HTWK-Leipzig
Fachbereich Informatik, Mathematik und Naturwissenschaften

CVS, Subversion und Darcs

Autor : Roland Gion
rgion@imn.htwk-leipzig.de

vorgelegt am: 18. Mai 2005

Fach: Oberseminar Softwareentwicklung

Betreuer: Prof. Dr. Johannes Waldmann

Zusammenfassung

Kleine Änderungen am Programmcode zeigen häufig einige Zeit später unerwünschte Nebenwirkungen in anderen Programmteilen. Nun muss man herausfinden, was sich denn genau geändert hat, um den Fehler zu korrigieren. Die Speicherung aller Zwischenstände erzeugt bereits bei kleinen Projekten eine beachtliche Datenmenge, die dann durchforstet werden muss. Dies ist ein langwieriges und mühseliges Unterfangen, das nicht praktikabel ist. Arbeiten mehrere Entwickler gemeinsam an einem Projekt verkompliziert das die Suche nach Änderungen erheblich. Hier tritt noch das Problem auf, dass Entwickler sich gegenseitig Änderungen überschreiben könnten.

Um diese Probleme zu lösen, wurden die Versionskontrollsysteme entwickelt. Drei Vertreter, das *Concurent Versions System* (CVS), *Subversion* und *Darcs*, sollen hier näher betrachtet werden. Weitere bekannte Systeme sind *Arch*, mit der letzten Implementierung *TLA* – besser bekann als *gnu-arch* – von Tom Lord. Weitere Vertreter sind das *Revision Control System* (RCS), das in CVS weiter lebt, und das *Source Code Control System* (SCCS), das in fast jeder UNIX Version enthalten ist.

Inhaltsverzeichnis

Tabellenverzeichnis

1 CVS

Das *Concurent Versions System* wurde im Dezember 1986 von *Dik Grune* als eine Sammlung von Shell-Scripts in einer Newsgroup als Erweiterung des RCS (Revision Control System) veröffentlicht. Im April 1989 wurde CVS von *Brian Berliner* einem Redesign unterzogen. Bei der Codierung wurde er später von *Jeff Polk* unterstützt. CVS wird unter der GPL veröffentlicht. Eine ähnliches System ist *SCCS*, das sich allerdings nicht so weit verbreitet hat wie *CVS*.

CVS steht für die meisten UNIXe (AIX, HP, SGI, Solaris), Linux, MacOS X, und Win32 auf http://www.cvshome.org zur Verfügung. Als dieses Dokument enstand, war die aktuelle Version von CVS 1.11.20.

CVS speichert sämtliche Dateien und Verzeichnisse in einem Repository (siehe Abschnitt 1.2). *Module* können zum Zusammenfassen von Dateien und Verzeichnissen zu einer Entität benutzt werden. Im Normalfall entspricht ein Modul dem Projekt.

Jede Datei hat eine eindeutige *Revisionsnummer* der Form “1.1”, “1.2.3.4.5” usw. Die Revisionsnummer besteht aus einer Folge von positiven, ganzen Zahlen. Die Vergabe der Nummern erfolgt durch das System, wobei die erste Nummer immer ’1.1’ ist.

Der Revisionsbaum ist nicht zwingend linear. Er kann in *branches* (Äste) aufgesplittet werden, wobei der Revisionsnummer eine weitere Zahl angehängt wird. Die Änderungen der branches können einfach in den *main branch* (Stamm) übernommen werden.

Beim konkurrierenden Zugriff auf Dateien kommt es häufig vor, dass mehrere Entwickler die gleiche Datei editieren. Abhilfe schafft hier der sogenannte *reserved checkout*, bei dem die Datei nach dem Checkout gesperrt wird und so exklusiver Zugriff zugesichert werden kann. Diese Art des Checkouts ist bei CVS nicht sehr gut implementiert und wird auch selten verwendet. Man verlässt sich darauf, dass mehrfache Änderungen zusammengeführt werden können.

