# Versionsverwaltung mit Subversion

Christian Barthel
Mail: bc@user-mode.org
Web: bc.user-mode.org

***Zusammenfassung*—Die Versionsverwaltung (version control system) mit Subversion beschäftigt sich mit der Erfassung von Änderungen an Dateien. Alle Veränderungen einer Datei werden festgehalten und können jederzeit über das Archiv eingesehen beziehungsweise sogar wiederhergestellt werden. Das wohl am häufigsten vorkommende Einsatzszenario der Versionsverwaltung liegt in der Softwareentwicklung: der Quellcode der Programme, der sich im Laufe der Zeit stark ändern kann, wird revisionssicher gespeichert und kann jederzeit wieder rückverfolgt werden. VCS, also die Kurzform für version control system, kann aber auch in Büroanwendungen, der IT-Administration oder bei Content Managementsystemen eingesetzt werden.**

***Index Terms*—Subversion, Versionsverwaltung, Git, Programmierung, Content Management, Revisionen, Versionen**

## I. VCS im Allgemeinen

Die revisionssichere Speicherung von Dokumenten wird mit Hilfe einer VCS-Software ermöglicht. Die Hauptaufgaben der Versionsverwaltung ist die

- Archivierung der unterschiedlichen Fassungen eines Dokuments,
- Protokollierung und Rückverfolgung von Veränderungen mit optionaler Zeit- und Benutzerangaben,
- Wiederherstellung alter Versionsstände,

Ferner regeln Versionsverwaltungssysteme auch den Vielfachzugriff auf einzelne Dokumente oder gesamte Projekte.

### *A. Therminologie*

In der Versionsverwaltung werden häufig Programmierprojekte verwaltet. Diese Programmierprojekte bestehen aus einer Vielzahl unterschiedlicher Dateien. In objektorientierten Entwicklungssprachen können dies beispielsweise unterschiedliche Klasse, Header-Dateien, GUI-Form-Dateien oder andere Dinge sein. Im folgenden wird die Therminologie des Versionsmanagements anhand eines Softwareentwicklungsprojekts beispielhaft erklärt.

#### A.1 Branch

Ein **Branch** ist eine Abspaltung (Zweig) der Hauptentwicklungslinie. Zweige ermöglichen eine parallele, aber unterschiedliche Entwicklung des Projekts. Stellen Sie sich vor Sie wollen probeweise ein neues Softwaremodul für Ihre Anwendung entwickeln, die aktuelle Hauptentwicklung soll aber dadurch nicht gefährdet werden. Ein Branch ermöglicht es, dass die Entwickler die normale Hauptentwicklungslinie folgen, parallel aber auch an einem neuen Modul programmieren können. Beide Entwicklungslinien können störungsfrei verwendet werden.

#### A.2 Merging

Es ist auch möglich - was grundsätzlich auch nur logisch ist - Änderungen von unterschiedlichen Entwicklungszweigen wieder zusammenzufühgen. Hierbei spricht man vom **Merging**, das Verschmelzen. Es ist also möglich, dass beisppielsweise die Entwicklung des neuen Moduls irgendwann - wenn es soweit ist - wieder in den Hauptentwicklungszweig zurückfließt und somit zum Main-Stram gehört.

#### A.3 Trunk

Der Hauptentwicklungszweig wird auch häufig als **Trunk** bezeichnet. Dieser Zweig stellt die Hauptentwicklungslinie dar, die meist sehr stabil und mit wenigen Experimenten versehen ist.

#### A.4 Fork

Ein **Fork** bezeichnet die Aufteilung eines Softwareprojekts. Teile der Historie sind gleich und werden ab dem Aufteilungsdatum getrennt entwickelt. Forks sind in der Regel Branches.

#### A.5 Tags

**Tags** sind feste, konkrete Versionsstände, die einer besonderen Bedeutung zukommen. Beispielsweise könnte ein Tag eine Veröffentlichung eines Softwarereleases darstellen oder eine Ausflieferung an einen bestimmten Kunden. Tags haben meist einen Namen oder eine feste Versionsnummer, über die später noch der jeweilige Entwicklungsstand eindeutig identifiziert werden kann.

#### A.6 Repository

Ein **Repository** ist der Aufbewahrungsort aller Versionsstände, des Hauptentwicklungszweiges und der Tags. Gegebenenfalls können auch Forks innerhalb des gleichen Repositories aufbewahrt werden.

