

Frank Budzuhn

## **Subversion 1.4**

# Auf einen Blick

<b>Vorwort</b> .....	15
 <b>Teil 1 Eine Einführung in Subversion</b>	
<b>1</b> <b>Einleitung</b> .....	21
<b>2</b> <b>Das Versionsmanagementsystem Subversion</b> .....	31
<b>3</b> <b>Der Entwicklungsprozess mit Subversion</b> .....	43
<b>4</b> <b>Installation</b> .....	59
<b>5</b> <b>Erste Schritte</b> .....	69
<b>6</b> <b>Der Entwicklungsprozess im Detail</b> .....	81
<b>7</b> <b>Fortgeschrittene Themen</b> .....	143
<b>8</b> <b>Die Administration von Subversion</b> .....	197
<b>9</b> <b>Subversion für CVS-Benutzer</b> .....	235
<b>10</b> <b>Ausblick</b> .....	253
 <b>Teil 2 Referenz</b>	
<b>11</b> <b>Subversion-Befehle</b> .....	257
<b>12</b> <b>Referenz der lokalen Konfigurationsdateien</b> .....	335
<b>A</b> <b>Subversion ohne Server verwenden</b> .....	345
<b>B</b> <b>Ein Leitfaden für Projektleiter</b> .....	349
<b>C</b> <b>Glossar</b> .....	353
<b>D</b> <b>Link- und Literaturverzeichnis</b> .....	359
<b>Index</b> .....	365

# Inhalt

Vorwort .....	15
---------------	----

## TEIL 1 EINE EINFÜHRUNG IN SUBVERSION

<b>1 Einleitung .....</b>	<b>21</b>
1.1 Zielgruppe des Buchs .....	21
1.2 Aufbau des Buchs .....	22
1.3 Anforderungen an den Leser .....	23
1.4 Die verwendeten Betriebssysteme .....	23
1.5 Konventionen in diesem Buch .....	24
1.6 Wofür Versionsmanagement? .....	25
1.6.1 Arbeit ohne Versionsmanagement .....	25
1.6.2 Ein zweiter Entwickler kommt hinzu .....	28
1.7 Entwickeln mit Versionsmanagement .....	28
1.7.1 Erweiterter Entwicklungsprozess mit Subversion .....	28
1.7.2 Die Änderungen im Einzelnen .....	29
<b>2 Das Versionsmanagementsystem Subversion .....</b>	<b>31</b>
2.1 Zur Geschichte von Subversion .....	31
2.2 Subversion im Kontext anderer Versionsmanagementsysteme .....	33
2.3 Clientprogramme für Subversion .....	34
2.3.1 TortoiseSVN .....	34
2.3.2 RapidSVN .....	35
2.3.3 eSvn .....	35
2.3.4 Eclipse .....	36
2.4 Was Subversion nicht kann – Abgrenzung zu anderen Entwicklungswerkzeugen .....	38
2.5 Die Architektur von Subversion .....	38
2.6 Subversion und Open Source .....	40
<b>3 Der Entwicklungsprozess mit Subversion .....</b>	<b>43</b>
3.1 Modell des kooperativen Entwickelns .....	43
3.2 Betrachtungen zum ersten Kontakt .....	44
3.3 Der Entwicklungszyklus mit Subversion .....	44
3.3.1 Eine Arbeitskopie von Subversion anfordern: Checkout ...	46

3.3.2	Abgleich der Arbeitskopie mit dem Repository: Update ..	46
3.3.3	Auflösung von aufgetretenen Konflikten .....	47
3.3.4	Entwicklung auf der Arbeitskopie bis neue Teilversion erreicht ist .....	48
3.3.5	Übernahme der Änderungen aus der lokalen Arbeitskopie in das Repository: Commit .....	49
3.4	Der Entwicklungszyklus in der Zusammenfassung .....	50
3.5	Der Entwicklungszyklus mit mehreren Entwicklern .....	55
3.6	Subversion und Kommunikation .....	56
3.7	Regeln im Umgang mit Subversion .....	56
3.8	Zusammenfassung .....	57

## **4 Installation .....** **59**

4.1	Installation unter Windows .....	59
4.1.1	Installation durch das Installationsprogramm .....	59
4.1.2	Subversion deinstallieren .....	61
4.1.3	Manuelle Installation .....	62
4.2	Installation unter Debian Linux .....	64
4.3	Installation auf anderen Linux- und Unix-Systemen .....	64
4.4	Die Programme und Module von Subversion .....	65
4.4.1	Die Kommandozeilenprogramme .....	65
4.4.2	Das Apache Modul .....	65
4.5	Die Verbindung zum Repository herstellen .....	66
4.6	Zusammenfassung .....	67

## **5 Erste Schritte .....** **69**

5.1	Ein erster Test .....	69
5.2	Protokoll einer Beispielsitzung .....	73
5.3	Zusammenfassung .....	79

## **6 Der Entwicklungsprozess im Detail .....** **81**

6.1	Revisionen .....	81
6.1.1	Revisionsschlüsselwörter .....	82
6.1.2	Gemischte Revisionen in der lokalen Arbeitskopie .....	82
6.2	Arbeitsweisen von Subversion-Befehlen .....	84
6.3	Repository Layout .....	85
6.4	Der Repository-Browser .....	87
6.5	Die eingebauten Hilfsfunktionen von Subversion .....	88

6.6	Implizite Argumente und Rekursion .....	90
6.6.1	Implizite Argumente .....	91
6.6.2	Rekursion .....	91
6.7	Ein neues Projekt beginnen: import .....	92
6.7.1	Eine Projektstruktur anlegen .....	92
6.7.2	Die Struktur importieren .....	92
6.7.3	Dateien vom Import ausschließen .....	92
6.7.4	Nach dem Import .....	94
6.8	Eine lokale Arbeitskopie anlegen: checkout .....	95
6.8.1	Die Arbeit an einem Projekt beginnen .....	95
6.8.2	Die Optionen des Befehls checkout .....	96
6.9	Eine lokale Arbeitskopie aktualisieren: update .....	97
6.9.1	Mögliche Fälle beim Befehl update .....	97
6.9.2	Die Optionen des Befehls update .....	101
6.10	Änderungen in das Repository übernehmen: commit .....	102
6.10.1	Überführung ins Repository .....	102
6.10.2	Log Messages .....	103
6.10.3	Die Implementierung des Befehls commit .....	104
6.11	Unterschiede zwischen lokaler Arbeitskopie und Repository bestimmen: diff .....	104
6.12	Den Zustand der Arbeitskopie abfragen: status .....	106
6.13	Die Historie von Dateien und Verzeichnissen verfolgen: log .....	110
6.14	Dateien und Verzeichnisse auflisten: list .....	113
6.15	Dateien anzeigen: cat .....	116
6.16	Dateien und Verzeichnisse hinzufügen: add .....	117
6.17	Dateien und Verzeichnisse löschen: delete .....	119
6.18	Dateien und Verzeichnisse kopieren: copy .....	119
6.19	Dateien und Verzeichnisse verschieben und umbenennen: move .....	121
6.20	Verzeichnisse unter Versionskontrolle anlegen: mkdir .....	123
6.21	Lokale Änderungen zurück nehmen: revert .....	123
6.22	Einen Versionsstand aufbewahren .....	124
6.23	Die Arbeit mit Verzweigungen .....	127
6.23.1	Gründe für Verzweigungen .....	127
6.23.2	Verzweigungen in Subversion .....	127
6.23.3	Zweige zusammen führen: merge .....	131
6.23.4	Unterverzweigungen .....	137
6.23.5	Einsatzbereiche von Verzweigungen .....	138
6.24	Änderungen rückgängig machen .....	138
6.25	Gelöschte Dateien und Verzeichnisse wiederherstellen .....	139

6.26 Sperren entfernen: cleanup ..... 141  
 6.27 Zusammenfassung ..... 142

**7 Fortgeschrittene Themen ..... 143**

7.1 Befehle abkürzen ..... 143  
 7.2 Lokale Arbeitskopien »umschalten«: switch ..... 144  
 7.3 Die .svn-Verzeichnisse in der lokalen Arbeitskopie ..... 146  
 7.4 Sourcecode exportieren ..... 148  
 7.5 Die Umgebungsvariable SVN\_EDITOR ..... 149  
 7.6 Die lokale Konfiguration des Subversion-Clients ..... 150  
     7.6.1 Das Verzeichnis auth ..... 152  
     7.6.2 Die Datei config ..... 152  
     7.6.3 Die Datei servers ..... 153  
     7.6.4 Die Konfiguration für alle Benutzer eines Computers ..... 154  
     7.6.5 Konfiguration über die Windows-Registry ..... 155  
 7.7 Die Sprache der Subversion-Programme umschalten ..... 156  
 7.8 Dateien zeilenweise analysieren: blame ..... 156  
 7.9 Erweiterte Informationen anzeigen: info ..... 158  
 7.10 Properties ..... 158  
     7.10.1 svn:mime-type ..... 162  
     7.10.2 svn:executable ..... 163  
     7.10.3 svn:ignore ..... 164  
     7.10.4 svn:keywords ..... 166  
     7.10.5 svn:eol-style ..... 166  
     7.10.6 svn:externals ..... 167  
     7.10.7 svn:special ..... 167  
     7.10.8 svn:needs-lock ..... 168  
     7.10.9 Automatisches Setzen von Properties ..... 168  
     7.10.10 Revisionsbezogene Properties ..... 169  
 7.11 Symbolische Links unter Unix ..... 170  
 7.12 Externals ..... 171  
 7.13 Vendor Branches ..... 173  
     7.13.1 Einbindung von fremder Software ..... 173  
     7.13.2 Die generelle Arbeitsweise ..... 173  
 7.14 Datums- und Zeitangaben in Subversion ..... 176  
 7.15 XML, HTML und Subversion ..... 176  
     7.15.1 Besonderheiten von XML und HTML ..... 177  
     7.15.2 Merging Algorithmus in Subversion ..... 177  
 7.16 Webseiten mit Subversion verwalten ..... 177

7.17	Webfrontends für Subversion .....	179
7.17.1	ViewVC .....	179
7.17.2	WebSVN .....	181
7.18	Schlüsselwortersetzung .....	183
7.19	Das Arbeiten mit Sperren .....	185
7.19.1	Sperrkommentare .....	191
7.19.2	Sperren brechen und stehlen .....	192
7.19.3	Die Implementierung von Sperren .....	194
7.20	Subversion und ASP.NET .....	194
7.21	Zusammenfassung .....	196

