update` hilfreich.

```
> cvs -n update
cvs update: Updating .
M main.c
```

Die Option `-n` gibt man dabei an, dass keine eventuelle schreibende Operationen erwünscht sind. Die Arbeitskopie wird dabei rekursiv durchsucht und listet alle Dateien, bei denen sich der Zustand gegenüber dem Archiv geändert hat. Zum Beispiel bedeutet `M` in dem oberen Bespiel, dass die Datei vom Benutzer verändert wurde. An dieser Stelle ist jedoch zu bemerken, dass mit dem Befehl `update` ebenfalls festgestellt werden kann, was andere Entwickler gemacht haben.
Um sich die inhaltlichen Änderungen anzuschauen, kann der CVS Befehl `diff` benutzt werden.

```
> cvs diff main.c
cvs diff: Diffing main.c
Index: main.c
===================================================
RCS file: /usr/local/cvsroot/projekt01/main.c,v
retrieving revision 1.1
diff -r1.1 main.c
4c4
< printf("hello wold!");
---
> printf("hello world!");
```

Der Befehl `diff` vergleicht die möglicherweise veränderten Dateien der Arbeitskopie mit den entsprechenden Gegenstücken aus dem Archiv. Die Unterschiede werden dabei auf dem Bildschirm ausgegeben, so dass der Benutzer diese vergleichen kann. Mit der Option `-c` wird die Ausgabe etwas leserlicher gestaltet und die Unterschiede sind somit leichter nachzuvollziehen.

3.4 Änderungen speichern

Mit dem Befehl `commit` können alle gemachten Änderungen in das Archiv übertragen werden.

```
> cvs commit -m "Fehler behoben"
cvs commit: Examining .
/usr/local/cvsroot/projekt01/main.c,v <-- main.c
new revision: 1.2; previous revision: 1.1
```

Mit der Option `-m` wird ein Kommentar angegeben, der die Änderungen charakterisiert. Die Kommentarangabe ist bei dem Befehl `commit` Pflicht. Wird keine Datei angegeben, so durchsucht CVS rekursiv das ganze Verzeichnis und übermittelt alle Änderungen an das Archiv. Dabei wird bei jeder Datei, die verändert wurde, die Revisionsnummer inkrementiert.

3.5 Mehrere Benutzer

Da CVS das Arbeiten an dem gleichen Projekt mehreren Entwicklern erlaubt, kann ein Benutzer im Verlauf der Entwicklung vor dem folgendem Problem stehen:

```
> cvs commit
cvs commit: Examining .
cvs commit: Up-to-date check failed for ‘main.c’
[...]
```

Mit dieser Ausgabe signalisiert CVS dem Benutzer, dass seine Arbeitskopie nicht mehr die aktulle Revisionsnummer hat. Diese Tatsache ist dadurch bedingt, dass inzwischen andere Benutzer das Projekt verändert haben, in dem sie eine oder mehrere Dateien angepasst haben. In einem solchen Fall müssen die unterschiedlichen Revisionen zusammengeführt werden.

```
> cvs update
cvs update: Updating .
RCS file: /usr/local/cvsroot/projekt01/main.c,v
retrieving revision 1.3
retrieving revision 1.4
Merging differences between 1.3 and 1.4 into main.c
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in main.c
C main.c
```

In diesem Fall vereinigt der Befehl `update` die Datei aus dem Archiv und die aus dem Arbeitsverzeichnis. Das Projektverzeichnis wird dabei rekursiv durchsucht und nur die betroffenen Dateien werden vereinigt.
Bei diesem Vorgang können Konflikte auftreten. Ein Konflikt tritt immer dann auf, wenn zwei Benutzer die gleichen Stellen in der gleichen Datei verändert haben. Im Falle eines Konflikts fügt CVS an der entspechender Stelle beide Lösungen in die Datei ein. Der Benutzer muss sich dann für eine Lösung entscheiden, indem er die Datei editiert und die nicht gewünschten Stellen einfach löscht.

```
[...]
<<<<<<< main.c
for (i = 0; i < size; i++)
=======
for (i = 0; i < size - 1; i++)
>>>>>>> 1.4
[...]
```

Auf keinen Fall sollte am Ende dieses Vorgangs ein `commit` vergessen werden, der die gewünschte Version der Datei in das Archiv überträgt.