Abhängig von den Operationen, die auf einer Datei ausgeführt wurden, befindet sich diese Datei in einem der folgenden Zustände:

Up-to-date Die Datei ist identisch mit der letzten Revision dieser Datei in dem Repository.

Locally Modified Die Datei wurde editiert, aber die Änderungen dem Repository noch nicht bekannt gemacht.

Locally Added Die Datei wurde in die Versionsverwaltung aufgenommen, aber die Änderungen wurden noch nicht bekannt gemacht.

Locally Removed Die Datei wurde entfernt und die Änderungen wurden noch nicht bekannt gemacht.

Needs Checkout Eine neuere Revision dieser Datei befindet sich im Repository. Ein *update* ist nötig.

Needs Patch Diese Datei muss ausgecheckt werden. Der Server sendet nur einen patch statt der gesamten Datei.

Needs Merge Die Datei des Repository wurde seit dem letzten Update geändert, wie auch die lokale Kopie. Die Änderungen müssen zusammengeführt werden.

Unresolved Conflict Wie *Locally Modified*, allerdings kann die Zusammenführung nicht automatisch vorgenommen werden.

Unknown Der Zustand ist unbekannt - meist bei neuen Dateien, die noch nicht in die Versionsverwaltung aufgenommen wurden.

1.1 Eine Beispielsitzung

In dieser Arbeit wird nur auf den Umgang mit den CVS Kommandozeilen-Tools eingegangen. Es existiert eine Vielzahl von zum Teil graphischen Programmen, von denen die häufigsten in Abschnitt 1.6 beschrieben werden. Gehen man davon aus, dass bereits ein Modul *samplemodule* in dem Repository (siehe Abschnitt 1.2) vorhanden ist, dann kann man dieses auschecken, um damit zu arbeiten.

```
$ cvs checkout samplemodule
```

Mit diesem Kommando erreicht man, dass ein lokales Vezeichnis *samplemodule* erstellt wird und die Dateien aus dem Repository hineinkopiert werden. Nachdem die Änderungen vorgenommen wurden, kann man mit dem folgenden Kommando dem Server die Änderungen bekannt machen.

```
$ cvs commit -m "Beschreibung der Aenderungen" die_geaenderte_datei
```

Nachdem die Änderungen auf dem Server bekannt sind, möchte man vielleicht aufräumen und die lokale Kopie löschen. Dies erreicht man mit folgender Anweisung.

```
$ cvs release -d samplemodule
```

1.2 Repository

Das *Repository* (Depot/Lager) speichert die Dateien und Verzeichnisse, die sich unter der Versionsverwaltung befinden.

1.2.1 Anlegen

Bevor das CVS genutzt werden kann, muss ein Repository angelegt werden.

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

Durch dieses Kommando wird in dem angegebenen Verzeichnis (/usr/local/cvsroot) ein Verzeichnis *CVSROOT* angelegt, in dem die Systemdateien gespeichert werden. Das *cvs init* Kommando ist vorsichtig, d.h. es überschreibt das Repository nicht wenn dies schon vorhanden ist. Unterhalb des CVSROOT werden die Verzeichnisse für die Module angelegt.

1.2.2 Bekanntmachung

Es gibt verschiedene Möglichkeiten CVS mitzuteilen wo es das Repository finden kann. Eine Möglichkeit ist, die -d Option zu nutzen.

```
$ cvs -d /usr/local/cvsroot checkout samplemodule
```

Eine weitere Möglichkeit ist, die *$CVSROOT* Umgebungsvariable zu setzen.

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

Ein Repository kann sich auf einem entfernten Rechner befinden. In diesem Falle sieht die Pfadangabe etwas anders aus.

```
:methode:user@hostname:/pfad/zu/dem/repository
```

Methode bezeichnet die Zugriffsmethode. Die häufigste ist hier *ext* oder *pserver*, falls der Server eine eigene Authentifizierung benutzt. Normalerweise wird die Authentifizierung des Servers benutzt, d.h. der Zugriff läuft über rsh oder ssh.