## **8 Die Administration von Subversion ..... 197**

8.1	Einen Subversion-Server aufsetzen .....	197
8.2	Die Installation von Subversion .....	199
8.3	Subversion selbst compilieren .....	199
8.4	Konfiguration von Repositories .....	202
8.4.1	Berkeley DB versus FSFS .....	202
8.4.2	Ein Repository anlegen .....	203
8.4.3	svnserve einrichten .....	204
8.4.4	Pfadbasierte Autorisierung .....	205
8.4.5	Den Zugriff per SSH tunneln .....	207
8.4.6	svnserve als Windows-Dienst einrichten .....	208
8.4.7	Subversion und Apache .....	210
8.4.8	Basic HTTP Authentifizierung .....	212
8.4.9	Zwischen Lese- und Schreibzugriffen unterscheiden .....	213
8.4.10	Autorisierung .....	214
8.4.11	Verschlüsselung mit SSL .....	216
8.5	Wartung und Problembhebung .....	218
8.5.1	Ein Berkeley DB Repository restaurieren .....	218
8.5.2	Ein Repository überprüfen .....	219
8.5.3	Log Messages ändern .....	220
8.5.4	Ein Repository inspizieren: svnlook .....	220
8.6	Hook-Skripte .....	222
8.6.1	Die Änderung revisionsbezogener Properties zulassen .....	226
8.6.2	Mitgelieferte Skripte .....	227
8.6.3	RSS-Beispiel .....	227
8.7	Backup und Migration .....	228
8.7.1	Dumps .....	229
8.7.2	Direktes Sichern der Repository-Dateien .....	232
8.8	Repositories mit dem Programm svnsync kopieren .....	233
8.9	Zusammenfassung .....	234

<b>9</b>	<b>Subversion für CVS-Benutzer .....</b>	<b>235</b>
9.1	Subversion und CVS sind sich grundsätzlich ähnlich .....	235
9.2	Lokale Arbeitskopien .....	236
9.3	Der Kommandozeilen-Client .....	237
9.4	Revisionsnummern .....	237
9.5	Versionsverwaltung für Verzeichnisse .....	238
9.6	Atomare Commits .....	239
9.7	Zugriffsverfahren .....	239
9.8	Tags und Verzweigungen .....	241
9.9	Behandlung binärer Dateien .....	243
9.10	Überwachtes Arbeiten .....	244
9.11	Schlüsselwortersetzung .....	244
9.12	Vendor Branches .....	245
9.13	Unterschiede in der Implementierung .....	245
9.14	Konvertierung bestehender Repositories .....	247
9.15	Zusammenfassung .....	251

<b>10</b>	<b>Ausblick .....</b>	<b>253</b>
-----------	-----------------------	------------

**TEIL 2 REFERENZ**

<b>11</b>	<b>Subversion-Befehle .....</b>	<b>257</b>
11.1	Befehlsaufbau .....	257
11.2	svn .....	258
11.2.1	add .....	258
11.2.2	blame .....	259
11.2.3	cat .....	261
11.2.4	checkout .....	262
11.2.5	cleanup .....	263
11.2.6	commit .....	264
11.2.7	copy .....	266
11.2.8	delete .....	268
11.2.9	diff .....	270
11.2.10	export .....	273
11.2.11	help .....	274
11.2.12	import .....	275
11.2.13	info .....	277
11.2.14	list .....	278
11.2.15	lock .....	279

11.2.16	log	281
11.2.17	merge	284
11.2.18	mkdir	286
11.2.19	move	288
11.2.20	propdel	290
11.2.21	propedit	291
11.2.22	propget	292
11.2.23	proplist	294
11.2.24	propset	295
11.2.25	resolved	297
11.2.26	revert	298
11.2.27	status	299
11.2.28	switch	301
11.2.29	unlock	303
11.2.30	update	304
11.3	svnadmin	305
11.3.1	create	305
11.3.2	deltify	306
11.3.3	dump	306
11.3.4	help	308
11.3.5	hotcopy	308
11.3.6	list-dblogs	309
11.3.7	list-unused-dblogs	309
11.3.8	load	309
11.3.9	lstxns	311
11.3.10	recover	311
11.3.11	rmtxns	312
11.3.12	setlog	313
11.3.13	verify	313
11.4	svndumpfilter	314
11.4.1	exclude	314
11.4.2	help	315
11.4.3	include	316
11.5	svnlook	317
11.5.1	author	317
11.5.2	cat	318
11.5.3	changed	319
11.5.4	date	319
11.5.5	diff	320
11.5.6	dirs-changed	321

11.5.7	help	322
11.5.8	history	322
11.5.9	info	323
11.5.10	lock	324
11.5.11	log	324
11.5.12	propget	325
11.5.13	proplist	325
11.5.14	tree	326
11.5.15	uuid	327
11.5.16	youngest	328
11.6	svnserve	328
11.7	svnsync	329
11.7.1	copy-revprops	330
11.7.2	help	330
11.7.3	initialize	331
11.7.4	synchronize	331
11.8	svnversion	332

**12 Referenz der lokalen Konfigurationsdateien ..... 335**

12.1	Die lokalen Konfigurationsverzeichnisse	335
12.1.1	Die Struktur der lokalen Konfigurationsverzeichnisse	336
12.1.2	Konfiguration auf Windows (Dateien)	336
12.1.3	Konfiguration auf Windows (Registry)	337
12.1.4	Konfiguration auf Unix	337
12.2	Die Datei config	337
12.2.1	Die Sektion [auth]	338
12.2.2	Die Sektion [helpers]	338
12.2.3	Die Sektion [tunnels]	338
12.2.4	Die Sektion [miscellany]	339
12.2.5	Die Sektion [auto-props]	339
12.3	Die Datei servers	340
12.3.1	Die Sektion [groups]	340
12.3.2	Die Sektion [globals]	340
12.3.3	Die selbst definierten Sektionen	341

**Anhang ..... 343**

A	Subversion ohne Server verwenden	345
A.1	Der lokale Zugriff auf das Repository	345
A.2	Die Beispieldateien installieren	346

B	Ein Leitfaden für Projektleiter .....	349
C	Glossar .....	353
D	Link- und Literaturverzeichnis .....	359
D.1	Internetlinks .....	359
D.2	Bücher .....	362
D.3	Mailinglisten .....	363
	Index .....	365

*Dieses Kapitel stellt zunächst ein paar Gedanken zum Verhältnis zwischen Sourcecode und Entwickler in einem Entwicklerteam vor, betrachtet wahrscheinliche Umstände des ersten Kontakts zwischen Entwickler und Subversion und zeigt dann den typischen Entwicklungszyklus mit Subversion.*

## 3 Der Entwicklungsprozess mit Subversion

Um es vorweg zu nehmen: Es kann durchaus sinnvoll sein, Subversion als einzelner Entwickler einzusetzen, denn auch ein einzelner Entwickler muss Versionsstände verwalten und hat gegebenenfalls das Problem der Verzweigung in seiner Entwicklungslinie. Doch die Verwendung von Subversion durch nur einen Entwickler ist nicht der Normalfall. Meist wird Subversion neben der reinen Versionsverwaltung auch zur Distribution und Synchronisation des Sourcecodes mehrerer Entwickler eingesetzt.

Einzelner  
Entwickler

### 3.1 Modell des kooperativen Entwickelns

Die grundsätzlichen Änderungen, die einen Entwickler bei der Verwendung von Subversion betreffen, sind in Kapitel 1 bereits dargestellt worden. Die erste Änderung beim Einsatz eines Versionsmanagementsystems wie Subversion sollte jedoch im Kopf des Entwicklers statt finden: der Sourcecode gehört nicht dem einzelnen Entwickler sondern dem Team als Ganzen. Er wird nicht vom einzelnen Entwickler erstellt sondern vom gesamten Team. Der einzelne Entwickler sollte sich von dem Gedanken trennen, dass der von ihm erstellte Sourcecode sein Eigentum ist (rein rechtlich ist er es meist sowie so nicht, da viele angestellte Programmierer die Rechte an von ihnen geschaffener Software per Angestelltenvertrag an ihren Arbeitgeber abtreten).

Sourcecode  
wird geteilt

Der Entwickler eines Teams arbeitet immer auf einer lokalen Kopie des Sourcecodes, niemals auf der Master-Kopie. Die Master-Kopie wird im Repository des Subversion-Servers gespeichert. Weil jeder Entwickler

Arbeit auf Kopien

nur eine Kopie erhält, gibt der einzelne Entwickler auch die Kontrolle über den Sourcecode an Subversion ab. Er muss sich weder um die Sicherung noch um das Aufbewahren von Versionsständen kümmern.

**Code-Distribution** Der Entwickler muss sich daran gewöhnen, dass Subversion auch die Distribution von Sourcecode übernimmt. Es treffen laufend Änderungen anderer Entwickler ein (so man Subversion nicht alleine nutzt) und die eigenen Änderungen werden umgekehrt genauso weiter verteilt. Das kann anfangs etwas ungewohnt sein. Den Zeitpunkt des Abgleichs mit Subversion bestimmt der Entwickler allerdings immer noch weitgehend selbst.

### 3.2 Betrachtungen zum ersten Kontakt

Viele Entwickler haben ihren ersten Kontakt zu einem Versionsmanagementsystem wie Subversion innerhalb eines Firmenprojekts, in dem die Zeit knapp bemessen ist. Das Projekt benötigt neue Ressourcen, also mehr Programmierer, ist sowieso schon hinter dem Zeitplan und allgemein hat niemand der Projektmitglieder die Muße, einen in die Geheimnisse des Versionsmanagement einzuführen. Oft bekommt der Entwickler vom Projektleiter eine kurze mechanische Einweisung, welche Befehlsfolgen auszuführen sind, um Sourcecode aus und in das Versionsmanagementsystem zu bekommen. Ein echtes Verständnis für das System bildet sich dabei meist nicht heraus, genauso mechanisch wie die Befehlsfolgen erlernt worden sind, werden sie auch ausgeführt. Es bleibt die Unsicherheit, etwas »kaputt zu machen« und das System nicht richtig zu bedienen. Meist macht ein einzelner Entwickler zwar aus Unwissenheit wenig »kaputt«, jedoch kann das Unverständnis des Systems durchaus zu Problemen und Mehrarbeit führen. Deswegen ist es wichtig, vor dem ersten Einsatz von Subversion ein grundsätzliches Bild des Systems im Kopf zu haben. Dieses soll helfen, die durchgeführten Operationen und Befehle zu begreifen und in ihren Auswirkungen auf das Repository zu verstehen.