3.6 Überblick behalten

Um einen Überblick über die Geschichte der Dateien zu bekommen, ist der Befehl `log` sehr nützlich.

```
> cvs log main.c
RCS file: /usr/local/cvsroot/projekt01/main.c,v
Working file: main.c
[...]
----------------------------
revision 1.1
date: 2005-10-20 08:40:07 +0000; author: set; [...]
branches: 1.1.1;
Initial revision
----------------------------
revision 1.1.1.1
date: 2005-10-20 08:40:07 +0000; [...] lines: +0 -0
Projekt01
```

In einer gut gegliederten und überschaulichen Ausgabe werden alle für die Datei gemachten Commits aufgelistet. Für jedes Commit sind auch solche Angaben wie Revisionnummer, Kommentar, Datum und Uhrzeit, Benutzername und sogar die Anzahl der geänderten Zeilen zu finden. Damit ist es eine gute Möglichkeit zu erfahren, was sich mit den einzelnen Datein in der Geschichte des Projekts abgespielt hat.

3.7 Änderungen zurücknehmen

Sollte einmal die Notwendigkeit bestehen, die gemachten Änderungen zurückzunehmen, kann dies mit CVS sehr leicht realisiert werden.
Als erstes gilt es zu erkennen, welche Änderungen genau zurückgenommen werden müssen. Für diesen Zweck kann der praktische Befehl `diff` verwendet werden. Mit der zusätzlichen Option `-r` können die Unterschiede zwischen zwei Revisionen nachvollzogen werden.

```
cvs diff -c -r 1.4 -r 1.5 hello.c
Index: main.c
===================================================
RCS file: /usr/local/cvsroot/projekt01/main.c,v
retrieving revision 1.4
retrieving revision 1.5
diff -r1.4 -r1.5
> printf("size: %d\n", size);
```

In diesem Beispiel sind die Revisionen 1.4 und 1.5 verglichen worden. In diesem Fall ist eine neue Zeile mit dem Aufruf der Funktion `printf` dazugekommen. Wird bei dem Befehlsaufruf nur eine Revision angegeben, so wird mit der aktuell 'ausgecheckten' Revision verglichen.
Es ist jedoch nicht erwünscht in der Vergangenheit zurückzugehen. Vielmehr will man den Zustand aus der Vergangenheit auf den aktuellen Zeitpunkt übertragen, so dass die durchgeführte Korrektur ebenfalls rückgängig gemacht werden kann. Somit geht die zurückgenommene Revision nicht verloren, sondern bleibt vielmehr ein Teil der Historie und kann jederzeit wiederhergestellt werden.
Um die Änderungen zurückzunehmen, wird zuerst der Befehl `update` herangezogen, um die aktuelle Arbeitskopie auf den neuesten Stand zu bringen. Ansch-

liessend wird wieder `update` mit der Option `-j` benützt, um die Änderungen zwischen zwei Revisionen rückgängig zu machen.

```
> cvs update
> cvs update -j 1.5 -j 1.4 main.c
> cvs commit
```

Im obigen Beispiel sind die Änderungen zwischen den Revisionen 1.4 und 1.5 rückgängig gemacht worden. Es ist dabei wichtig auf die Reihenfolge der Revisionen zu denken, sonst bleibt der gewünschte Effekt aus.

3.8 Aufräumen

Nachdem man mit der Arbeit am Projekt fertig ist und die Arbeitskopie löschen möchte, ist es eine gute Angewohnheit sich zuerst von dem Projekt mit dem Befehl `release` zu lösen. Denn in diesem Fall weist CVS den Benutzer auf eventuelle noch nicht gespeicherte Daten hin. Mit der Option `-d` wird ebenfalls das Verzeichnis gelöscht.

```
> cvs release -d projekt01
```

Akzeptable Vorgehensweise ist jedoch auch die vorherige Überprufung auf vorhandene nicht gespeicherte Änderungen in dem Archiv mit dem Befehl `update`, bevor man die Arbeitskopie endgültig löscht.

3.9 Nützliche Kommandos

Im Folgenden sind noch einige nützliche Befehle aufgelistet:

- **Dateien hinzufügen**
  ```
  > cvs add main.c
  > cvs commit
  ```
- **Dateien entfernen**
  ```
  > rm main.c
  > cvs remove main.c
  > cvs commit
  ```
- **Verzeichnisse hinzufügen**
  ```
  > cvs add bin
  ```
- **Binärdateien hinzufügen**
  ```
  > cvs add -kb main
  > cvs commit
  ```

4 Momentaufnahmen - Tags

Manchmal ist es sehr nützlich sich den Zustand eines Projekts in der Vergangenheit anzuschauen (z.B. stabile Version der Software). Da in CVS jedoch jede Datei eine eigene Revisionsnummer besitzt und diese sich im Laufe der Zeit stark von denen der anderen Dateien unterscheiden kann, wäre es sehr schwer sich die