1.2.3 Dateien

Die Dateien liegen in *history files* vor. Für jede Datei, die unter Versionsverwaltlung steht, liegt im Repository eine Datei mit gleichem Namen vor. Dem Namen wird lediglich ein "*,v*" angehängt. Diese Dateien enthalten ausreichende Informationen, um jede Version einer Datei erzeugen zu können. Diese Dateien werden nur mit Lese-Rechten erzeugt. Die Zugriffsrechte können also ausschließlich über die Verzeichnisse gesteuert werden.

1.3 Modul

Ein Modul entspricht in den meisten Fällen dem Projekt. Für jedes Modul wird im Repository ein Verzeichnis angelegt, in dem die Dateien und die Verzeichnisstruktur gespeichert werden. Mit folgendem Kommando wird ein Modul *samplemodule* im Repository angelegt und die Verzeichnisstruktur aus dem aktuellen Verzeichnis in diesem Modul erzeugt. *Sample* entspricht hier dem vendor- und *start* dem release-tag. Hinter der Option "-m" wird eine Nachricht angegeben, die in das history-file geschrieben wird.

```
$ cvs import -m "Created directory structure" samplemodule sample start
```

Beim Anlegen eines neuen Projektes empfiehlt es sich, die Verzeichnisstruktur anzulegen und diese dann, wie oben beschrieben, zu importieren.

1.4 Häufige Operationen

Im Folgenden werden die wichtigsten Kommandos beim Umgang mit CVS erläutert. Damit die Kommandos wissen mit welchem Repository sie arbeiten sollen, muss entweder die Systemvariable *CVSROOT* gesetzt sein, oder mit dem Parameter *-d REPOSITORY* der Pfad zu dem Repository angegeben werden.

In vielen Fällen kann bei den Anweisungen auf den Dateinamen verzichtet werden. In diesem Fall werden die Anweisungen auf alle Dateien des Moduls angewendet.

1.4.1 Lokale Kopie des Moduls erstellen

Ein Modul kann mit Hilfe des Kommandos *cvs checkout* vom Server geholt werden.

```
$ cvs checkout samplemodule
```

Durch diese Anweisung wird ein lokales Verzeichnis mit der Verzeichnisstruktur und den Dateien dieses Moduls erzeugt. In jedem Verzeichnis des Moduls befindet sich ein Verzeichnis *CVS*, in dem Systemdateien zu finden sind.

1.4.2 Datei hinzufügen

Eine vorhandene Datei wird mit Hilfe des *add* Kommandos der Versionskontrolle hinzugefügt.

```
$ cvs add -m "irgendeine Nachricht" dateiname
```

1.4.3 Datei entfernen

```
$ cvs remove dateiname
```

Diese Anweisung entfernt eine Datei aus der Versionsverwaltung.

1.4.4 Mit Repository abgleichen

Der Abgleich mit dem Repository erfolgt über das *update* Kommando. Ist eine neuere Revision einer Datei in dem Repository vorhanden, werden die Änderungen in die lokale Datei übernommen. Sind die Änderungen zu nahe an den lokalen Änderungen, kann das System diese nicht automatisch zusammenführen. Ein Konflikt tritt auf. Beide Versionen der Änderung befinden sich in der lokalen Datei und sind hervorgehoben. Diese Änderungen müssen nun manuell zusammengeführt weden.

1.4.5 Änderungen in das Repository übernehmen

Die Änderungen an einer Datei, wie auch die Aufnahme oder das Entfernen aus der Versionsverwaltung, werden erst in dem Repository wirksam, wenn das *commit* Kommando ausgeführt wurde.

```
$ cvs commit dateiname
```

1.4.6 Version erstellen

Eine Version, d.h. eine Ansammlung von Dateien mit einer jeweils bestimmten Revisionsnummer, wird über das *tag* Kommando erreicht.

```
$ cvs tag release-1-0 .
```

Nun kann diese Version immer mit folgendem Kommando ausgecheckt werden.

```
$ cvs checkout -r  release-1-0 samplemodule
```

1.4.7 Änderungshistorie ansehen

```
$ cvs log dateiname
```

Mit dieser Anweisung werden alle Kommentare, die beim Commiten dieser Datei eingetragen wurden, aufgelistet.