### 3.3 Der Entwicklungszyklus mit Subversion

**Arbeitsschritte im Entwicklungsprozess** Genau wie die Softwareentwicklung selbst ein zyklischer Prozess ist (im kleinsten ist das oft der Edit-Compile-Run-Zyklus), so ist auch die Arbeit mit Subversion ein zyklischer Prozess. Die Arbeitsschritte in diesem Prozess sind nicht streng vorgegeben, normalerweise folgen sie aber etwa diesem Ablauf:

- ▶ Eine Arbeitskopie vom Subversion-Server anfordern

oder

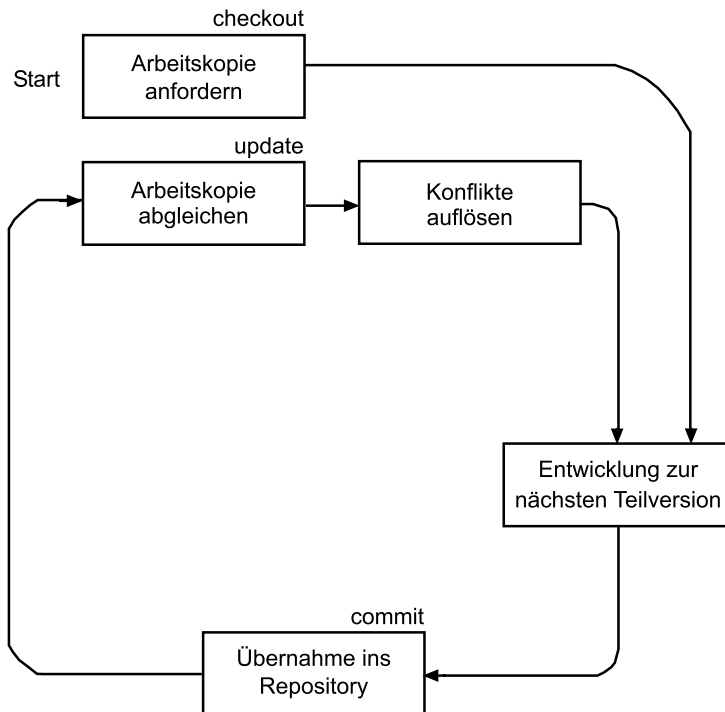
- ▶ Eine bestehende Arbeitskopie mit dem Repository abgleichen
- ▶ Auflösung von aufgetretenen Konflikten

dann

- ▶ Entwicklung auf der Arbeitskopie bis neue Teilversion erreicht ist
- ▶ Übernahme der Änderungen aus der lokalen Arbeitskopie in das Repository

Mit Beginn des zweiten Durchlaufs wird im ersten Schritt normalerweise eine bestehende Arbeitskopie mit dem Repository abgeglichen, statt eine neue lokale Arbeitskopie anzufordern. Abbildung 3.1 zeigt den Entwicklungszyklus.

Der zweite Durchlauf



**Abbildung 3.1** Der Entwicklungszyklus mit Subversion

Die einzelnen Schritte sollen nun im folgenden detailliert erläutert werden.

### 3.3.1 Eine Arbeitskopie von Subversion anfordern: Checkout

Es wird immer  
auf einer Kopie  
gearbeitet

Der erste Schritt für einen Entwickler, vorausgesetzt es gibt bereits Sourcecode im Repository, ist eine lokale Arbeitskopie des Sourcecodes vom Subversion-Server anzufordern. Ein Entwickler arbeitet **immer** auf einer Kopie des Sourcecodes, niemals auf denen im Repository befindlichen Originaldaten. Die Anforderung von Sourcecode aus dem Repository wird als *Checkout* bezeichnet. Der Checkout liefert einen Baum von Sourcecode-Dateien und Verzeichnissen. Wie der Sourcecode ins Repository gekommen ist, soll an späterer Stelle beschrieben werden (siehe dazu Abschnitt 6.7, »Ein neues Projekt beginnen: import«). Beim Checkout fordert der Entwickler eine Kopie aller (Unter-)Verzeichnisse und Dateien eines Verzeichnisses im Repository an. Wichtig für das Verständnis von Subversion ist dabei zu wissen, dass ein Checkout keine Zustandsänderung auf dem Server bewirkt, der Subversion-Server führt kein Buch über die Checkouts. Der Entwickler kann also eine Arbeitskopie anfordern, sie löschen und nochmals wieder eine Kopie anfordern. Subversion ist es »egal« wie viele Kopien im Umlauf sind, das Versionsmanagementsystem »weiß« es nicht. Die Arbeit mit lokalen Kopien bedeutet auch, dass alle Änderungen, die an den lokalen Dateien und Verzeichnissen vorgenommen werden, zunächst für niemand anderes sichtbar werden. Es werden auch keine Dateien gesperrt, so dass jeder Entwickler jederzeit alles in seiner lokalen Kopie ändern kann.

### 3.3.2 Abgleich der Arbeitskopie mit dem Repository: Update

In den meisten Fällen besitzt der Entwickler bereits eine lokale Arbeitskopie. In diesen Fällen wird der Entwickler kein Checkout durchführen, sondern er wird die vorhandene lokale Arbeitskopie mit dem Repository abgleichen. Dieser Abgleich wird als *Update* bezeichnet. Beim Update werden die Änderungen aus dem Repository in die lokale Arbeitskopie übernommen. Es wird ausschließlich die lokale Arbeitskopie verändert, das Repository wird nicht berührt. Auch wird auf dem Subversion-Server keine Zustandsänderung durch das Update bewirkt, genau wie ein Checkout ist auch ein Update ohne Auswirkung auf andere Entwickler.

Das Update überträgt alle Änderungen aus dem Repository in die lokale Arbeitskopie. Nicht veränderte Dateien und Verzeichnisse werden übersprungen. Nur im Repository veränderte Dateien und neue Dateien und neue Verzeichnisse im Repository werden einfach in die lokale Arbeitskopie übernommen, alte Dateien in der lokalen Arbeitskopie werden dabei überschrieben. Ausschließlich in der lokalen Arbeitskopie verän-

derte Dateien werden bei einem Update nicht angefasst. Sollten sich Änderungen an einer Datei sowohl im Repository als auch in der lokalen Arbeitskopie ergeben haben, so versucht Subversion die beiden Dateien zu einer zusammen zu führen. Dieser Vorgang wird als *Merging* bezeichnet.

Beim Merging vergleicht Subversion zeilenweise die Datei aus dem Repository mit der Datei aus der lokalen Arbeitskopie und erzeugt daraus eine neue Datei. Haben sich Änderungen an einer Stelle nur in einer der beiden Versionen ergeben, so werden diese in die Ergebnisdatei übernommen. Gibt es dagegen Änderungen an gleicher Stelle in beiden Dateien, so muss Subversion aufgeben. Es lässt sich automatisch nicht mehr bestimmen, welche Version korrekt ist. Dieser Vorfall wird als *Konflikt* bezeichnet. Die Tabelle zeigt alle Fälle, die beim Update einer Datei auftreten können.

Merging

lokale Arbeitskopie	Repository	Aktion	Zustand danach in der lokalen Arbeitskopie
Datei nicht verändert	Datei nicht verändert	Keine Aktion: Überspringen	Nicht modifiziert
Datei verändert	Datei nicht verändert	Keine Aktion	Modifiziert
Datei nicht verändert	Datei verändert	Kopieren	Nicht modifiziert
Datei verändert	Datei verändert	Merge	Falls Merge erfolgreich: modifiziert, sonst Konflikt
Datei oder Verzeichnis nicht vorhanden	Neue Datei oder neues Verzeichnis vorhanden	Kopieren bzw. anlegen	Nicht modifiziert
Datei oder Verzeichnis nicht verändert	Datei oder Verzeichnis gelöscht	Löschen	Nicht vorhanden
Neue Datei oder neues Verzeichnis vorhanden	Datei oder Verzeichnis nicht vorhanden	Keine Aktion	Datei oder Verzeichnis unbekannt. Muss erst durch den Befehl <b>add</b> dem Repository hinzugefügt werden.

### 3.3.3 Auflösung von aufgetretenen Konflikten

Bei einem Konflikt gibt Subversion eine entsprechende Kennzeichnung aus und platziert so genannte *Konfliktmarker* innerhalb der Datei: Es werden beide Versionen in die Datei übernommen und durch deutlich

Konfliktmarker

erkennbare Strichlinien voneinander und vom Rest des Sourcecodes abtrennt. Die Konfliktmarker sind so gewählt, dass sie in allen gängigen Programmiersprachen Fehlermeldungen beim Compilieren oder bei der Ausführung verursachen.

**Konflikte auflösen** Sind Konflikte aufgetreten, so müssen diese vom Entwickler in der lokalen Arbeitskopie aufgelöst werden. Meist entscheidet er sich in Absprache mit seinen Entwicklerkollegen für eine der beiden Versionen oder baut ein Gesamtwerk aus den beiden Varianten. Ist der Konflikt beseitigt, so muss dies Subversion explizit mitgeteilt werden, so dass es den Zustand der betroffenen Dateien zurück setzt. Konflikte betreffen nur die lokale Arbeitskopie, auf dem Server selbst gibt es keinen Konflikt.

**Konflikte sind selten** Allen Subversion-Anfängern sei an dieser Stelle gesagt, dass Konflikte seltener auftreten, als man zunächst vermuten sollte. Schließlich arbeiten die einzelnen Entwickler eines Entwicklerteams meist an unterschiedlichen Teilaspekten und damit auch an unterschiedlichen Stellen im Programmcode. Konflikte deuten daher meist auf mangelnde Absprache unter den Entwicklern hin.

### 3.3.4 Entwicklung auf der Arbeitskopie bis neue Teilversion erreicht ist

Nach dem Checkout oder dem Update tut der Entwickler, was er auch ohne Subversion tun würde. Er arbeitet ganz normal mit seiner Entwicklungsumgebung auf den Dateien der lokalen Arbeitskopie. Es findet keine Interaktion mit Subversion und damit auch nicht mit anderen Entwicklern statt.

**Zeitpunkt der Übernahme ins Repository** Irgendwann erreicht der Entwickler einen Punkt, an dem er meint, dass alle anderen Entwickler seines Teams seine Änderungen übernehmen könnten. Wann dieser Punkt erreicht ist, liegt im Ermessen des Entwicklers selbst. Welchen Zustand die Software zu diesem Zeitpunkt haben sollte, wird meist durch den Projektleiter festgelegt. So sollten sich generell alle Änderungen übersetzen (compilieren) lassen und andere Entwickler sollten in ihrer Arbeit durch die Neuerungen nicht behindert werden. Hat der Entwickler beispielsweise zu Testzwecken die Hauptmenüleiste ausgebaut, so ist dies nicht unbedingt eine Version, die in das Repository übernommen werden sollte, da dann andere Entwickler nicht mehr testen können.