### *B. Konzept: Versionsverwaltung*

Kein Entwickler arbeitet mit den Dateien im Repository: Würden alle Programmierer auf den gleichen Dateien Änderungen durchführen, dann käme es sehr schnell zu Konflikten und eventuell sogar zu Datenverlusten. Bevor ein Entwickler seine Arbeit beginnen kann, muss er den aktuellen Stand des Projektes aus dem Repository laden. Er kopiert sich daher die Dateien, die er braucht, und legt diese in seine lokale Arbeitskopie. Mit dieser Arbeitskopie arbeitet nur ein Entwickler. Der Vorgang des kopierens der Repository-Daten in die lokale Arbeitskopie wird auch als **Auschecken** (Checkout) bezeichnet. Die Änderungen,

die der Entwickler an seinen Dateien durchführt, werden nicht sofort in das Repository übertragen. Der Entwickler bestimmt eigenständig, wann er seine modifizierte lokale Arbeitskopie wieder in das Repository übertragen möchte und seine Änderungen somit öffentlich zugänglich macht. Dieser Vorgang - also das Veröffentlichen der Änderungen - wird als **Commit, Einchecken oder Check-in** bezeichnet.

Für die Aufgaben des Auscheckens und Eincheckens gibt es meist kommandozeilenbasierte Programme, grafische Benutzeroberflächen oder direkte Plugins für unterschiedliche Entwicklungsumgebungen.

Prinzipiell unterscheidet man 3 unterschiedliche Arten der Versionsverwaltung:

### B.1 Lokale Versionsverwaltung

Die älteste Versioniserungsmöglichkeit ist die lokale Versionsverwaltung. Im Gegensatz zu den anderen Versioniserungskonzepten kann die lokale Verionsverwaltung nicht über Netzwerkebenen arbeiten und beschränkt sich häufig auf eine einzige Datei. Bekannte Produkte sind SCCS, Source Code Control System sowie RCS, das Revision Control System. SCCS wurde 1972 von Marc J. Rochkind an den Bell Labs entwickelt und wird als die erste Versionsverwaltungssoftware angesehen. Üblicherweise wird SCCS bei kommerziellen Unix-Versionen mitgeliefert. Das Revision Control System wurde 1980 entwickelt funktioniert dabei ähnlich wie SCCS und beschränkt sich auf die Verwaltung einzelner Dateien, nicht aber auf ganze Projekte. RCS wird heute beispielsweise noch von Twiki, einem frei erhältlichem Wiki-System eingesetzt.

### B.2 Zentrale Versionsverwaltung

Die zentrale Versionsverwaltung ist nach dem Client/Server-Prinzip aufgebaut. Das Repository, indem auch die komplette Versionsgeschichte gespeichert ist, kann zentral über das Netzwerk zur Verfügung gestellt werden. Das erste und ehemals sehr weit verbreitete Produkt war CVS (Concurrent Versions System). CVS wurde 1989 als netzwerkfähiger Aufsatz für RCS entwickelt. Der Nachfolger Subversion wurde im Jahr 2000 entwickelt und ist neben Git vermutlich das interessanteste Versionsverwaltungssystem für die Softwareentwicklung.

### B.3 Verteilte (distributed) Versionsverwaltung

Im Vergleich zur zentralen Versionsverwaltung verwendet das verteilte Versionsverwaltungssystem kein zentrales Repository. Jeder Projektbeteiligte besitzt sein eigenes Repository und kann dieses mit jedem beliebigen Repository abgleichen. Würde bei einer zentralen Versionsverwaltung mehrere Benutzer die gleichen Datei manipulieren, so wären Konfolikte vorhersehrbar. Bei der verteilten Revisionierung können Änderungen parallel durchgeführt werden und zu einem späteren Zeitpunkt zu einer neuen Version zusammengeführt werden. Dadurch entsteht keine verkettete Versionsgeschichte sondern eine Polyhierachie. Meist entwickeln Gruppen bestimmte Module oder Funktionen, die dann von einer Person mit einer entsprechenden Berechtigungsstufe überprüft und zusammengeführt werden. Wichtigster Vertreter dürfte hier Git sein, das von Linus Torvalds für die Linux Kernelentwicklung verwendet wird.

### B.4 Übersicht

- Lokale Versionsverwaltung
  - RCS (Open Source)
  - SCCS
- Zentrale Systeme
  - CVS (Open Source)
  - Subversion (Open Source)
  - Alienbrain
  - Team Foundation Server
  - Visual SourceSafe
- Verteilte Systeme
  - Bazaar (Open Source)
  - Darcs (Open Source)
  - Git (Open Source)
  - GNU arch (Open Source) item BitKeeper

## *C. Modify-Mechanismen*

Im Laufe der Zeit haben sich 2 unterschiedliche Varianten entwickelt, um den Mehrfachzugriff auf Dateien sicherzustellen:

### C.1 Lock - Modify - Write

Einzelne Dateien werden vor einer Änderung gesperrt und können erst dann verwendet werden, wenn die Datei wieder freigegeben wurde. Bei dieser Art des Zugriffs wird keine Zusammenführung benötigt, da zu jeder Zeit die Datei nur durch einen Benutzer bearbeitet werden kann.