1.4.8 Änderungen finden

```
$ cvs diff datei
```

Mit dieser Anweisung werden zwei Revisionen einer Datei verglichen und die Unterschiede aufgelistet. Über einen Parameter kann man die Revisionsnummern angeben, die miteinander verglichen werden sollen.

1.5 Schwächen

CVS ist mittlerweile in die Jahre gekommen und im Laufe der Zeit haben sich die Anforderungen an das Versionsverwaltungssystem geändert. Projekte haben heute mehr Mitglieder als früher und immer häufiger müssen binäre Dateien, wie z.B. Bilder, verwaltet werden.

Die Schwächen traten also im Laufe der Zeit immer stärker hervor.

Skalierbarkeit - CVS ist nicht skalierbar, d.h. bei großen Projekten können sich die Operationen, wie update oder checkout, als Geduldsspiel erweisen. Vor allem bei langsamen Netzverbindungen, wie z.B. Modem, wirkt sich dies stark aus.

Zugriff - Der Zugriff ist auf *rsh* oder *ssh* beschränkt, was beim Einsatz von Firewalls zu Einschränkungen führen kann.

Binärdateien - Binärdateien werden in CVS nur stiefmütterlich behandelt. Die Änderungen werden vollständig gespeichert, also nicht nur die Differenz der Dateien, was große Mengen an Plattenplatz kosten kann.

Historie von Verzeichnissen - Verzeichnisse unterliegen nicht der Versionsverwaltung und bleiben, einmal angelegt, für immer bestehen. Bei der Reorganisation des Dateibaums muss direkt auf dem Repository gearbeitet werden, um die Log-Informationen nicht zu verlieren.

Keine Transaktionskontrolle - Beim Einspielen von Änderungen, die sich über mehrere Dateien erstrecken, kann das Repository in einen inkonsistenten Zustand gebracht werden, wenn die Kommunikation, durch bspw. einen Absturz des Clients oder Servers oder Wegfall der Netzwerkverbindung, gestört wurde.

1.6 Client-Tools

CVS ist für die gängigen Betriebssysteme verfügbar, somit stehen die Kommandozeilen-Tools ebenfalls zur Verfügung. Bei Windows-Systemen ist *WinCVS*, eine graphische Anwendung die CVS vollständig unterstützt, eine beliebte Alternative zu der Kommandozeile. Ebenfalls gerne verwendet wird *TortoiseCVS*, eine sehr kleine Applikation, die sich direkt in den Windows-Explorer einbinden lässt. Die Steuerung wird hier über Kontextmenüs vorgenommen. In vielen IDEs, wie z.B. in *Eclipse*, ist allerdings eine Unterstützung für CVS schon vorhanden.

2 Subversion

ColabNet ist der Hersteller von *SourceCast*, einer Software zur Unterstützung der Zusammenarbeit in Projekten. *SourceCast* nutzt intern CVS als Versionskontrollsystem, das auf Grund seiner Schwächen abgelöst werden sollte. Im Jahr 2000 begannen die Bemühungen, *Subversion* als Nachfolger von *CVS* zu erschaffen. Nach 14 Monaten Entwicklungszeit "verwaltete" sich Subversion ab dem 31. August 2001 selbst, d.h. die Subversion Quellen wurden nicht mehr mit CVS, sondern mit Subversion verwaltet.