Beim Erreichen einer stabilen Version hat der Entwickler zwei Möglichkeiten: entweder versucht er seine Änderungen direkt in Subversion zu übernehmen oder er gleicht zunächst seine Änderungen mit dem Repository ab. Schließlich können sich in der Zwischenzeit Änderungen durch andere Entwickler ergeben haben. Die erste Variante sollte nur gewählt werden, wenn die Entwicklungszeit kurz war und die im Repository gehaltene Version sich vermutlich kaum verändert hat. Die zweite Variante ist auf jeden Fall die defensivere Variante, da der Entwickler hier zunächst seine Änderungen mit denen der anderen Entwickler zusammenführt und auch testen kann, bevor er das Gesamtwerk in das Repository überführt. Ob der Entwickler vor der Übernahme seiner Änderungen ins Repository noch ein Update durchführt, wird oft auch von Erfahrungswerten bestimmt. Diese ergeben sich zwangsläufig nach einer gewissen Zeit bei der Arbeit an einem konkreten Projekt und dem Einsatz von Subversion.

Zwei mögliche Herangehensweisen

### 3.3.5 Übernahme der Änderungen aus der lokalen Arbeitskopie in das Repository: Commit

Nun ist der Entwickler so weit, dass er seine Änderungen in das Repository überführen möchte. Bis jetzt hat alles was er gemacht hat keinerlei Auswirkungen auf das Repository oder auf andere Entwickler gehabt. Würde er an dieser Stelle seinen Sourcecode einfach löschen, so würde niemand bemerken, dass er überhaupt an der Entwicklung beteiligt war. Erst mit der Übernahme der Änderungen aus der lokalen Kopie in das Repository wird dieses verändert und damit bekommen auch alle anderen Entwickler diese Neuerungen mit ihrem nächsten Update in ihre jeweiligen Arbeitskopien überführt. Die Übernahme aus der lokalen Kopie in das Repository wird als *Commit* oder auch als *Einchecken* bezeichnet. Ein Commit wird normalerweise nur auf Dateien und Verzeichnissen ausgeführt, die in der lokalen Kopie auch tatsächlich verändert worden sind.

Ein Commit kopiert die Änderungen des Entwicklers in das Repository. Für den Fall, dass eine Datei im Repository Änderungen enthält, die vom Entwickler noch nicht übernommen worden sind, verweigert Subversion das Commit. Die Zurückweisung des Commits durch Subversion darf nicht mit einem Konflikt verwechselt werden. Ein Konflikt kann grundsätzlich nur beim Update auftreten. Die Ablehnung des Commits durch Subversion erfolgt nur, weil die Revision im Repository neuer ist, als in der lokalen Arbeitskopie. Subversion kann aber nur ein Commit auf der aktuellen im Repository gespeicherten Revision durchführen,

Subversion kann das Einchecken verweigern

daher erzwingt es das Update um die aktuelle Revision aus dem Repository in die lokale Arbeitskopie zu übertragen. Erst dieses Update kann zu einem Konflikt führen, muss es aber nicht! Nach dem Update ist der Entwickler aufgefordert, eventuelle Konflikte zu beseitigen und dann das Commit erneut durchzuführen.

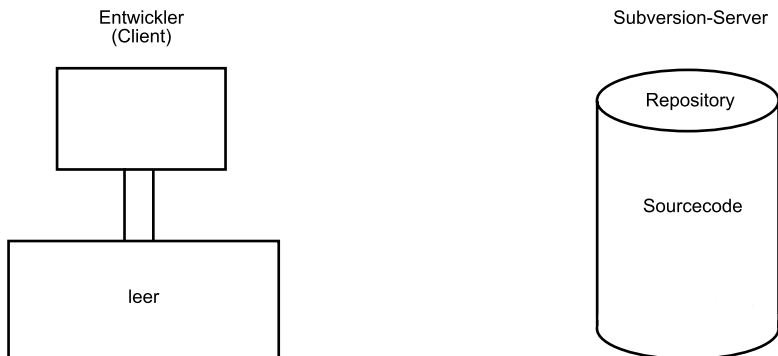
**Ein neuer Zyklus** Nach dem Commit kann der Entwickler einfach auf der Arbeitskopie weiterarbeiten. Oft führt ein Entwickler ein Commit zum Abschluss seines Arbeitstages durch, so dass er nach dem Commit Feierabend macht und die Arbeit unterbricht. Am nächsten Arbeitstag führt der Entwickler dann ein Update auf der lokalen Arbeitskopie aus, um die Änderungen zu erhalten, die sich in der Zwischenzeit durch die Arbeit der anderen Entwickler ergeben haben. Damit leitet der Entwickler einen neuen Arbeitszyklus ein.

### 3.4 Der Entwicklungszyklus in der Zusammenfassung

Nun soll der Zyklus der Entwicklung mit Subversion nochmals zusammengefasst dargestellt werden:

- Wenn ein Entwickler mit der Entwicklung an einem bereits bestehenden Projekt beginnt, so besitzt er noch keinen lokalen Sourcecode des Projekts. Dieser ist nur im Repository des Subversion-Servers vorhanden. Dieser Zustand ist in Abbildung 3.2 dargestellt.

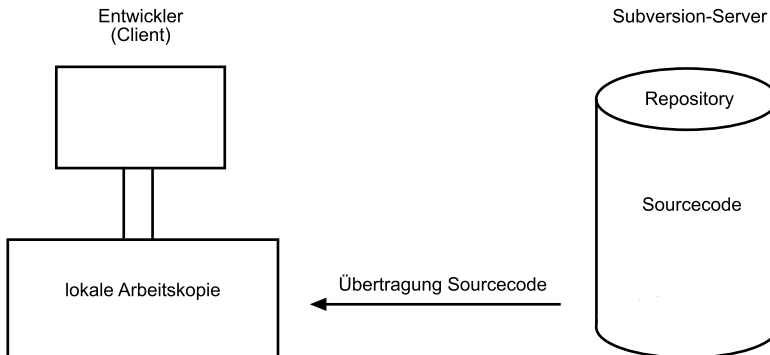
Start: Entwickler hat noch keinen Sourcecode



**Abbildung 3.2** Entwickler besitzt noch keinen Sourcecode

- ▶ Um mit der Entwicklung zu beginnen, fordert der Entwickler den Sourcecode vom Subversion-Server an. Dazu verwendet er den Befehl **checkout**. Der Sourcecode wird aus dem Repository in das Arbeitsverzeichnis des Entwicklers kopiert, es wird eine lokale Arbeitskopie angelegt. Dieser Vorgang wird nach dem gleichnamigen Befehl als *Checkout* bezeichnet und ist in Abbildung 3.3 zu sehen.

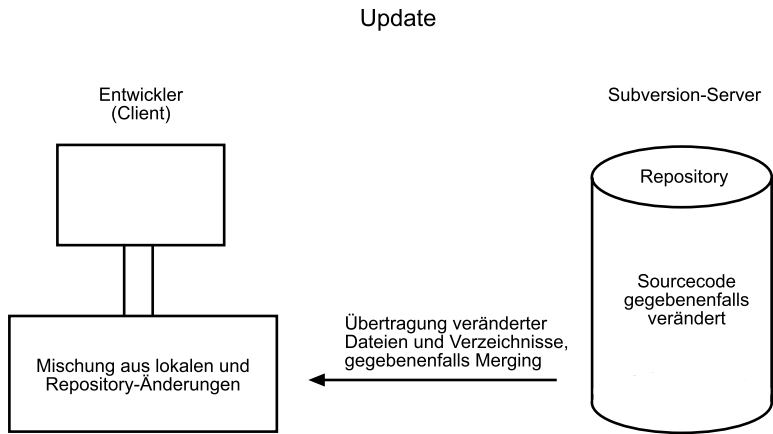
### Checkout



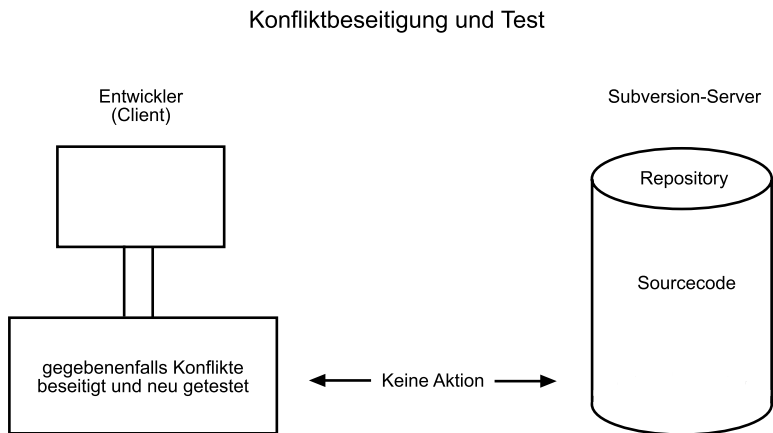
**Abbildung 3.3** Entwickler fordert Sourcecode an

- ▶ In den meisten Fällen besitzt der Entwickler schon eine Kopie des Sourcecodes in Form einer lokalen Arbeitskopie. In diesem Fall führt der Entwickler keinen Checkout durch. Statt dessen wird die bereits vorhandene lokale Arbeitskopie zu Beginn eines neuen Zyklus mit dem Repository abgeglichen. Dazu verwendet der Entwickler den Befehl **update**. Der Vorgang wird – entsprechend dem Namen des Befehls – ebenfalls als Update bezeichnet. Die Änderungen aus dem Repository werden in die lokale Arbeitskopie des Entwicklers kopiert. Der Ablauf ist in Abbildung 3.4 auf Seite 52 gezeigt.
- ▶ Sind vor dem Update einzelne Dateien sowohl im Repository als auch in der lokalen Arbeitskopie verändert worden, so versucht Subversion bei diesen Dateien, sie automatisch zusammenzuführen. Subversion führt einen *Merge* auf den Dateien durch. Nun kann es sein, dass dabei Konflikte auftreten. Diese muss der Entwickler manuell in der lokalen Arbeitskopie beseitigen. Danach sollte der Entwickler aus dem Sourcecode eine neue Programmversion erstellen und diese testen. Schließlich können beim Merging oder auch nur beim Update Programmeile verschiedener Entwickler zusammengekommen sein, die von der Programmlogik her nicht miteinander harmonieren und

im fertigen Programm Probleme verursachen. Konfliktbeseitigung und Test werden in Abbildung 3.5 gezeigt.