### C.2 Copy - Modify - Merge

Der wohl häufiger zum Einsatz kommende Ansatz ermöglicht mehreren Benutzern die gleiche Datei zu ändern. Über das Merging (zusammenführen) werden die unterschiedlichen Änderungen in einer Datei gespeichert. Dabei kann es auch zu Konflikten kommen die nur durch einen menschlichen Eingriff gelöst werden können.

# II. Subversion

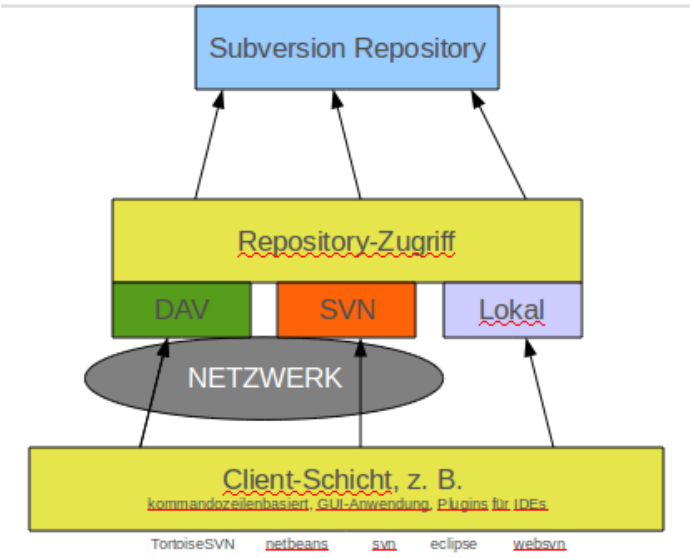
Im folgenden geht es um Subversion, einem zentralen Versionsverwaltungssystem das im Jahr 2000 von CollabNet entwickelt wurde und derzeit als Apache Projekt frei erhältlich ist. Subversion findet eine sehr weite Verbreitung, beispielhafte Projekte sind Free Pascal, FreeBSD, GCC, Django, Ruby, MediaWiki, Google, Tigris.org, SourceForge, etc. Aber nicht nur in der Open source Gemeinschaft ist Subversion weit verbreitet. Auch im kommerziellen Bereich ist Subversion die führende Software bezüglich des Versions- und Changemanagements.


- Erste Veröffentlichung: 20.10.2000

- Implementierungssprache: C
- Betriebssystem: cross-platform
- Lizenz: Apache License
- Website: subversion.apache.org

### A. Subversion Architektur

Die Architektur von Subversion ist in mehrere Schichten aufgeteilt. Daher können Sie unterschiedliche Protokolle für den Zugriff auf das Repository benutzen, Sie können ebenfalls unterschiedliche Speichertechniken im Repository nutzen und natürlich können Sie auch verschiedene Subversion-Client-Programme verwenden.




### B. Subversion installieren

Um Subversion erfolgreich zu installieren haben Sie unterschiedliche Möglichkeiten:

1. Download des Original-Quellcodes und Kompilierung der Dateien
2. Installation über das Paketsystem unter Linux (apt-get, yum)
3. Installation über vorkompilierte Binärdateien unter Windows

Wichtig ist, dass die Apache Portable Runtime Bibliothek auf Ihrem System vorhanden ist. Mit Hilfe der APR wird die Plattformunabhängigkeit von Subversion gewährleistet und es ist daher möglich Subversion unter Linux, Windows oder anderen, APR kompatiblen Betriebssystemen, zu verwenden. Die Installation von Subversion kann beispielsweise unter Debian GNU/Linux folgendermaßen angestoßen werden:

```
apt-get install subversion
```

Sofern Sie Ihr Repository auch über das HTTP/HTTPS Protokoll erreichen möchten benötigen Sie noch den Apache Webserver und das SVN-Plugin für Apache. Die untere Befehlszeile installiert Subversion, Apache, das benötigte SVN-Plugin und zusätzliche Abhängigkeiten, die für die erfolgreiche Installation notwendig sind:

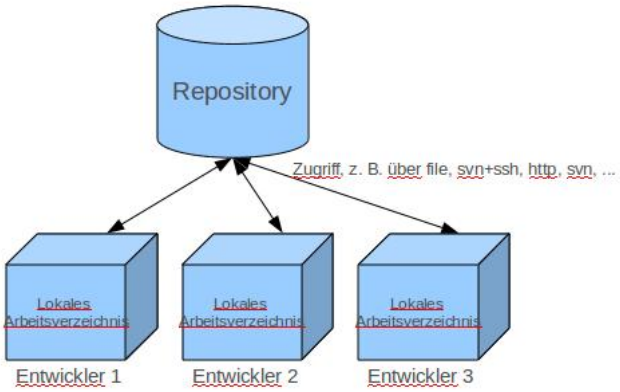
```
apt-get install subversion libapache2-svn
    apache2
```

### C. SVN Komponenten

Sobald die Installation erfolgreich abgeschlossen ist wurden einige zusätzliche Programme auf Ihrem Systyem installiert.

*svn* ist das kommandozeilenbasierte Client-Programm,
*svnadmin* ist ein Verwaltungstool mit dem Sie zum Beispiel Repositories anlegen können,
*svnserve* ist das von Subversion mitgelieferte Server-Programm.

Es gibt noch weitere Programme, wie zum Beispiel svnlook, svndumpfilter oder svndiff auf die später noch eingegangen wird. Subversion basiert auf der Client/Server Architektur. Daher liegt das zentrale Repository am Server und die Anwender besitzen jeweis lokale Kopien des Repositories. Die Änderungen werden immer in der lokalen Kopie durchgeführt.




### D. Subversion Repository-Zugriffsmethoden

Wie bereits vorhin erwähnt ist Subversion eine sehr modulare Software. Dies gilt auch für den Zugriff auf Repositorys: Sie können das Repository über unterschiedliche Verfahren erreichen:

- SVN-Protokoll, meist in einem lokalen Netzwerk
- SSH+SVN (SVN-Protokoll wird über SSH getunnelt/-abgesichert), stellt einen gesicherten Zugriff über ein unsicheres Netzwerk auf das Repository dar,
- HTTP/HTTPS (Zugriff z. B. über das Internet auf Repositorys), Verfahren, um Repository-Benutzer über das Internet zu verbinden (mit SSH+SVN meist ebenfalls möglich)
- file (lokaler Zugriff), meist nur bei einem Entwickler, der keine Netzwerkfunktionalität benötigt

Mit Hilfe des svn-Kommandozeilenprogrammes können Sie jede beliebige Repository-URL ansprechen:

```
svn://path/to/repos
ssh+svn://path/to/repos
http://path/to/repos
https://path/to/repos
file:///path/to/repos
```

*E. Revisionen*

Nach einem Inital-Import beginnt der Lebenslauf eines versionsgesteuerten Projekts. Dabei beginnt das Projekt bei der Revision 0. Jeder weitere erfolgreiche `Commit`-Befehl inkrementiert diese Revision. Jedes Commit wird als atomare Transaktion angesehen und ändert - bei erfolgreicher Durchführung - die Revisionsnummer des gesamten Projekts. Dies ist auch dann der Fall, wenn sich beisipelsweise nur eine Datei geändert hat. Im übrigen ist es auch möglich, lokal gemischte Revisionen zu haben und mit diesen zu arbeiten. Wenn Sie beispielsweise Wissen, dass Ihr Entwickler-Kollege drastische Änderungen an der Benutzerschnittstelle vornehmen wird, diese Sie aber nicht interessieren, dann können Sie beispielsweise einfach ungestört mit einer älteren Revision arbeiten. Gemischte Revisionen weisen allerdings Einschränkungen beim Löschen von Dateien/Verzeichnissen auf.

*F. Arbeitskopien*

Die Arbeitskopien unterscheiden sich von normalen, nicht revisionierten Ordnern oder Dateien dadurch, dass es ein Verzeichnis `.svn` gibt. Dieses unter Linux versteckte Verzeichnis übernimmt wichtige Verwaltungsaufgaben der lokalen Arbeitskopie. Es speichert beispielsweise die aktuelle Arbeitsrevision oder den Zeitstempel der letzten Aktualisierung mit dem Repository. Dadurch kann der svn-Client selbst entscheiden, in welchem Zustand sich eine Datei befindet:

- unverändert und aktuell, da lokal und im Repository keine Änderung an der Datei vorgenommen wurde.
- lokal geändert und aktuell, wenn die Datei im Repository seit dem letzten Commit nicht verändert wurde, aber lokal geändert wurde.
- unverändert und unaktuell bezeichnet den Zustand, wenn eine Datei im Repository verändert wurde, aber lokal noch auf einem alten Versionsstand ist.
- lokal geändert und unaktuell, wenn die Datei lokal und im Repository seit dem letzten checkout verändert wurde.

## III. SVN in der Praxis

Im folgenden werden die wichtigsten Befehle kurz besprochen.