Um die CVS-Nutzer zu gewinnen, wurde das "look-and-feel" ähnlich dem von CVS entworfen und dessen Schwachstellen (siehe 1.5) beseitigt. Das System wurde von Grund auf neu implementiert, um die Altlasten loszuwerden. Subversion unterliegt, anders als CVS, nicht der *GPL*, sondern der *Apache License*. Dies liegt daran, dass die *Apache Portable Runtime Library* (APR Library) verwendet wird. Dies ändert jedoch nichts an der Tatsache, dass es frei verfügbar ist. Subversion läuft unter Unix, Linux, Win32, BeOS, OS/2 und MacOS X. Als dieses Dokument entstand, war die letzte stabile Version 1.1.4. Das Projekt wird inzwischen von Tigris.org (http://subversion.tigris.org), einem Tochterunternehmen von CollabNet, betreut.

2.1 Repository

Eine der wichtigsten Neuerungen ist, dass das Repository nicht mehr in flatfiles, sondern in einer Datenbank gespeichert wird. Zum Einsatz kommt hier die *Berkeley-DB*. Diese Neuerung bringt einige Vorteile mit sich. Einer der wichtigsten ist die Transaktionssicherheit. Ein Commit über mehrere Dateien hinterlässt immer ein konsistentes Repository, auch wenn die Verbindung gestört wurde. Im Fehlerfall erhält der Anwender eine Meldung und kann so das Commit ein weiteres Mal anstoßen. Die unzulängliche Verarbeitung von Verzeichnissen gehört nun ebenfalls der Vergangenheit an. Subversion unterscheidet nicht mehr zwischen Verzeichnissen und einfachen Dateien. Verzeichnisse unterliegen nun auch der Versionsverwaltung.

Gewöhnungsbedürftig ist sicherlich das neue Revisionsnummernkonzept von Subversion. Es gibt nur eine Versionsnummer für das gesamte Repository, die bei jedem Commit inkrementiert wird. Wenn man also von der Revision einer Datei spricht, meint man damit die Version der Datei, wie sie in eben dieser Revision des Repository war. Verschiedene Revisionen einer Datei bedeuten also nicht zwangsläufig, dass sich der Inhalt dieser Datei geändert hat.

2.1.1 Das Repository anlegen

Das Anlegen des Repository erfolgt analog zu CVS.

```
$ svnadmin create /usr/local/svnroot
```

Die Berkeley-DB benutzt Locking-Funktionen des Dateisystems, deshalb sollte ein Repository nur auf einer lokalen Festplatte erzeugt werden.

2.1.2 Ein Projekt anlegen

Die Projekte werden, wie bei CVS auch, als Verzeichnis in dem Repository angelegt. Es emplfiehlt sich, im Projektverzeichnis drei weitere Verzeichnisse zu erstellen: Das Verzeichnis "trunk" enthält den Hauptast des Projektes, das Verzeichnis "branches" enthält eventuelle Entwicklungszweige und "tags" nimmt die Zwischenstände (Versionen) der Entwicklung auf.

```
$ cd zu_importierendes_verzeichniss
$ svn import . file://localhost/usr/local/svnroot/sample/trunk
-m "Created directory structure"
```

2.1.3 Sicherungen

Wie schon beschrieben sind commits atomar und hinterlassen somit immer ein konsistentes Repository. Sollte der Server beim Schreiben von Dateien in das Repository abstürzen, können die Replay-Logs dabei helfen, ein beschädigtes Repository zu reparieren. In diesen Logs zeichnet Subversion alle Änderungen der Datenbank auf. Somit genügt ein einfaches `svnadmin recover`, um die in diesen Logs enthaltenen Änderungen in das Repository zu übernehmen. Um nach einer beschädigten Festplatte das Repository rekonstruieren zu können, sollten regelmäig Hot-Backups, mit Hilfe von `svnadmin hotcopy`, erzeugt werden. Zu diesem Zweck kann der Server weiter laufen und es entstehen keine Downtimes.

