Versionskontrolle mit Subversion

Für Subversion 1.6

(Übersetzt aus der Revision 4919)

Ben Collins-Sussman
Brian W. Fitzpatrick
C. Michael Pilato

Versionskontrolle mit Subversion: Für Subversion 1.6: (Übersetzt aus der Revision 4919)

von Ben Collins-Sussman, Brian W. Fitzpatrick und C. Michael Pilato

Veröffentlicht (TBA)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Beispiele

Geleitwort

Karl Fogel
Chicago, 14, März 2004.

Eine schlechte FAQ (Frequently Asked Questions) ist eine, die nicht aus den Fragen besteht, die wirklich gefragt wurden, sondern aus denen, die der Autor sich von den Fragenden *gewünscht* hätte. Vielleicht haben Sie solche schon gesehen:

> F: Wie kann ich Glorbosoft XYZ einsetzen, um die Team-Produktivität zu maximieren?
>
> A: Viele unserer Kunden wollen wissen, wie sie Ihre Produktivität mit unseren patentierten Office Groupware Innovationen maximieren können. Die Antwort ist einfach: zuerst klicken Sie auf das Menü „`Datei`“, fahren hinunter zum Eintrag „`Erhöhe Produktivität`“, und dann …

Das Problem mit solchen FAQs ist, dass sie keine FAQs im eigentlichen Sinne sind. Niemand fragt den technischen Support: „Wie können wir unsere Produktivität steigern?“ Üblicherweise fragen Leute sehr spezifische Fragen, wie: „Wie können wir das Kalendersystem so ändern, dass es die Erinnerungen zwei Tage statt einen Tag im Voraus aussendet?“ und so weiter. Aber es ist viel leichter, häufig gestellte Fragen zu erfinden, als die richtigen Fragen zu entdecken. Eine sinnvolle FAQ-Sammlung zusammenzustellen, erfordert eine ausdauernde, planvolle Anstrengung: über die Lebensdauer einer Software müssen hereinkommende Anfragen ausgewertet und Rückmeldungen evaluiert werden und zu einem konsistenten und benutzerfreundlichen Ganzen zusammengeführt werden, das die gesammelte Erfahrung der Anwendenden wiedergibt. Es erfordert die geduldige, aufmerksame Einstellung eines Naturforschers. Nicht großartige Hypothesen und visionäre Vorhersagen, sondern hauptsächlich offene Augen und genaue Aufzeichnungen sind gefragt.

Was ich an diesem Buch liebe, ist, dass es genau aus einem solchen Prozess gewachsen ist und dies auf jeder Seite sichtbar ist. Es ist das direkte Ergebnis der Begegnungen der Autoren mit Benutzern. Es begann mit Ben Collins-Sussmans Beobachtung, dass Leute immer wieder die gleichen grundlegenden Fragen auf der Subversion-Mailingliste stellten: Was sind die Standard-Arbeitsabläufe mit Subversion? Funktionieren Branches und Tags genau so wie in anderen Versionskontrollsystemen? Wie finde ich heraus, wer eine bestimmte Änderung durchgeführt hat?

Frustriert davon, Tag für Tag immer wieder die gleichen Fragen zu sehen, arbeitete Ben im Sommer 2002 über einen Monat intensiv daran, *The Subversion Handbook* zu schreiben, eine 60-seitige Anleitung, die die Grundlagen der Benutzung von Subversion beschrieb. Die Anleitung erhob keinen Anspruch auf Vollständigkeit, aber sie wurde mit Subversion verteilt und half vielen über die ersten Buckel der Lernkurve. Als O'Reilly and Associates sich entschieden, ein vollständiges Buch über Subversion herauszugeben, war der Weg des geringsten Widerstandes offensichtlich: *The Subversion Handbook* muss erweitert werden.

Die drei Co-Autoren des neuen Buches erhielten somit eine seltene Gelegenheit. Eigentlich war es ihre Aufgabe, ein Buch beginnend mit dem Inhaltsverzeichnis und einem Rohkonzept zu schreiben; jedoch hatten sie auch Zugang zu einem ständigen Strom – ja einem unkontrollierbaren Geysir – aus Quellmaterial. Subversion wurde bereits von tausenden experimentierfreudigen Menschen benutzt, und diese gaben Unmengen an Rückmeldungen – nicht nur über Subversion, sondern auch über die bestehende Dokumentation.