*A. Repository erstellen: svnadmin create*

Zu Beginn einer Versionsgeschichte muss zunächst ein Repository erstellt werden. Dies wird meist von einem Administrator getätigt, der die Zugriffsrechte auf das `svnadmin`-Programm hat.

```
svnadmin create /pfad/zu/repository
```

Dieser Befehl erstellt ein leeres Subversion-Repository unter dem angegebenen Pfad, der bereits existieren sollte.

*B. Aufbau und Initial Import: svn import*

Das leere Subversion-Verzeichnis muss nun im nächsten Schritt befüllt werden. Beim ersten Importvorgang sollten Sie eine Verzeichnisstruktur anlegen, die aus den Ordnern branches, tags und trunk besteht. Der Aufbau kann so aussehen, wobei sich unterhalb des Ordners trunk bereits die Programmdateien aus der Hauptentwicklungslinie befinden können:

```
/tmp/project/branches
/tmp/project/tags
/tmp/project/trunk
                    -> foo.c
                    -> bar.c
                    -> Makefile
                    -> ....
```

Äquivalent zur Versionsverwaltungstherminologie ist der Ordner brances für Verzeigungen und tags für besondere Versionen gedacht. Trunk bezeichnet den Hauptentwicklungsast und stellt meist den Ordner dar, indem alles beginnt. Der Importvorgang wird wieder mit Hilfe des `svn`-Tools angestoßen:

```
svn import /tmp/project file:///pfad/zu/
    repos -m "initial import"
```

Der Befehl importiert alle Verzeichnisse und Dateien des Ordners `/tmp/project` in das Subversion-Repository. Um später einen besseren Überblick der Im- und Exportvorgänge zu bekommen, sollte immer eine Nachricht mit einer Erklärung der Änderungen mitangegeben werden. Beachten Sie, dass es sich bei `file:///` um ein lokales Repository handelt. Sofern das Repository über andere Protokolle erreichbar ist (siehe: D). Als Ausgabe erhalten Sie alle zum Repository hinzugefügten Dateien und Ordner:

```
...
Adding         /tmp/project/branches
Adding         /tmp/project/tags
...
```

**Achtung:** Arbeiten Sie nicht mit den importierten Dateien weiter! Sie können die Originaldateien, die Sie soben importiert haben, löschen oder umbennenen. Es ist aber nicht möglich, die vorhanden Dateien mit dem Repository abzugleichen.

*C. Repository-Dateien ansehen: ls*

Sie können - wie in der Linux-Shell ebenfalls möglich - Repository-Dateien mit dem Befehl `ls` auflisten. Sie müssen lediglich `svn ls` und eine Repository-URL aufrufen:

```
svn ls svn+ssh://server/path/to/repo
branches/
tags/
trunk/
```

*D. Arbeiten mit dem Repository: checkout*

Um eine lokale Arbeitskopie der Repository-Daten anzulegen, müssen Sie zunächst die Projektdateien aus dem Repository auschecken:

```
svn checkout file:///pfad/zu/repos/trunk
    project
```

Dieser Befehl holt alle Dateien, die im Ordner `trunk` gespeichert sind und legt eine lokale Arbeitskopie im Ordner `project` ab. Sobald nun ein Projekt ausgecheckt wurde können Sie Ihre Arbeit an den Dateien durchführen. Ein Checkout über das HTTP-Protokoll könnte in etwa so aussehen:

```
svn checkout http://svn.example.com/repos/
    calc
```

Sofern kein Ordnername als letztes Argument übergeben wird, dann legt der Subversion-Client den letzten Ordnername in der Verbindungsurl, also `calc` an und speichert alle Projektdateien darunter. Sofern Sie einmal eine ältere Revision aus dem Repository auschecken möchten können Sie dies über den `--revision`-Schalter erledigen:

```
svn checkout --revision 1729
```

*E. Nach der Arbeit: commit*

Sobald Sie Ihre Änderungen durchgeführt haben können Sie die Datei wieder in das Repository verschieben. Nach einem erfolgreichen `commit` Vorgang können auch Ihre Mitarbeiter Ihre Änderungen einsehen. Ein Aufruf von

```
svn commit
```

überführt alle Änderungen in das Repository. Sofern Sie nur eine bestimmte Datei im Repository veröffentlichen wollen können Sie optional den Dateinamen angeben:

```
svn commit trunk/button.c
```

Sofern ein Commit-Vorgang erfolgreich abläuft gibt Ihnen das SVN Client-Programm die Änderungen aus und nennt Ihnen die neue Revision:

```
Adding          trunk/test.txt
Transmitting file data .
Committed revision 4.
```

Ein Commit-Befehl kann ebenfalls zu Konflikten führen.