2.2 Interaktion mit dem Server

Die Basis bilden, wie bei CVS auch, die Kommandozeilen-Tools. Diese sind ebenfalls für die meisten gängigen Betriebssysteme verfügbar. Wer lieber mit graphischen Benutzeroberflächen arbeitet, findet auch hier eine Vielfalt an Anwendungen. *TortoiseSVN* ist eine Explorer-Erweiterung (analog zu TortoiseCVS). *RapidSVN* ist nicht ganz so einfach zu bedienen, ist aber auch für diverse UNIXe verfügbar. Wer mit Eclipse arbeitet, kann das plugin *subeclipse* installieren, das sich nahtlos in die Entwicklungsumgebung einfügt. Im Kontexktmenü "Team" befinden sich, analog zu CVS, die Funktionen für die Interaktion mit dem Server.

Subversion ist, dank eines Binary-diff-Algorithmus, in der Lage die Änderungen an Binärdateien zu identifizieren und diese abzuspeichern, anstatt, wie CVS, jeweils eine neue Kopie der Datei zu erstellen. Auch beim kopieren von Dateien in ein anderes Verzeichnis, wie z.B. beim Erstellen eines neuen branches, wird nur eine "cheap copy" erstellt, d.h. die Verwaltungsdaten des Repository werden als kopiert markiert. Um einen neuen Branch zu erstellen, muss lediglich das zu verzweigende Verzeichnis aus dem Verzeichnis "trunk" in das Verzeichnis "branches" kopiert werden. Um diese Funktion auszuführen, steht das Kommando `svn copy` zur Verfügung. Auf diese Weise wird auch eine Version (Tag) erstellt. Die Kopie landet dabei in dem Verzeichis "tags". Mit folgender Anweisung wird eine Version "release-1.0" aus dem Hauptzweig erstellt.

```
$ svn copy http://svn.example.com/repos/calc/trunk \
  http://svn.example.com/repos/calc/tags/release-1.0 \
  -m "Tagging the 1.0 release of the 'calc' project."
```

Wird beim Checkout nicht die URL eines bestimmten Moduls, sondern die des Repository angegeben, wird eine lokale Kopie aller Module erstellt. Zusätzlich werden auch die .svn-Verzeichnisse, die administrativen Zwecken dienen, ausgecheckt. Diese Verzeichnisse enthalten die Dateien der Projekte seit dem letzten checkout oder commit. Dies erlaubt einen Offline-Betrieb, d.h. viele Operationen können ausschließlich mit Hilfe dieser Dateien ausgeführt werden. Dieses Vorgehen reduziert die Netzlast in Client/Server Umgebungen auf Kosten von Speicherkapazität auf dem Client. Heutzutage ist Plattenplatz allerdings günstiger als Bandbreite.

2.2.1 CVS und Subversion Kommandos

Wie bereits erwähnt, wurde beim Entwurf bereits großer Wert auf ein ähnliches "look-and-feel" zu CVS gelegt. In Tabelle 1 werden die wichtigsten Kommandos gegenübergestellt.

Aktion	**CVS**	**SVN**
Initialisierung	cvs init	svnadmin create directory
Checkout	cvs checkout module	avnadmin checkout method://rep/path
Dateien hinzufügen	cvs add file	svn add file
Dateien entfernen	cvs delete file	svn delete file
Commit	cvs commit file[s]	svn commit file[s]
Logs ansehen	cvs log file[s]	svn log file[s]

Tabelle 1: Vergleich der CVS und Subversion-Kommandos

2.3 Zugriffsmethoden

Eine wichtige Neuerung sind die Zugriffsmethoden. Die von CVS genutzten Protokolle *ssh* und *rsh* wurden um *http* erweitert. Zur Verfügung stehen also die Zugriffsmethoden in Tabelle 2.