Während der gesamten Zeit, in der sie dieses Buch schrieben, durchstöberten Ben, Mike und Brian unablässig die Subversion-Mailinglisten und Chaträume und notierten die Probleme, die Benutzer im echten Leben hatten. Die Beobachtung derartiger Rückmeldungen war ohnehin ein Teil ihrer Arbeit bei CollabNet, was ihnen einen Riesenvorteil verschaffte, als sie sich entschlossen, Subversion zu dokumentieren. Das Buch, das sie schrieben, gründet auf dem festen Fels der Erfahrung und nicht auf dem Treibsand des Wunschdenkens. Es vereint die Vorteile von Bedienungsanleitung und FAQ. Diese Zweigleisigkeit ist vielleicht nicht gleich zu erkennen. Von vorne nach hinten gelesen ist das Buch einfach eine Beschreibung einer Software. Es gibt die Übersicht, den obligatorischen Rundgang, das Kapitel über Administration, einige fortgeschrittene Themen und natürlich eine Funktionsübersicht sowie eine Anleitung zur Problemlösung. Erst wenn Sie es später wieder zur Hand nehmen, um die Lösung für ein bestimmtes Problem zu suchen, wird die Zuverlässigkeit des Buches offenbar: in den beschriebenen Details,die nur aus der Erfahrung mit dem Unerwarteten erwachsen konnten, in den Beispielen, die aus dem tatsächlichem Einsatz gebildet wurden, und am meisten durch das Gefühl für die Bedürfnisse und den Blickwinkel der Benutzer.

Natürlich kann niemand versprechen, dass dieses Buch alle Fragen beantwortet, die Sie über Subversion haben. Manchmal wird die Genauigkeit, mit der es Ihre Fragen erwartet, unheimlich und telepathisch erscheinen; gelegentlich werden Sie jedoch in ein Loch im Wissen der Gemeinschaft stolpern und mit leeren Händen dastehen. Wenn das passiert schreiben Sie am besten eine E-Mail an `<users@subversion.apache.org>` und schildern Ihr Problem. Die Autoren sind nach wie vor dort und beobachten. Und das betrifft nicht nur die drei, die auf dem Umschlag erscheinen sind, sondern viele andere, die Korrekturen

und neues Material beigesteuert haben. Aus der Sicht der Gemeinschaft ist die Lösung Ihres Problems lediglich ein erfreulicher Nebeneffekt eines viel größeren Projektes – nämlich das Buch und schlussendlich auch Subversion selbst immer näher an die Art und Weise anzupassen, in der es tatsächlich benutzt wird. Diese Personen sind begierig darauf, von Ihnen zu hören, nicht nur weil sie Ihnen helfen können, sondern auch weil ihnen selbst damit geholfen ist. Für Subversion – so wie für alle aktiven freien Software-Projekte – gilt: *Sie sind nicht allein*.

Lassen Sie dieses Buch Ihren ersten Begleiter sein.

Vorwort

> „Es ist wichtig, die Vollkommenheit nicht zum Feind des Guten werden zu lassen, selbst dann, wenn darüber Einigkeit besteht, was Vollkommenheit ist. Erst recht, wenn man sich nicht darüber einig ist. So unangenehm es ist, durch vergangene Fehler gefangen zu sein, kann man während des Entwurfs keinen Fortschritt erzielen, wenn man Angst vor dem eigenen Schatten hat."
>
> —Greg Hudson, Subversion-Entwickler

In der Welt der Open-Source-Software war das Concurrent Versions System (CVS) für viele Jahre das Werkzeug der Wahl für Versionskontrolle. Und das zu Recht. CVS war selbst Open-Source-Software und seine nicht-einschränkende Vorgehensweise und Unterstützung für netzbasierten Einsatz erlaubte dutzenden geografisch verteilten Programmierern, ihre Arbeit zu teilen. Es passte sehr gut zur kollaborativen Natur der Open-Source-Welt. CVS und sein halb chaotisches Entwicklungsmodell sind seitdem zu Eckpfeilern der Open-Source-Kultur geworden.