E.1 Lokale Kopie veraltet: svn update

Zum Beispiel ist es denkbar - und bei mehreren Projektteilnehmern auch sehr wahrscheinlich -, dass während Ihrer Arbeitszeit andere Entwickler Ihre Änderungen eingebucht haben. Somit haben sich auch die Dateien im Repository weiterentwickelt und Sie arbeiten lokal mit einem veralteten Stand.

**Es ist nicht möglich, einen Commit-Befehl mit einer nicht aktuellen lokalen Revision durchzuführen!**

Sie werden also gezwungenermaßen Ihre lokale Kopie auf den aktuellen Stand bringen müssen. Dies geschieht mit dem Befehl Update:

```
svn update [optionaler Dateiname]
```

Nachdem dieser Befehl ausgeführt wird, kümmert sich SVN um die Aktualisierung Ihrer lokalen Kopie und holt alle veränderten Dateien aus dem Repository. Mit `svn update` können Sie im übrigen auch zu einer älteren Revision zurückspringen:

```
svn update --revision 3 foo.c
```

E.2 Komplikationen beim Zusammenführen

Subversion verfügt intern über Algorithmen, die ein automatisiertes Zusammenführen, auch merging genannt, erlauben. Sobald aber eine gewisse Code-Passage im Quellcode von mehreren Entwicklern verändert wurde, kann der merging-Algorithmus die Änderungen nicht mehr zusammenführen und Sie müssen diesen Konflikt manuell beheben. Konflikte können Sie bereits durch `svn status -u` vorhersehen, da diese mit einem C gekennzeichnet werden. Um den Konfliktbereiche zu kennzeichnen schreibt Subversion sogenannte Konfliktmarker in die Datei und legt ausserdem zusätzliche Dateien an: eine Datei stellt den alten Zustand beim Auscheck-Vorgang dar (**dateiname.rOLDREV**), eine andere Datei stellt die vom Server empfangene Datei dar (**dateiname.r.NEWREV**). **dateiname.mine** ist die veränderte Datei in ihrer lokalen Arbeitskopie, bevor Sie mit den Konfliktmarkern versehen wurde und **dateiname** enthält Konfliktmarker von Subversion. Den Konflikt müssen Sie beheben:

1. Manulles editieren der mit Konfliktmarker behfateten Datei,
2. Sie kopieren eine der temporären Dateien an die Stelle Ihrer Arbeitsdatei
3. oder Sie verwerfen Ihre Änderungen mit `svn revert`

Danach müssen Sie Subversion mitteilen, dass der Konflikt behoben ist:

```
svn resolved
```

Die Konfliktlösung mit Hilfe des Kopriens könnte in etwa so aussehen:

```
$ svn update
C testdat.txt
Updated to revision 2.
$ ls testdat.*
testdat.txt testdat.txt.mine testdat.txt.
    r2
$ cp testdat.txt.r2 testdat.txt
$ svn resolved testdat.txt
```

*F. Dateioperationen in der working-copy: add, delete, copy, move*

Alle Dateioperationen wie z. B. das Verschieben, Hinzufügen oder Löschen von Dateien sollte mit Hilfe des svn-Befehls erfolgen.

```
svn add dateiname
```

Fügt die Datei `dateiname` zum Repository hinzu. Beim nächsten commit wird die Datei ebenfalls im Repository erreichbar sein.

```
svn delete dateiname
```

Merkt eine Datei zum Löschen vor: Der nächste commit löscht die Datei ebenfalls im Repository. Natürlich können Sie die Datei jederzeit auch über eine ältere Revision wiederherstellen.

```
svn copy dateiname
```

Kopiert eine Datei. Die Änderung wird ebenfalls erst nach einem commit im Repository bekannt.

```
svn move dateiname
```

Eine Mischung aus `svn copy` und `svn delete` ist `svn move`. Dieser Befehl sollte verwendet werden, wenn Sie eine Datei umbenennen möchten.

*G. Änderungen zurücksetzen: svn revert*

Das schöne an einem Versionsverwaltungssystem ist es, dass Datein wieder auf einen älteren - vielleicht funktionstüchtigeren - Stand zurückgesetzt werden können.

```
svn revert datei
```

Und natürlich können Sie auch gelöschte Dateien wieder mit revert wiederherstellen.