Protokoll	**Beschreibung**
file://	Zugriff auf ein Repository in einem lokalen Filesystem
svn://	Zugriff über den standalone SVN-Server
svn+ssh://	Zugriff über den standalone SVN-Server durch einen SSH-Tunnel
http://	Zugriff über einen Apache2-Webserver und das WebDAV[1]-Protokoll
https://	Zugriff über einen Apache2-Webserver und das WebDAV-Protokoll und SSL-Verschlüsselung

Tabelle 2: Zugriffsmethoden auf einen Subversion-Server

[1]Das *Web-based Distributed Authoring and Versioning* (WebDAV)-Protokoll ist eine Erweiterung des http-Protokolls und entwickelt sich immer mehr zu einem Standard für Filesharing. http://www.webdav.org

3 Darcs

David's Advanced Revision Control System (Darcs), ein von David Roundy entwickeltes Programm, ist ein weiterer Vertreter der Revisionssysteme. Wie die vorher genannten, steht es ebenfalls zur freien Verfügung, da es unter der GPL lizensiert ist. Darcs wurde in Haskell entwickelt und steht so für viele Betriebssysteme, wie Linux, MacOS X, FreeBSD und Windows, zur Verfügung. Die letzte stabile Version beim Entstehen dieses Dokumentes war 1.0.2. Diese kann von der Seite http://www.darcs.net bezogen werden.

3.1 Repository

Eine der wichtigsten Unterschiede zwischen Darcs und CVS/Subversion ist die Tatsache, dass jede lokale Kopie des Repository ein vollständiges Repository darstellt. Jedes vollständige Auschecken erzeugt also ein neues Repository. Dies unterstützt natürlich die "mobile Entwicklung", denn das Repository ist auch unterwegs immer erreichbar. Ein weiterer Vorteil ist die extrem reduzierte Netzlast, bedingt durch den Abgleich mit dem lokalen Repository. Ein Nachteil ist das deutlich größere lokale Repository.

Darcs kümmert sich nicht um die Dateiversionen, sondern um die Projektänderungen, die sogenannten Patches. Das ist eine völlig andere Herangehensweise als bei den vorher beschriebenen Versionierungssystemen. Diese Patches werden zwischen den verschiedenen Repository ausgetauscht, um die Änderungen in diesen bekannt zu machen. Verschiedene Repository können sich diese Patches in Form von Dateiverweisen teilen, um Speicherplatz zu sparen. Dies setzt allerdings voraus, dass sich die Repsoitory auf dem gleichen Dateisystem befinden. Darcs unterstützt, ebenso wie Subversion, das Umbenennen und Verschieben von Dateien. Branches werden durch die Erzeugung eines neuen Repository erstellt.

3.1.1 Repository anlegen

Das Anlegen des Repository ist ähnlich trivial wie bei CVS/Subversion und wird durch folgende Anweisung erledigt.

```
$ cd projektverzeichnis
$ darcs initialize
```

Diese Anweisung legt das `_darcs` Verzeichnis an, in dem die Verwaltungsdateien gehalten werden.

3.1.2 Projekt anlegen

Die Dateien werden mit dem Kommando *darcs add* der Versionsverwaltung hinzugefügt.

```
$ darcs add datei[en]
```

Nach dem Hinzufügen können die Änderungen persistent gemacht werden.

```
$ darcs record --all
```

3.1.3 Zugriffsmöglichkeiten

Darcs benutzt gängige Protokolle, wie http, ftp und ssh, um auf das Repository zuzugreifen. Für ein schnelles browsen des Repository per http stehen cgi-scripts zur Verfügung, die in den Webserver eingebunden werden können.

Darcs benutzt keine eigene Rechteverwaltung. Die Verwaltung dieser kann ausschließlich über die Zugriffsrechte des Dateisystems oder des Webservers gesteuert werden.

3.2 Häufige Operationen

Im Folgenden werden die wichtigsten Kommandos für den Umgang mit Darcs erläutert. Im Unterschied zu den beiden vorangegangenen Versionsverwaltungen gibt es bei Darcs Kommandos, um mit dem lokalen Repository und mit einem entfernten Repository arbeiten zu können.