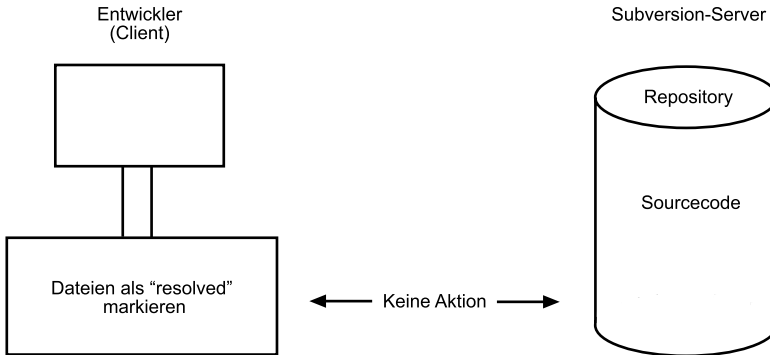
**Abbildung 3.4** Abgleich des Repositories mit der lokalen Arbeitskopie



**Abbildung 3.5** Konflikte beseitigen und testen

- ▶ Wenn der Entwickler einen Konflikt beseitigt hat, so muss er dies Subversion explizit mitteilen. Dazu ruft er den Befehl **resolved** auf. Solange der Befehl nicht aufgerufen worden ist, lassen sich keine Änderungen in das Repository übernehmen, da sich die vom Konflikt betroffenen Dateien weiterhin im Konfliktzustand befinden. Erst der Befehl **resolved** setzt diesen Zustand zurück. Dies ist in Abbildung 3.6 zu sehen.

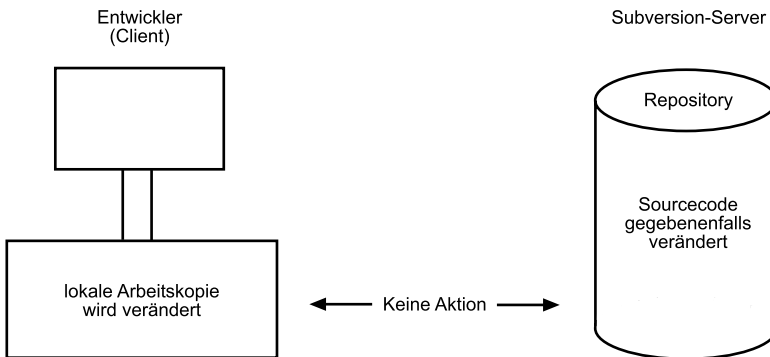
Entwickler setzt Konfliktzustand zurück



**Abbildung 3.6** Den Konfliktzustand zurück setzen

- Nun arbeitet der Entwickler auf seiner lokalen Arbeitskopie weiter bis er eine neue stabile Teilversion erreicht hat. Während dieser Zeit findet keine Interaktion mit dem Subversion-Server statt. Dies zeigt Abbildung 3.7.

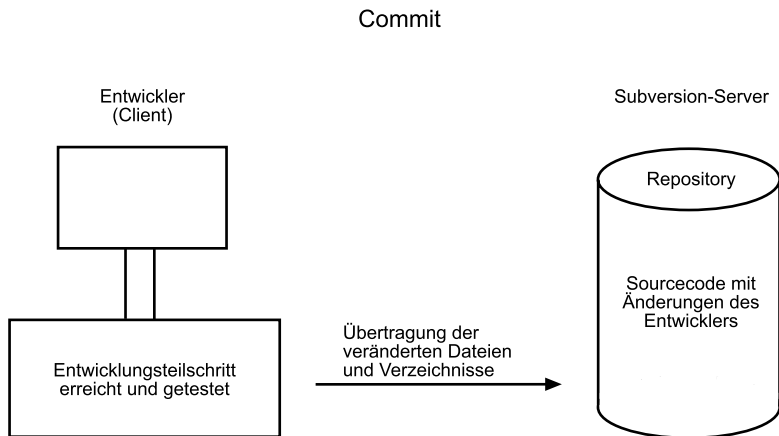
Entwicklung



**Abbildung 3.7** Keine Aktion während der Entwicklung

- Hat die Entwicklung der neuen Teilversion eine gewisse Zeit gebraucht, so kann der Entwickler nun nochmals seine Arbeitskopie mit dem Repository abgleichen (vergleiche Abbildung 3.4). Eventuell dabei auftretende Konflikte muss er beheben (vergleiche Abbildungen 3.5 und 3.6). Er kann sich allerdings auch dazu entschließen, gleich zum nächsten Schritt zu gehen und seine Änderungen einzuchecken.

- ▶ Der Entwickler überführt nun seine Änderungen in das Repository. Dieser Vorgang wird als *Einchecken* oder als *Commit* bezeichnet und ist in Abbildung 3.8 gezeigt. Die Änderungen des Entwicklers werden aus der lokalen Arbeitskopie in das Repository kopiert. Dazu verwendet der Entwickler den Befehl **commit**. Hat der Entwickler zuvor keinen Abgleich mit dem Repository durchgeführt und eine Datei oder ein Verzeichnis wurde in der lokalen Arbeitskopie und im Repository verändert, so verweigert Subversion das Commit. In diesem Fall muss der Entwickler auf jeden Fall den Befehl **update** auf der Datei beziehungsweise auf dem Verzeichnis durchführen, bevor er die diese einchecken kann. Das Update ist notwendig, um die aktuelle Revision der betreffenden Ressource in die lokale Arbeitskopie zu übernehmen. Ein Commit wird nämlich von Subversion nur auf der jeweils aktuellen Revision akzeptiert. Treten bei diesem Update Konflikte auf, so müssen diese beseitigt werden, bevor das Commit durchgeführt werden kann. Die Konflikte treten folglich immer beim Update in der lokalen Arbeitskopie auf, niemals beim Commit im Repository. Das Commit wird von Subversion eben genau dann verweigert, wenn es zu Konflikten kommen *könnte*.



**Abbildung 3.8** Änderungen ins Repository übernehmen

Nach dem Einchecken sind keine weiteren Schritte notwendig. Der Entwickler arbeitet einfach auf der bestehenden lokalen Arbeitskopie weiter. Der Entwicklungszyklus beginnt von vorn.

### 3.5 Der Entwicklungszyklus mit mehreren Entwicklern

Innerhalb eines Entwicklungsteams greift jeder Entwickler auf den Subversion-Server aus seinem eigenen Entwicklungszyklus heraus zu. Damit laufen mindestens genauso viele Entwicklungszyklen parallel ab, wie es Entwickler im Team gibt. (Es kann durchaus noch mehr Zyklen geben, wenn beispielsweise zu Testzwecken weitere Arbeitskopien aus dem Repository »gezogen« werden.) Welcher Zyklus sich in welchem Zustand befindet, ist nicht bestimmbar, es ist jederzeit jede Kombination von Zuständen denkbar. Es ist weder für Subversion noch für einen einzelnen Entwickler überhaupt feststellbar, wie viele Arbeitskopien und damit Entwicklungszyklen existieren. Subversion führt über Arbeitskopien kein Buch. Jeder Zyklus kann jederzeit beendet werden, es können jederzeit neue Zyklen entstehen. Wenn ein Entwickler dies wünscht, kann er auch auf mehreren Arbeitskopien gleichzeitig arbeiten und damit mehrere Zyklen führen.

Daher lässt sich folgern, dass ein Entwickler jederzeit davon ausgehen muss, dass sich das Repository seit dem letzten Zugriff auf den Subversion-Server verändert hat. Von den bisher beschriebenen Befehlen verändert nur **commit** das Repository. Zwar kennt Subversion noch mehr Befehle, die das Repository verändern, jedoch ist es der Befehl **commit**, der die Änderungen am Sourcecode über das Repository an die anderen Entwickler verteilt. Dies ist daher auch der Befehl, bei dem man sich vorher genau überlegen sollte, wann man ihn aufruft!

Da der Server über lokale Kopien kein Buch führt, können diese jederzeit gefahrlos entsorgt werden. Weder der Server, noch andere Entwickler werden davon betroffen.

Lokale Arbeitskopien können jederzeit gelöscht werden

Synchronisiert werden mehrere Entwicklungszyklen letztendlich durch die Befehle **update** und **commit**. Jeder ist dabei für eine Richtung zuständig. Aus Sicht des Entwicklers führt **update** die eingehende Synchronisation durch und **commit** die ausgehende.

Abbildung 3.9 auf der nächsten Seite zeigt einen Subversion-Server mit drei Entwicklungszyklen.

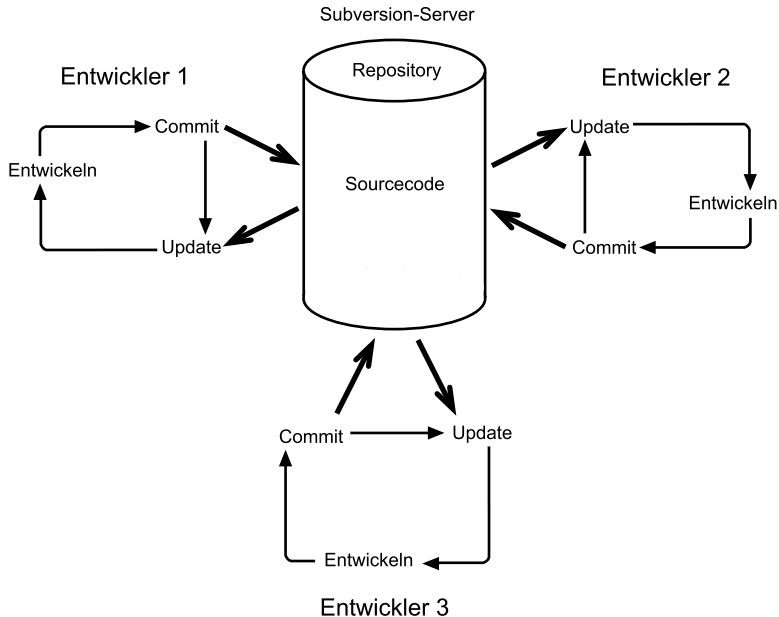


Abbildung 3.9 Subversion-Server mit drei Entwicklungszyklen

### 3.6 Subversion und Kommunikation

Subversion kann die Kommunikation in einem Entwicklerteam nicht ersetzen. Im Gegenteil: die Kommunikation der Entwickler untereinander ist unabdingbar! Eine gute Kommunikation und eindeutig verteilte Aufgaben der Entwickler helfen Konflikte zu vermeiden. Wenn jeder Entwickler ein ungefähres Bild der Aufgaben und Arbeitsgebiete der anderen Entwickler im Kopf hat, so wird er hoffentlich den zuständigen Entwickler fragen, bevor er sich an die Veränderung von Dateien macht, die eigentlich gar nicht in seinem Zuständigkeitsbereich liegen.

### 3.7 Regeln im Umgang mit Subversion

Für die Entwickler eines Teams ist es meist sinnvoll zur Verwendung von Subversion gewisse Regeln aufzustellen, um eine reibungslose Zusammenarbeit zu fördern. Oft werden diese Regeln durch den Projektleiter aufgestellt (siehe Anhang B, »Ein Leitfaden für Projektleiter«). Generell sollte der Zustand der in Subversion eingecheckten Sourcecode-Dateien geregelt sein. So ist es sinnvoll, nur compilierbare Dateien einzuchecken

und dafür Sorge zu tragen, dass die Lauffähigkeit des Gesamtsystems gegeben ist.