*H. Umgang mit der Versionierung*

Im zentralen Repository liegen die versionierten Projektdaten. Die Revisionsnummer wird nach jedem Commit inkrementiert, also um eins erhöht. Es gibt eine Vielzahl unterschiedlicher Befehle, die auch Operationen auf älteren Revisionen durchführen können. Um eine bestimmte Revision zu kennzeichnen gibt es den Optionsschalter `-revision`:

```
svn BEFEHL --revision REV1[:REV2]
```

Anstelle einer Revisionsnummer REV1 kann auch ein Schlüsselwort oder ein bestimmtes Datum genannt werden.

H.1 Revisionsschlüsselwörter

Um sich das Leben etwas zu vereinfachen wurden für wichtige Revisionsstände feste Revisionsschlüsselwörter eingeführt. Folgende Wörter bezeichnen eine bestimmte Revision:

| | |
|---|---|
| HEAD | Neueste Revision im Repository |
| BASE | Rev in der ausgecheckten Arbeitscopy |
| COMMITTED | l. Rev in der sich etwas änderte |
| PREV | Rev direkt vor der letzten Rev |

H.2 Differenzen anzeigen: diff

`svn diff` zeigt die Veränderungen zweier Revisionen auf.

```
svn diff hello.c
```

Dieser Befehl zeigt die Änderungen an der Datei `hello.c`.

```
===================================================================
--- hello.c     (revision 5)
+++ hello.c     (working copy)
@@ -5,5 +5,5 @@

 int main(){
         printf("Hello World\n");
-        return 0;
+        return 1;
 }
```

Die veränderten Zeilen werden jeweils mit einem Additions- oder Subtraktionszeichen gekennzeichnet.

- + bedeutet, dass diese Zeile hinzugekommen ist
- - weist Zeilen aus, die weggekommen sind.

Das Diff-Kommando vergleicht dabei standardmäßig die geänderten Ordner mit dem `.svn`-Verzeichnis in der lokalen Arbeitskopie. Sie können aber auch über den Schalter `--revision` genau angeben, mit welcher Revision Sie Ihre aktuelle Datei vergleichen möchten:

```
svn diff --revision 2 hello.c
```

Mit einer Bereichsangabe können Sie noch genauer bestimmen, welche Revisionen sie vergleichen wollen:

```
svn diff --revision 2:3 hello.c
```

Sofern Sie keinen Dateinamen wie beispielsweise `hello.c` übergeben, werden alle Änderungen angezeigt. Dies kann unter Umständen zu einer längeren Ausgabe führen. `svn list` hat ebenfalls einen verbose-Schalter für eine detailliertere Ausgabe.

H.3 Log-Nachrichten: log

Ein zu `diff` ähnlicher arbeitender Befehl ist `svn log`. Log zeigt die Protokollmeldungen an, die Sie oder Ihre Kollegen bei einem Commit-Vorgang der jeweiligen Revision mitanheften. Mit

```
svn log
```

erhalten Sie alle Änderungen, Sie können diese aber auch noch Eingrenzen:

```
svn log --revision 2:3
```

`svn log` besitzt auch einen Verbose-Schalter, der noch genauere Informationen zu den Änderungen ausgeben kann:

```
svn log -r 8 -v
```

Auch können Sie die Protokoll-Historie einzelner Dateien ansehen, indem Sie die Datei direkt übergeben:

```
svn log dateiname
```

H.4 Historie untersuchen: list und cat

Sofern Sie den Werdegang der unterschiedlichen Dateien ansehen wollen haben Sie auch noch zwei weitere praktische Befehle: **svn cat** zeigt den Inhalt der Datei, wie er an einer bestimmten Revision vorlag. **svn list** hingegen gibt die Dateiein in einem Verzeichnis zu einer bestimmten Revision an:

```
svn list svn+ssh://server/path/to/repo
svn cat --revision 2 svn+ssh://server/path
    /to/repo/file.c
```

### H.5 Status ermitteln: status

Um den Status der lokalen Arbeitskopie abzugleichen können Sie den Befehl

```
svn status
```

aufrufen. Dieser Befehl vergleicht die lokale Kopie `nicht` mit dem Repository! Es prüft nur, ob Änderungen lokal durchgeführt wurden. Die Ausgabe zeigt diese Änderungen an und informiert den Benutzer ebenfalls über die Art der Änderung. Sofern Sie die Änderungen aus dem Repository miteinbeziehen wollen können Sie den Schalter `-u` aktivieren:

```
svn status -u -v
```

Der Schalter `-v` gibt noch genauere Angaben zu den Dateien zurück. Es ist auch möglich, eine Datei speziell auf Updates zu prüfen, ohne dass andere Änderungen ermittelt oder überprüft werden. Dateien, die mit einem Sternchen ausgezeichnet sind, markieren Dateien die im Repository hinzugefügt oder geändert wurden. Ein `svn update` holt diese Dateien in die lokale Arbeitskopie und würde somit erst ein `Commit` ermöglichen.