3.2.1 Die Änderungen eines entfernten Repository einlesen

Um die Änderungen eines entfernten Repository in das eigene zu übernehmen, muss man lediglich in das eigene Repository-Verzeichnis gehen und folgendes Kommando absetzen.

```
$ darcs pull http://your.server.org/repos/yourproject
```

Sollte noch keine Kopie des entfernten Repository angelegt worden sein, kann das mit folgendem Kommando erzeugt werden.

```
$ darcs get http://your.server.org/repos/yourproject
```

3.2.2 Datei hinzufügen

Eine vorhandene Datei wird mit Hilfe des *add* Kommandos der Versionskontrolle hinzugefügt.

```
$ darcs add dateiname
```

3.2.3 Datei entfernen

```
$ darsc remove dateiname
```

Diese Anweisung entfernt eine Datei aus der Versionsverwaltung.

3.2.4 Änderungen in das Repository übernehmen

Änderungen werden mit dem Kommando `darcs record` in das lokale Repository übernommen. Sollen diese Änderungen in ein fremdes Repository übernommen werden, kann das Kommando `darcs push` benutzt werden. Das `record`-Kommando ist durch `unrecord` umkehrbar.

3.2.5 Änderungen finden

Um Änderungen zu finden, müssen ebenfalls unterschiedliche Kommandos benutzt werden, abhängig davon, ob mit dem lokalen oder einem entfernten Repository verglichen wird. `Darcs diff` wird benutzt, um mit dem lokalen Repository zu vergleichen. Analog dazu wird `darcs whatsnew` für den Vergleich mit einem entfernten Repository benutzt.

3.2.6 CVS und Darcs Kommandos

Im folgenden werden die wichtigsten CVS- und Darcs-Kommandos gegenübergestellt.

Aktion	**CVS**	**SVN**
Initialisierung	cvs init	darcs initialize
Checkout	cvs checkout	darcs get
Update	cvs update	darcs whatsnew — darcs pull
Dateien hinzufügen	cvs add	darcs add
Dateien entfernen	cvs delete	darcs delete
Commit	cvs commit	darcs record — darcs push
Änderungen bestimmen	cvs diff	darcs diff — darcs whatsnew
Branch erzeugen	cvs tag -b	darcs get
Version erzeugen	cvs tag	darcs tag

Tabelle 3: Vergleich der CVS und Darcs-Kommandos

4 Fazit

CVS ist vermutlich, trotz seiner Schwächen, immer noch das am häufigsten benutzte Revisionssystem. Die Funktionalität entspricht heutzutage nicht mehr den Anforderungen, die an ein Revisionssystem gestellt werden. Somit verliert CVS immer mehr Anteile. *Subversion* hat, dank des an CVS angelehnten "look-and-feels", das Potenzial dieses in nicht allzu ferner Zukunft abzulösen. Viele große Open Source Projekte, wie Apache, setzen bereits auf Subversion. Subversion ist bereits in vielen Linux Distributionen enthalten und steht ebenfalls für die Windows-Welt als Binary zur Verfügung, was seine Verbreitung erleichtern wird. Für viele CVS-Clients steht ein Subversion-Pendant zur Verfügung, so z.B. TortoiseSVN oder das Eclipse Plugin. Darcs – in dieser Gruppe sicher ein Außenseiter – hat durch die andere Herangehensweise an die Repository-Verwaltung einen Vorteil bei der mobilen Entwicklung, die sich immer mehr verbreitet. Das "look-and-feel" ist ebenfalls ähnlich dem von CVS, allerdings sind keine graphischen Tools verfügbar. Die Frage ist, ob der Vorteil des vollständigen lokalen Repository gegenüber der besseren Repository-Verwaltung (Datenbank, Recovery, Hot-Backup, etc.) von Subversion überwiegt.

Literatur

[CSFP05] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. TBA, 2005. http://subversion.tigris.org.

[P+05] Derek R. Price et al. *Version Management with CVS*. Free Software Foundation, 2005. http://www.cvshome.org.

[Rou] David Roundy. *Darcs Manual*. http://www.darcs.net.