Ein paar Gedanken sollte sich der Entwickler auch dazu machen, wann und wie oft er Updates vornimmt und wann er seine eigenen Änderungen zurückführt (eincheckt). Häufige Updates halten einen immer auf den neuesten Stand können bei konzentrierter Entwicklung aber auch stören, da sie zu sofortigen Änderungen zwingen und damit aus dem eigentlichen Problem herausreißen können. Zu seltene Updates können viele Konflikte auslösen, schließlich hat sich in der Zwischenzeit viel getan und damit steigt auch die Wahrscheinlichkeit für Konflikte. Auch ein Commit sollte wohlüberlegt erfolgen. Meist bietet es sich an, wenn ein Teilproblem vollständig gelöst wurde und wenige Auswirkungen auf den Rest der Entwicklung zu erwarten sind. Idealerweise findet das gesamte Entwicklerteam zu einem gemeinsamen Rhythmus.

Häufigkeit von Updates

### 3.8 Zusammenfassung

Dieses Kapitel hat das theoretische Basiswissen vermittelt, das ein Entwickler besitzen sollte, wenn er mit Subversion arbeiten möchte. Ein besonderer Schwerpunkt ist auf die Beschreibung der grundsätzlich ablaufenden Prozesse und auf die Auswirkungen der Zusammenarbeit mehrerer Entwickler gelegt worden. Nun ist es an der Zeit, erste praktische Schritte zu unternehmen. Dazu werden in den nächsten beiden Kapiteln zunächst die Installation des notwendigen Client-Programms vorgenommen und danach eine erste Sitzung mit Subversion durchgespielt.



*Subversion lässt sich auf vielen Betriebssystemen installieren. In diesem Kapitel soll die Installation auf Windows und auf Linux gezeigt werden.*

## 4 Installation

Obwohl Client und Server zusammen installiert werden, geht es in diesem Kapitel schwerpunktmäßig um die Einrichtung des Client-Programms.

### 4.1 Installation unter Windows

Zunächst wird die Installation mittels Setup Wizard, danach die manuelle Variante gezeigt.

#### 4.1.1 Installation durch das Installationsprogramm

Die einfachste Art Subversion unter Windows zu installieren ist es, das dafür vorgesehene Installationsprogramm zu verwenden. Das Programm führt Windows-typisch in mehreren Schritten durch die Installation. Es ist nach dem Schema **svn-<versionsnummer>-setup.exe** benannt, also beispielsweise **svn-1.4.0-setup.exe** bei Version 1.4.0. Das Programm wird durch einen Doppelklick gestartet. Nach einer Abfrage, ob man Subversion auch wirklich installieren möchte, erscheint das in Abbildung 4.1 (siehe nächste Seite) gezeigte Dialogfeld.

Die einfachste Art

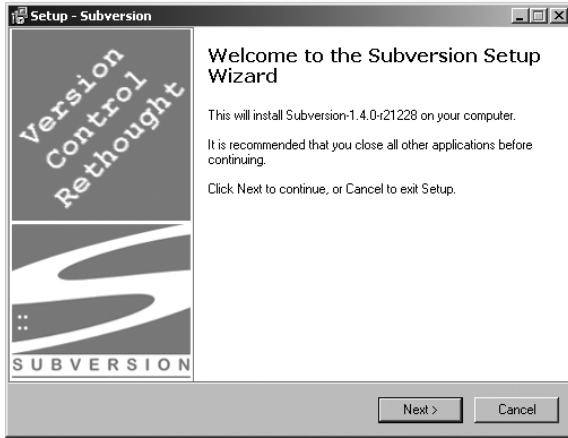
Im zweiten Schritt der Installation müssen die Lizenzbedingungen bestätigt werden. Bei der Lizenz handelt es sich um eine leicht abgewandelte Apache-style Open-Source-Lizenz.

Lizenz-  
bedingungen

Im dritten Schritt werden URLs zu aktuellen Informationen im Internet betreffend Subversion angezeigt.

In Schritt vier der Installation wird das Installationsverzeichnis bestimmt. Bei einer normalen Windows-Installation steht die Vorgabe auf **C:\Programme\Subversion**.

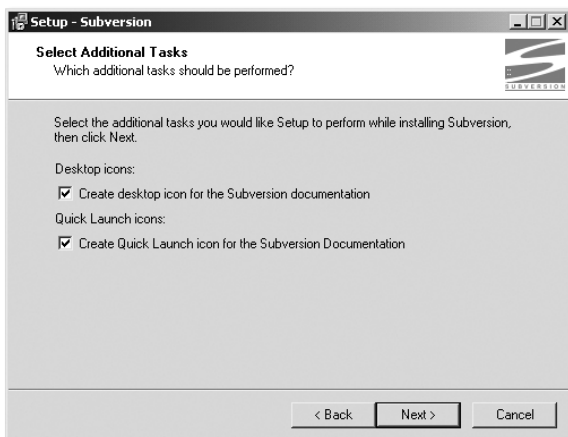
Installations-  
verzeichnis



**Abbildung 4.1** Startdialog des Subversion-Installationsprogramms

Im fünften Schritt wird der Name für den Eintrag in das Startmenü von Windows ausgewählt. Die Vorgabe hier ist »Subversion«. Möchte man keinen Eintrag im Startmenü erzeugen – im Startmenü werden nur Links auf die Dokumentation und den Deinstaller eingetragen, nicht jedoch auf die Programme selbst – so setzt man in diesem Schritt ein Häkchen bei »Don't create a Start Menu Folder«.

Der sechste Schritt erlaubt die Anlage von Verknüpfungen auf die Subversion-Dokumentation auf dem Desktop und in der Schnellstartleiste. Möchte man diese Verknüpfungen nicht anlegen, so deaktiviert man die beiden Auswahlboxen. Abbildung 4.2 zeigt diesen Schritt.



**Abbildung 4.2** Anlage von Verknüpfungen auf die Dokumentation

Im Schritt sieben wird nochmals eine Zusammenfassung der vorzunehmenden Installationsaufgaben angezeigt. Ein Klick auf die Schaltfläche **Install** startet die Installation. Die Installation wird durch einen Fortschrittsbalken angezeigt.

Nachdem alle Dateien installiert worden sind, werden noch einige Informationen zur Installation angezeigt. So werden Windows 9x und ME-Benutzer aufgefordert, die Umgebungsvariable `APR_ICONV_PATH` in der Datei **AUTOEXEC.BAT** auf das Verzeichnis **iconv** der Subversion-Installation zu setzen, also beispielsweise folgendermaßen:

```
SET APR_ICONV_PATH="c:\Programme\Subversion\iconv"
```

Zum Schluss wird noch die Bestätigung angezeigt, dass die Installation beendet worden ist. Ein Neustart ist nicht erforderlich.

Windows 9x

#### 4.1.2 Subversion deinstallieren

Eine mit dem Installationsprogramm angelegte Subversion-Installation lässt sich einfach wieder deinstallieren. Dazu ruft man entweder direkt den im Startmenü eingetragenen Deinstaller auf oder man geht in die Systemsteuerung und wählt unter **Software** den Eintrag Subversion aus. Ein Klick auf die Schaltfläche **Ändern/Entfernen** entfernt Subversion aus dem System. Abbildung 4.3 zeigt diesen Schritt.

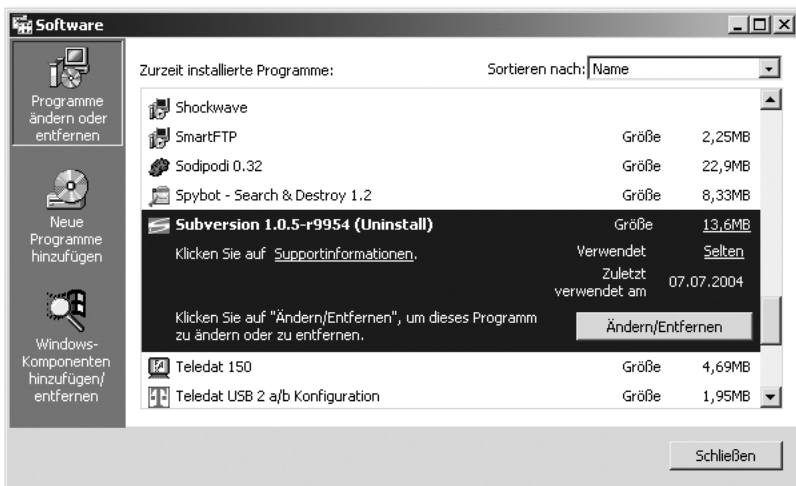
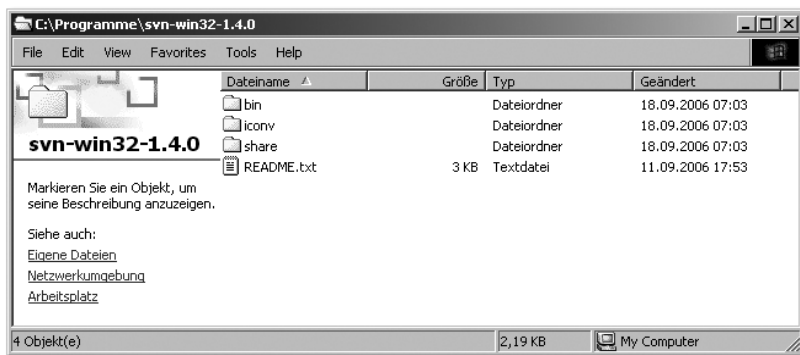


Abbildung 4.3 Subversion deinstallieren

### 4.1.3 Manuelle Installation

Man kann Subversion unter Windows auch recht einfach manuell installieren. Dazu wird Subversion in Form eines ZIP-Archivs ausgeliefert. Das Archiv ist nach dem Schema **svn-win32-<Versionsnummer>.zip** benannt, also beispielsweise **svn-win32-1.4.0.zip** bei Version 1.4.0. Das ZIP-Archiv enthält bereits eine Verzeichnisstruktur, die sich direkt – beispielsweise nach **c:\Programme** – auspacken lässt. Nach dem Auspacken erhält man ein Verzeichnis mit dem Namen **svn-win32-<Versionsnummer>** und den in Abbildung 4.4 gezeigten Unterverzeichnissen.



**Abbildung 4.4** Unterverzeichnisse der Subversion-Installation

Die Datei **README.txt** beschreibt die installierte Version und die verwendeten Bibliotheken. Das Verzeichnis **bin** enthält die ausführbaren Programme und die benötigten DLLs, sowie die Module des Apache Webservers. Im Verzeichnis **iconv** sind allerlei Module der *Apache Portable Runtime (APR)* enthalten, einer betriebssystemunabhängigen Bibliothek auf deren Basis Subversion implementiert worden ist. Der Ordner **share** enthält die Übersetzungen für verschiedene Sprachen.