### *I. Subversion-Hilfe*

Subversion besitzt auch eine Hilfsfunktion. Diese kann zum Beispiel als Nachschlagewerk sehr praktisch sein und ist standardmäßig immer installiert:

```
svn help
svn help <befehl>
```

### *J. Subversion-Kürzel*

Subversion arbeitetsehr häufig mit Buchstaben, die eine bestimmte Veränderung einer Datei kenntlich machen:

| | |
|---|---|
| A | Added, also hinzugefügt |
| C | Conflict, Datei die konfliktbehaftet ist |
| D | Deleted |
| M | Modified (verändert) |
| X | nicht versioniert |
| U | Datei wurde aktualisiert |

## IV. Verzweigungen und Zusammenführungen

Meist verläuft die Entwicklung eines Softwareprojekts nicht geradlinig ab! Daher sind unterschiedliche Zweige keine Ausnahmen sondern stellen häufig den normalen Entwicklungsprozess dar. Bei Verzweigungen handelt sich um Entwicklungslinien, die unabhängig von einer anderen Linie existiert.

### *A. Zweig anlegen*

In Subversion gibt es keinen eigenen Befehl. Das Anlegen eines Zweigs wird über `svn copy` erledigt.

```
svn checkout http://svn.example.com/repos/
    GrossesProjekt projekt
cd projekt
svn copy trunk/ branches/
    GrossesProjektVerzw
```

Das folgende Listing holt zunächst das komplette Projekt (mit den Ordnern branches und tags). Danach wird eine Kopie des Trunk-Ordners unterhalb von branches angelegt. Beim nächsten commit wird die neue Verzweigung ebenfalls im Repository gespeichert.

```
svn commit -m "Creating a private branch
    of GrossesProjekt (GrossesProjektVerzw
    )"
```

Möglich ist es auch, einen Branch ohne eine lokale Arbeitskopie anzulegen:

```
svn copy http://server/repos/
    GrossesProjekt/trunk
\http://server/GrossesProjekt/branches/
    GrossesProjektVerzw
\-m "Creating a private branch of /calc/
    trunk"
```

### *B. Zweig auschecken*

Der neu angelegte zweig kann dann wiederum ganz normal ausgecheckt werden.

```
svn checkout http://server/repos/
    GrossesProjekt/branches/
    GrossesProjektVerzw
```

### *C. Zweige zusammenführen*

Angenommen, zwei unterschiedliche Entwickler arbeiten an der gleichen Datei, integer.c, die sich aber ebenfalls in unterschiedlichen Zweigen befinden. Möchte man nun diese zwei Entwicklungsstränge wieder vereinigen, dann kann nun `svn merge` verwendet werden:

```
svn merge -r 343:344 http://server/repos/
    projekt/trunk
```

Da in Revisoin 343 die Änderung im Zweig vorgenommen wurde, und in der Revision 344 die Änderung des zweiten Entwicklers in das Repository übernommen wurden, können diese unterschiedlichen Entwicklungen der Datei zusammengeführt werden. Dieser Vorgang verändert die lokale Arbeitskopie.

Sofern Sie sich beim Zusammenführen unsicher sind haben Sie auch die Möglichkeit, nur eine Versuchszusammenführung zu starten:

```
svn merge --dry-run -r 343:344 http://
    server/repos/projekt/trunk
```

## V. Tags

Tags stellen Momentaufnahmen eines aktuellen Versionsstandes dar. Häufig wird dann ein Tag angelegt, wenn es sich zum Beispiel um die Auslieferung eines Softwareprodukts an einen Kunden handelt oder eine bestimmte Version erreicht wurde.

### *A. Tags anlegen*

Auch Tags werden einfach durch kopieren angelegt:

```
svn copy file:///path/to/repos/trunk \
file:///path/to/repos/tags/release-1.0 \
-m "Tagging the 1.0 release"
```

Häufig sollten Tags im nachhinein nicht mehr editierbar sein. Es ist also praktisch, wenn Sie durch Zugriffssteuerungen jegliche Schreibrechte von tags entfernen. So können Sie sicherstellen, dass niemand aus versehen Dateien ändert die später noch wichtig werden könnten.

## Literatur


[1] Kalid Azad *A Visual Guide to Version Control*,
[2] Kalid Azad *Intro to Distributed Version Control,*
[3] Frank Budszuhn *Subversion 1.5, Das Praxisbuch*, Galileo Computing, 3. Auflage, 2009.
[4] C. Michael Pilato, Ben Collins-Sussman, Brian W. Fitzpatrick *Version Control with Subversion, Second Edition*, O'Reilly Media, September 2008.