Revisionsnummer jeder einzelnen Datei zu jedem Zeitpunkt der Entwicklung zu merken.
Für diesen Zweck bietet CVS die Möglichkeit des 'Tagging'. Dabei bildet CVS eine Gruppe von Revisionen, reprasentiert durch die aktuelle Arbeitskopie, auf einen Tag-String ab.

```
> cvs tag rel-01
cvs tag: Tagging .
T Makefile
T main.c
```

Im oberen Beispiel ist die aktuelle Arbeitskopie und damit auch die Revisionen der Dateien im Projekt mit einem Tag `rel-01` markiert. Über dieses Tag kann der Benutzer später auf genau diesen Zustand des Projekts zugreifen. Zum besseren Verständnis kann man Tags auch mit zeitlichen Momentaufnahmen vergleichen.
Es ist zu beachten, dass nach dem Tagging kein `commit` erforderlich ist.

4.1 Praktische Anwendung von Tags

Im Folgenden sind einige Anwendungsbeispiele von Tags aufgelistet:

- `cvs checkout -r rel-01 projekt01`
  Mit diesem Befehl holt man z.B. das Projekt mit dem Namen `projekt01` in dem Zustand als es mit `rel-01` markiert worden ist.
- `cvs diff -r rel-01 main.c`
  Hier vergleicht man die aktuelle Version der Datei `main.c` mit dem Zustand, als es mit `rel-01` markiert worden ist.
- `cvs update -r rel-01`
  Dieser Befehl wandelt die aktuelle Arbeitskopie in diejenige mit dem Tag `rel-01` um.

4.2 Sticky Tags

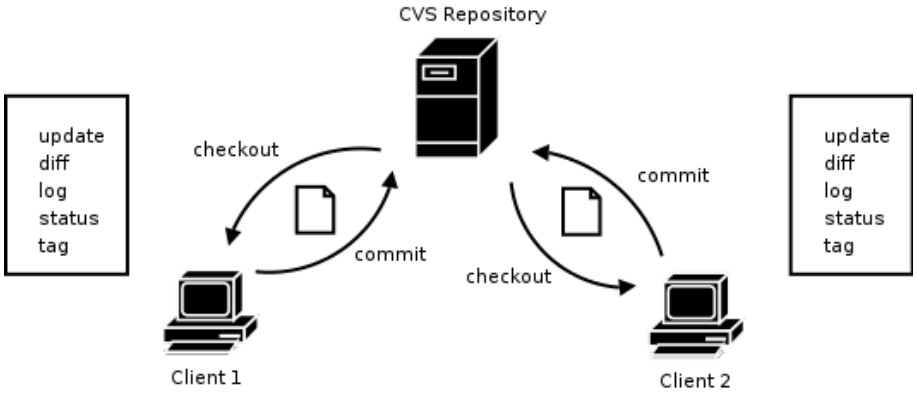
Holt man sich über Tags eine Version des Projekts aus der Vergangenheit, verändert es und versucht anschliessend die Änderungen abzuspeichern, so tritt der Folgende Sachverhalt auf:

```
> cvs checkout -r rel-01
> vi projekt01/main.c
> cvs commit
cvs commit: Examining .
cvs commit: sticky tag 'rel-01' for file ...
cvs [commit aborted]: correct above errors first!
```

Das Problem in diesem Fall liegt darin, dass CVS alle Dateirevisionen aus der Vergangenheit mit den sogenannten 'Sticky Tags' markiert und somit diese Dateien nicht mehr veränderbar macht. Die 'Stiky Tags' können nur entfernt werden, wenn wenn man die Arbeitskopie auf den aktuellen Stand bringt oder Verzweigungen benutzt, deren Beschreibung jedoch den Rahmen dieser Ausarbeitung sprengen würde.