#### Windows-Path-Variablen

Damit die Subversion-Kommandozeilenprogramme einfach verwendet werden können, sollte man das Unterverzeichnis **bin** in die Umgebungsvariable `PATH` aufnehmen. Auf Windows 2000 und Windows XP geht man dazu in der Systemsteuerung auf das Symbol **System** und klickt dann auf dem Reiter **Erweitert** auf die Schaltfläche **Umgebungsvariablen**. Daraufhin wird das in Abbildung 4.5 gezeigte Dialogfeld geöffnet.

Man wählt im unteren Bereich Systemvariablen den Eintrag **Path** aus und klickt auf die Schaltfläche **Bearbeiten....** Es öffnet sich das in Abbildung 4.6 gezeigte Dialogfenster.

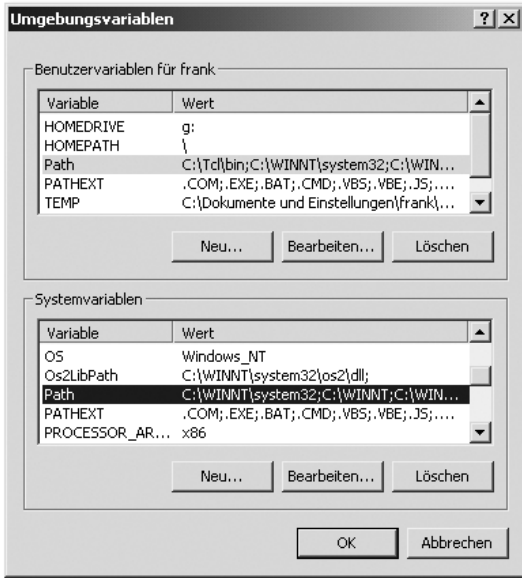


Abbildung 4.5 Änderung der Path-Variablen

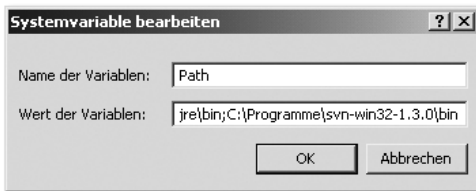


Abbildung 4.6 Pfad für Subversion hinzufügen

Nun geht man an das Ende des Eintrags und fügt dort, durch ein Semikolon abgetrennt, das Verzeichnis an, in dem sich die ausführbaren Subversion-Dateien befinden, also beispielsweise **C:\Programme\svn-win32-1.4.0\bin**. Man sollte beachten, dass man auf Windows NT, Windows 2000 und Windows XP Administratorrechte benötigt, um diese Änderung durchzuführen. Besitzt man keine Administratorrechte, so kann man den Pfad auch für den eigenen Benutzer setzen. Dazu legt man die Benutzervariable **Path** für den eigenen Benutzernamen an (oberes Feld in Abbildung 4.5, Schaltfläche **Neu...**), wenn sie noch nicht vorhanden ist. Man sollte nicht vergessen, an dieser Stelle auf die systemweite Path-Variablen zu verweisen, also beispielsweise folgendermaßen:

```
%Path%;C:\Programme\svn-win32-1.4.0\bin
```

Damit ist Subversion einsatzbereit! Man kann dies testen, indem man in einer Eingabeaufforderung

```
svn help
```

eingibt. Subversion sollte dann eine Liste aller seiner Unterbefehle ausgeben.

## 4.2 Installation unter Debian Linux

Unter Debian Linux lässt sich Subversion komfortabel über die eingebaute Paketverwaltung installieren. Seit Debian 3.1 (Sarge) ist Subversion Teil der normalen Paketquellen und kann einfach per **apt** installiert werden.

Installiert wird Subversion durch den Aufruf von

```
apt-get install subversion
```

Die Paketverwaltung übernimmt die Arbeit

Die Paketverwaltung von Debian installiert alle Pakete, die zur Ausführung von Subversion notwendig sind. Dazu gehören die Apache Portable Runtime (APR), die Berkeley DB und einige andere Bibliotheken. Zur anschließenden Verwendung des Client-Programms **svn** ist keine weitere Konfiguration notwendig!

## 4.3 Installation auf anderen Linux- und Unix-Systemen

Die Installation auf Debian wurde beispielhaft für alle Linux-Distributionen beschrieben. Auf anderen Linux-Derivaten ist meist ein RPM-Paket zu installieren. Dies sollte ähnlich reibungslos funktionieren wie auf Debian. Andere Unix-Systeme wie beispielsweise die BSD-Systeme verwenden wieder andere Paketformate. Daher kann an dieser Stelle keine allgemeingültige Installationsanleitung gegeben werden. Nach der Installation ist keine Konfiguration am Subversion-Client vorzunehmen, er sollte einfach funktionieren. Wer für sein System kein Installationspaket auftreiben kann, der kann Subversion auch aus dem Sourcecode selbst übersetzen. Das Compilieren von Subversion aus dem Sourcecode wird an späterer Stelle in Abschnitt 8.3, »Subversion selbst compilieren«, beschrieben.

## 4.4 Die Programme und Module von Subversion

### 4.4.1 Die Kommandozeilenprogramme

Eine Subversion-Installation besteht aus sechs Kommandozeilenprogrammen:

Sechs Programme

- ▶ **svn** ist das Clientprogramm für den Entwickler. Zur Verwaltung von Sourcecode wird ausschließlich dieses Programm benötigt, alle anderen Programme dienen administrativen Zwecken.
- ▶ **svnadmin** legt Repositories an und verwaltet diese. Das Programm kann nur auf dem Server ausgeführt werden, die Verwaltung von Repositories über ein Netzwerk ist nicht vorgesehen.
- ▶ **svnlook** dient zur Inspektion von Subversion-Repositories. Auch dieses Programm muss direkt auf dem Server ausgeführt werden und kann nicht über ein Netzwerk arbeiten. **svnlook** besitzt nur lesende Optionen und kann ein Repository nicht verändern.
- ▶ **svnversion** gibt die in einer lokalen Arbeitskopie vertretenen Revisionen aus.
- ▶ **svnserve** ist ein kleiner, einfacher Server, der Subversion-Repositories über das Netzwerk zugänglich macht. Der Server verwendet ein simples, proprietäres Protokoll und kann recht einfach konfiguriert werden. Der Server wird in Abschnitt 8.4.3, »svnserve einrichten«, beschrieben.
- ▶ **svndumpfilter** ist ein Filterprogramm, mit dem durch das Programm **svnadmin** erstellte Dumps des Repositories nach bestimmten Kriterien gefiltert werden können. Ein solchermaßen gefilterter Dump kann wiederum in ein Repository importiert werden. Auf diese Weise können Repositories nachträglich um bestimmte Informationen bereinigt werden.
- ▶ **svnsync** ist ein Programm zur Synchronisierung von Repositories. Die Synchronisierung erfolgt immer von einem Quellrepository zu einem Zielrepository. Das Programm kann zum Spiegeln von Repositories verwendet werden. Es wird in Abschnitt 8.8, »Repositories mit dem Programm svnsync kopieren«, beschrieben.

### 4.4.2 Das Apache Modul

Das Apache-Modul **mod\_dav\_svn** dient zum Zugriff auf Subversion-Repositories über den Apache Webserver. Dabei übernimmt Apache sowohl den Transport der Daten über das Netzwerk, wie auch die

Authentifizierung der Benutzer. Zur Übertragung der Daten wird das WebDAV-Protokoll verwendet, eine Erweiterung von HTTP die auch den schreibenden Zugriff erlaubt. Dabei können Übertragungen auch per SSL verschlüsselt werden.

## 4.5 Die Verbindung zum Repository herstellen

Verzeichnis  
auschecken

Um eine Subversion-Installation zu testen, sollte man versuchsweise auf ein bestehendes Repository zugreifen und dort ein Verzeichnis auschecken. Wer von seinem Entwicklungsrechner Zugriff auf das Internet hat, kann für diesen Versuch den Sourcecode von Subversion selbst auschecken. Der Zugriff auf ein Repository wird bei Subversion durch eine URL definiert. Dabei sind verschiedene Zugriffsprotokolle möglich (HTTP, HTTPS, SVN, SVN+SSH, FILE). Das Repository muss bei Befehlen, die nicht aus einer lokalen Arbeitskopie heraus ausgeführt werden, angegeben werden. Dies ist bei den Befehlen **import** und **checkout** immer der Fall. Um den Sourcecode von Subversion auszuchecken gibt man ein:

```
svn checkout http://svn.collab.net/repos/svn/trunk svn
```

Subversion sollte daraufhin eine Ausgabe folgender Art machen:

```
A svn\Makefile.in
A svn\ac-helpers
A svn\ac-helpers\install.sh
A svn\ac-helpers\install-sh
A svn\build.conf
A svn\win-tests.py
A svn\www
A svn\www\release-history.html
A svn\www\testing-goals.html
...
```

Der Vorgang kann gefahrlos abgebrochen werden, indem man `[Strg]+[Pause]` (Windows) oder `[Strg]+[C]` (Unix) drückt oder auf Windows die Eingabeaufforderung schließt (nicht alle Windows-Versionen von Subversion reagieren auf die Eingabe von `[Strg]+[Pause]`). Wer keinen Zugang zum Internet hat, der testet Subversion mittels eines Zugriffs auf den eigenen Server. Für Leser, die keinen Zugriff auf einen eigenen Subversion-Server haben, wird in Anhang A der lokale Zugriff auf ein Repository als Alternative beschrieben. Wer statt dessen selbst einen Subversion-Server betreiben möchte, findet dazu eine ausführliche Anleitung in Abschnitt 8.1, »Einen Subversion-Server aufsetzen«.

## 4.6 Zusammenfassung

Nach den theoretischen Überlegungen zum Entwicklungszyklus mit Subversion und der Installation der Software ist es nun an der Zeit, erste praktische Schritte mit dem System durchzuführen. Das nachfolgende Kapitel startet mit einer einfachen Beispielsitzung.