Jedoch war CVS nicht makellos, und diese Makel einfach zu beseitigen, versprach einen enormen Aufwand. Bühne frei für Subversion. Subversion wurde als Nachfolger für CVS entworfen, und seine Schöpfer zogen los, um auf zwei Wegen die Herzen der CVS-Benutzer zu gewinnen – indem sie ein Open-Source-System erschufen, dessen Design (und „look and feel") ähnlich wie CVS war, wobei sie versuchten, die auffälligsten Makel von CVS zu vermeiden. Das Ergebnis war, und ist, zwar nicht der nächste Evolutionsschritt in Sachen Versionskontrolle, dennoch *ist* Subversion sehr mächtig, sehr brauchbar und sehr flexibel.

Dieses Buch ist geschrieben worden, um die Serie 1.6 des Subversion™[1] Versionskontrollsystems zu dokumentieren. Wir haben stets versucht, die Themen gründlich zu behandeln. Jedoch hat Subversion eine florierende und tatkräftige Entwicklergemeinde, so dass bereits eine Menge an Features und Verbesserungen für künftige Versionen von Subversion geplant sind, die Änderungen mancher Kommandos und bestimmter Anmerkungen in diesem Buch bewirken könnten.

Was ist Subversion?

Subversion ist ein freies/Open-Source *Versionskontrollsystem* (VCS). Das bedeutet, Subversion verwaltet Dateien und Verzeichnisse und die Änderungen an ihnen im Lauf der Zeit. Das erlaubt Ihnen, alte Versionen Ihrer Daten wiederherzustellen oder die Geschichte der Änderungen zu verfolgen. Unter diesem Blickwinkel denken viele Leute bei einem Versionskontrollsystem an eine Art „Zeitmaschine".

Subversion kann netzwerkübergreifend arbeiten, was die Benutzung durch Menschen an verschiedenen Computern ermöglicht. Auf einer bestimmten Ebene fördert die Fähigkeit, unterschiedlicher Personen dieselbe Menge an Daten bearbeiten und verwalten zu können, die Zusammenarbeit. Ohne auf einen einzigen Kanal beschränkt zu sein, über den alle Änderungen abgewickelt werden müssen, kann das Vorankommen beschleunigt werden. Und weil die Arbeit versioniert ist, braucht nicht befürchtet zu werden, dass die Qualität bei Verlust dieses Kanals geopfert wird – falls irgendeine falsche Änderung an den Daten gemacht wird, kann man sie einfach zurücknehmen.

Manche Versionskontrollsysteme sind auch *Software-Konfigurationsmanagement-Systeme* (SCM). Diese Systeme sind maßgeschneidert, um ganze Verzeichnisbäume mit Quellcode zu verwalten und verfügen über viele Merkmale, die spezifisch für Software-Entwicklung sind – etwa das Verstehen von Programmiersprachen oder das Bereitstellen von Werkzeugen zum Bauen von Software. Jedoch gehört Subversion nicht zu diesen Systemen. Es ist ein allgemeines System, das verwendet werden kann, um *alle möglichen* Sammlungen von Dateien zu verwalten. Für Sie mag es sich dabei um Quellcode handeln – für andere mag es dabei um alles von Einkaufslisten bis zu digitalen Videomischungen und weit darüber hinaus gehen.

Ist Subversion das richtige Werkzeug?

Falls Sie ein Anwender oder Systemadministrator sind und den Einsatz von Subversion erwägen, sollte die erste Frage, die Sie sich stellen, sein: "Ist es das richtige Werkzeug für die Aufgabe?" Subversion ist ein fantastischer Hammer, achten Sie jedoch darauf, dass Sie nicht jedes Problem als einen Nagel sehen.

Falls Sie alte Datei- und Verzeichnisversionen aufbewahren, sie eventuell wiedererwecken müssen, oder Protokolle darüber auswerten möchten, wie sie sich im Lauf der Zeit geändert haben, ist Subversion das genau passende Werkzeug für Sie.

[1] In diesem Buch werden wir es einfach „Subversion" nennen. Sie werden uns dankbar sein, sobald Sie feststellen, wieviel Platz das spart!