5 Zusammenfassung

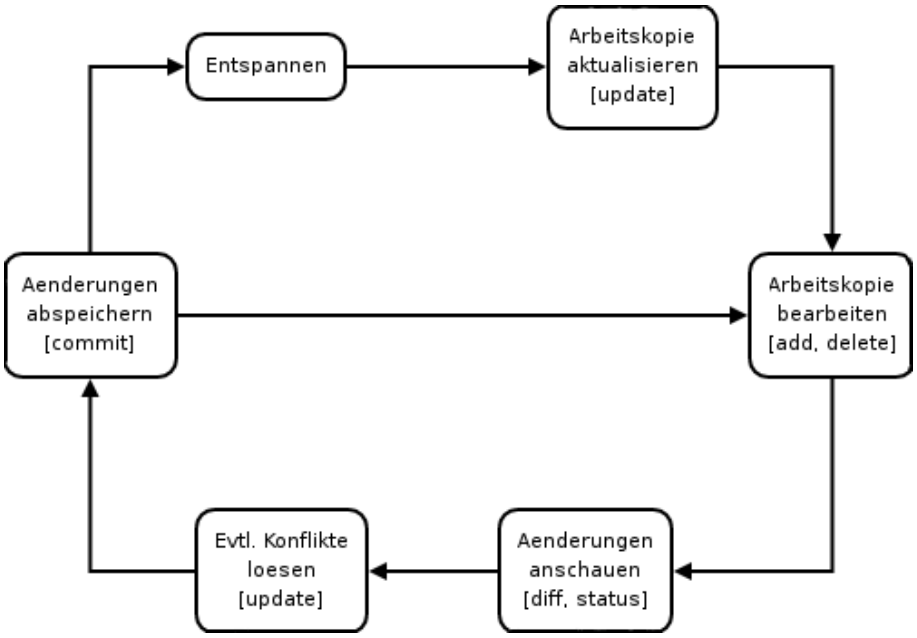
An dieser Stelle ist es nützlich einen zusammenfassenden Überblick über CVS zu geben:



CVS ist ein Versionsverwaltungssystem, mit dem grosse Software-Projekte verwaltet werden können. Organisiert ist es nach dem Client-Server-Konzept. Das bedeutet, dass alle Kopien der Projekte sich auf den Servern in den Repositorys befinden und die CVS Clients auf diese Resourcen zugreifen können. Übliche Vorgehensweise ist es, sich eine aktuelle Arbeitskopie mit dem Befehl `checkout` auf den lokalen Rechner zu holen, diese zu bearbeiten und anschliessend alle Änderungen mit `commit` zurück auf den Server zu übertragen. Darüber hinaus stehen noch solche Befehle wie `update`, `log`, `diff`, `status`, `tag` und viele andere dem Benutzer zur Verfügung.

5.1 CVS Befehlszyklus

Im alltäglichen Gebrauch kommt ein Benutzer nur mit wenigen Befehlen aus, deren Anwendung einem Zyklus unterliegt.



Das erste was ein Entwickler jeden Tag vor dem Bearbeiten des Projekts zu verrichten hat, ist die Aktualisierung der Arbeitskopie durch den Befehl `update`. Anschliessend wird das Projekt, das auf den neuesten Stand gabracht wurde, bearbeitet. Darunter versteht man die Veränderung der Dateien oder aber auch das Anwenden der Befehle `add`, `remove` usw. Die somit gemachten Änderungen können mit den Befehlen `diff` und `status` angeschaut werden. Falls andere Entwickler in der Zwischenzeit das Projekt auch verändert haben sollten, so muss in doesem Fall wieder der Befehl `update` ausgeführt werden. Abschliessend müssen die gemachten Änderungen an das Archiv mit dem Befehl `commit` übertragen werden. Nun kann an dieser Stelle der Befehlszyklus geschlossen werden, wenn der Benutzer sofort weiter das Projekt bearbeitet. Alternativ könnte der Benutzer die Arbeit an dem Projekt auch später fortsetzen. In diesem Fall muss dann aber wieder eine Aktualisierung der Arbeitskopie stattfinden. Erst danach kann das Projekt wieder bearbeitet werden.

5.2 Nachteile von CVS

Aber CVS hat natürlich auch mit einigen Schwächen zu kämpfen, die durch Weiterentwicklungen wie SVN zum grössten Teil behoben worden sind.

- Verschieben und Umbenennen von Verzeichnissen/Dateien ist umständlich und bringt unerwünschte Nebenwirkungen mit sich
- Keine Verwaltung von Verzeichnissen und Metadateien wie Dateiberechtigungen
- Umständlicher Umgang mit Binärdateien
- Commits sind nicht atomar und können somit zu inkonsistenten Zuständen führen
- Operationen wie 'update', 'tagging' und 'branching' sind recht langsam

- Windows und Linux Clients behindern sich gegenseitig

5.3 Quellen

- http://www.nongnu.org/cvs/ - Ofizielle Homepage
- http://ximbiot.com/ - Dokumentation
- http://cvsbook.red-bean.com/ - deutsche Dokumentation
- Man Pages