# Index

\$HeadURL\$ 184  
\$LastChangedBy\$ 184  
\$LastChangedDate\$ 184  
\$LastChangedRevision\$ 184  
.bash\_profile 150  
.svn 71, 146, 178, 236

## A

---

Abstammung 129, 136, 353  
Access Controll Lists 254  
add 117, 258  
    *Erkennung binärer Formate* 118  
Administration 21  
Ancestry 129, 136, 353  
annotate 157  
Apache Group 353  
Apache Portable Runtime 41, 62, 197,  
    200, 236, 353  
Apache Webserver 40, 198, 201, 210,  
    236  
APPDATA 150, 337  
application/octet-stream 94, 118, 162  
APR\_ICONV\_PATH 61  
APR-Util 201  
AT&T 31  
atomar 353  
Attic 235  
Auschecken 353  
Ausführungsattribut 163  
auth (Verzeichnis) 152, 336  
Authentifizierung 70  
    *Basic HTTP* 212  
AUTOEXEC.BAT 61  
Autorisierung 214

## B

---

Backup 29, 228  
BASE 82, 105, 353  
Befehlsaufbau 257  
Beispieldateien 346  
Berkeley DB 40, 201, 202, 236, 353  
Berkeley DB Repository  
    *restaurieren* 218

Binärdatei 353  
binäre Dateien 243  
Bitkeeper 33  
blame 156, 259  
Branch 26, 354  
branches 86, 128

## C

---

Carriage Return 166  
cat 116, 261  
changed 222  
Checkin 354  
Checkout 354  
checkout 69, 95, 262  
    *Optionen* 96  
Checkout (Vorgang) 46  
cleanup 141, 263  
ClearCase 33, 253  
Cobol 31  
commit 72, 102, 264, 354  
    *atomar* 104, 239  
    *Out of Date* 102  
Commit (Vorgang) 49  
COMMITTED 82, 354  
config (Datei) 93, 152, 337  
copy 119, 266  
    *Tags und Verzweigungen* 121  
CR 167  
CRLF 167  
CVS 15, 21, 32, 33, 40, 235, 253, 354  
    *Entwicklungsmodell* 15  
    *Macken* 15  
cvs2svn 247  
CVSNT 248  
CVSROOT 241

## D

---

Datei  
    *analysieren* 156  
    *ignorieren* 92  
delete 119, 268  
DeltaV 357  
diff 104, 270

dirs-changed 222  
DOS 166  
dump 229  
Dump-Datei 229  
Dumps  
    *inkrementell* 231

## E

---

Eclipse 36  
Edit-Compile-Run-Zyklus 44  
Einchecken 49, 354  
entries 178  
Entwicklungszyklus  
    *mehrere Entwickler* 55  
Export 354  
export 148, 273  
Externals 171, 354

## F

---

FAT 202  
Fortran 31  
FSFS 202, 354

## G

---

Gemischte Revisionen 82  
GIT 33  
global-ignores 258  
GNU Autoconf 201

## H

---

Hauptentwicklungslinie 86  
Hauptentwicklungszweig 69  
Hauptzweig 354  
HEAD 82, 105, 354  
HeadURL 185  
help 88, 257, 274  
history 221  
Hooks  
    *JScript* 225  
    *VBScript* 225  
Hook-Skripte 222  
    *Windows* 225  
hot-backup.py 232  
hotcopy 232  
HTML 23, 176

HTTP 201, 210  
httpd.conf 211  
HTTP-Proxy 340  
HTTPS 201

## I

---

IBM 36  
Id 185  
Import 354  
import 92, 275  
info 158, 277  
Installation  
    *Debian Linux* 64  
    *Windows 9x* 61  
Installationsprogramm  
    *Windows* 59  
ISO 8601 176

## J

---

Java 23, 36, 247  
Journaling File Systems 141

## K

---

Kommunikation 350  
Komposition 171  
Konflikt 47, 76, 98, 100, 134, 355  
Konfliktmarker 47, 77, 98, 100, 355  
Konventionen 24

## L

---

LANG 156  
LastChangedBy 185  
LastChangedDate 185  
LastChangedRevision 185  
LaTeX 23  
Laufzeitkonfigurationsbereich 150  
LDAP 212  
LF 167  
Linefeed 166  
Linux 24, 35, 197, 335  
list 113, 278  
load 230  
Lock 355  
lock 279  
Log Message 72, 92, 103, 110, 281, 355

lokale Arbeitskopie 355  
*umschalten* 144  
*Verwaltungsinformationen* 146

## M

---

MacOSX 23, 41, 197, 335  
 Master-Kopie 43  
 merge 131, 284  
 Merging 47, 99, 177, 355  
 Metadaten 158  
 Migration 228  
 MIME-Typ 163, 244, 355  
 MIME-Typs 94  
 mkdir 123, 286  
 mod\_authz\_svn.so 211  
 mod\_dav.so 211  
 mod\_dav\_svn 65  
 mod\_dav\_svn.so 211  
 move 121, 288  
 Murphys Law 351

## N

---

native 167  
 Neon 201  
 Netzwerklaufwerk 202  
 NTFS 202

## O

---

Ocaml 247  
 Onlinehilfe 88  
 Open Source 15, 32, 40, 173, 254  
 OpenSSH 207  
 OpenSSL 201  
 OS/2 166

## P

---

passwd (Datei) 205  
 PATH 62  
 Perforce 33  
 Perl 247  
 PHP 181, 247  
 post-commit 223  
 post-lock 224  
 post-revprop-change 223  
 post-unlock 224

praise 157  
 pre-commit 223  
 pre-lock 224  
 pre-revprop-change 223, 226  
 pre-unlock 224  
 PREV 82, 355  
 Pristine Copy 147, 355  
 Projektleiter 21  
 propdel 290  
 propedit 291  
 Properties 158  
   *automatisches Setzen* 168  
   *binäre Werte* 161  
   *Konflikte* 161  
   *revisionsbezogen* 169  
 Property 356  
 propget 292  
 proplist 294  
 propset 295  
 Python 247

## R

---

RapidSVN 35  
 RCS 31, 235  
 recover 219  
 Referenzkarte 21  
 Rekursion  
   *abschalten* 91  
 Repository 28, 356  
   *Abgleich* 29  
   *anlegen* 203  
   *lokaler Zugriff* 345  
 Repository Layout 85  
 Repository-Browser 87  
 resolved 78, 101, 297  
 revert 123, 298  
 Revision 81, 356  
 Revisionsnummer 81, 237, 356  
 Revisions Schlüsselwörter 82  
 Rollback 356  
 RPM-Paket 64  
 RSH 208  
 RSS 182, 227  
 RSS-Feed 356  
 Ruby 247

## S

---

- Sandbox 356
- SCCS 31
- Schlüsselwortersetzung 166, 183, 244, 356
  - abschalten* 184
- servers (Datei) 153, 340
- setlog 220
- Softwareentwickler 21
- Sourcecode-Distribution 30
- SQL 254
- SSH 198, 207, 356
- SSL 66, 154, 198, 216
- SSL-Zertifikat 340
- start-commit 223
- status 72, 106, 141, 299
  - svn:ignore* 164
  - Zeichencodes* 108
- Subclipse 36
- Subversion
  - Alternativen* 33
  - Arbeitsweise* 29
  - Architektur* 38
  - Betriebssysteme* 41
  - Buchstabencodes* 71, 72
  - Client-Server* 34
  - Code-Distribution* 44
  - compilieren* 199
  - Dateityp* 94
  - Datums- und Zeitangaben* 176
  - deinstallieren* 61
  - einzelner Entwickler* 43
  - Entwicklungszyklus* 44
  - erster Kontakt* 44
  - Geschichte* 31
  - GUI-Clients* 34
  - Homepage* 359
  - Kommandozeilenprogramme* 65
  - Kommunikation* 56
  - Lizenz* 40, 59
  - Plattformunabhängigkeit* 23
  - Regeln* 56, 349
  - URL* 69
  - Version* 21
  - Versionsverwaltung* 30
  - Verzeichnispfade* 24
  - Webfrontends* 179
  - Webseiten* 177
  - Zugriffsmethoden* 197
  - Zugriffsverfahren* 39
- Subversion-Befehle
  - Arbeitsweisen* 84
- Subversion-Client 28
- Subversion-Server 28
- Subversive 36
- svn 65, 258
  - author* 170
  - Befehle abzukürzen* 143
  - date* 170
  - eol-style* 166
  - executable* 163
  - externals* 167, 171
  - ignore* 93, 94, 118, 164, 258
  - Implizite Argumente* 90
  - keywords* 166, 184, 244
  - log* 170
  - mime-type* 94, 118, 162
  - needs-lock* 168
  - Rekursion* 91
  - special* 167
- SVN (Protokoll) 70, 198
- SVN\_EDITOR 103, 149
- svn\_load\_dirs.pl 176
- svnadmin 65, 305
  - create* 305
  - deltify* 306
  - dump* 306
  - help* 308
  - hotcopy* 308
  - list-dblogs* 309
  - list-unused-dblogs* 309
  - load* 309
  - ltxns* 311
  - recover* 311
  - rmtxns* 312
  - setlog* 313
  - verify* 313
- svndumpfilter 65, 314
  - exclude* 314
  - help* 315
  - include* 316
- svnlook 65, 220, 317
  - author* 317
  - cat* 318
  - changed* 319
  - date* 319
  - diff* 320
  - dirs-changed* 321
  - help* 322

- history* 322
- info* 323
- log* 324
- propget* 325
- proplist* 325
- tree* 326
- uuid* 327
- youngest* 328
- svnserve 65, 204, 328
  - Firewalls* 204
- svnserve (Protokoll) 198
- svnserve.conf 204
- svnsync 329
  - help* 330
- svnversion 65, 332
- SWIG 247
- switch 144, 301
- Synchronisation 55

## T

---

- Tag 86, 125, 241, 356
- tags (Verzeichnis) 86
- text/plain 162
- text-base 147
- TextPrinter1 69
- TortoiseSVN 34, 359
- Transaktion 104, 357
- trunk 86

## U

---

- überwachtes Arbeiten 244
- Universal Unique Identifier 222, 327
- Unix 23, 41, 166
  - symbolische Links* 170
- unlock 303
- Unterverzweigung 137
- Update 357
- update 72, 304
  - Mögliche Fälle* 97
  - Optionen* 101
- Update (Vorgang) 46
- UUID 357
- uuid 222

## V

---

- Vendor Branches 173, 245

- verify 219
- Versionshistorie
  - manuelle* 26
- Versionsmanagement 357
  - wofür?* 25
- Versionsmanagementsystem 28, 357
- Verzweigung 26, 87, 127, 241, 357
  - Einsatzbereiche* 138
  - Gründe* 127
  - Tipps* 138
  - Zusammenführung* 131
- ViewCVS 179
- ViewVC 179
- Visual C++ 201
- Visual Source Safe 33, 253

## W

---

- WebDAV 66, 210, 357
- WebSVN 181
- Wedging 202
- WinCvs 246
- Windows 23, 41, 166, 197
- Windows 9x 202
- Windows ME 202
- Windows XP 62, 63
- Windows\_2000 62, 63
- Windows\_NT 63
- Windows-Registry 155
- www.subversionbuch.de 359

## X

---

- XML 147, 176

## Y

---

- youngest 221

## Z

---

- Zielgruppe 21
- zlib 201
- Zugriffsverfahren
  - Apache* 198
  - lokal* 198
  - SVN* 198