Subversion ist auch geeignet, wenn Sie mit mehreren Leuten gemeinsam (üblicherweise über das Netz) an Dokumenten arbeiten und verfolgen müssen, wer welche Änderung gemacht hat. Deshalb wird Subversion so oft in Softwareentwicklungsumgebungen eingesetzt – die Arbeit in einem Entwicklerteam ist von Natur aus eine soziale Tätigkeit und Subversion vereinfacht die Zusammenarbeit mit anderen Programmierern. Natürlich ist die Benutzung von Subversion nicht umsonst zu bekommen: es kostet administrativen Aufwand. Sie müssen ein Daten-Projektarchiv verwalten, das die Informationen und ihre gesamte Geschichte speichert, und Sie müssen sich gewissenhaft um Sicherheitskopien kümmern. Wenn Sie täglich mit den Daten arbeiten, werden Sie sie nicht auf die gleiche Art kopieren, verschieben, umbenennen oder löschen können wie gewohnt. Stattdessen müssen Sie dafür Subversion verwenden.

Unter der Annahme, dass Ihnen die zusätzlichen Arbeitsabläufe nichts ausmachen, sollten Sie trotzdem sicher sein, dass Sie Subversion nicht für die Lösung eines Problems verwenden, das andere Werkzeuge besser lösen könnten. Zum Beispiel wird Subversion, weil es die Daten an alle Beteiligten verteilt, als generisches Verteilsystem missbraucht. Manchmal wird Subversion zum Verteilen von umfangreichen Bildersammlungen, digitaler Musik oder Softwarepaketen verwendet. Das Problem damit ist, dass sich diese Art Daten für gewöhnlich überhaupt nicht verändert. Die Sammlung selber wächst stetig, jedoch werden die einzelnen Dateien der Sammlung nicht verändert. In diesem Fall ist die Benutzung von Subversion zu viel des Guten.[2] Es gibt einfachere Werkzeuge, die hervorragend Daten replizieren, *ohne* dabei Änderungen mitzuverfolgen, etwa **rsync** oder **unison**.

Die Geschichte von Subversion

Anfang 2000 begann CollabNet, Inc. (http://www.collab.net) Entwickler zu suchen, die einen Ersatz für CVS schreiben sollten. CollabNet bot[3] eine Software-Suite namens CollabNet Enterprise Edition (CEE) für die Zusammenarbeit an, die auch eine Komponente für Versionskontrolle beinhaltete. Obwohl CEE ursprünglich CVS als Versionskontrollsystem verwendete, waren die Einschränkungen von CVS von Anfang an offensichtlich, und CollabNet war sich bewusst, dass letztendlich etwas Besseres gefunden werden musste. Unglücklicherweise war CVS der de-facto Standard in der Open-Source-Welt geworden, hauptsächlich deshalb, weil es nichts Besseres gab, zumindest nicht unter einer freien Lizenz. Also beschloss CollabNet, ein vollständig neues Versionskontrollsystem zu schreiben, welches die grundlegenden Ideen von CVS beibehalten, jedoch die Fehler und Fehlentwicklungen vermeiden sollte.

Im Februar 2000 nahmen sie Verbindung mit Karl Fogel auf, dem Autor von *Open Source Development with CVS* (Coriolis, 1999), und fragten ihn, ob er an diesem neuen Projekt mitarbeiten wolle. Zufälligerweise besprach Karl bereits einen Entwurf für ein neues Versionskontrollsystem mit seinem Freund Jim Blandy. Im Jahr 1995 gründeten die beiden Cyclic Software, eine CVS-Beratungsfirma, und sie benutzten, obwohl sie die Firma später verkauften, bei ihrer täglichen Arbeit immer noch CVS. Ihre Enttäuschung über CVS veranlasste Jim, sorgfältig über bessere Möglichkeiten zur Verwaltung versionierter Daten nachzudenken. Er hatte sich nicht nur bereits den Subversion-Namen ausgedacht, sondern auch den grundsätzlichen Entwurf der Subversion-Datenablage. Als CollabNet rief, stimmte Karl sofort der Mitarbeit am Projekt zu, und Karl gelang es, dass sein Arbeitgeber Red Hat Software ihn praktisch auf unbestimmte Zeit dem Projekt spendete. CollabNet stellte Karl und Ben Collins-Sussman ein und der detaillierte Entwurfsprozess begann im Mai. Dank einiger Stupser von Brian Behrendorf und Jason Robbins von CollabNet sowie Greg Stein (zu dieser Zeit als unabhängiger Entwickler aktiv im der WebDAV/DeltaV Spezifikationsprozess), zog Subversion schnell eine Gemeinde aktiver Entwickler an. Es stellte sich heraus, dass viele Leute dieselben enttäuschenden Erfahrungen mit CVS gemacht hatten und nun die Gelegenheit begrüßten, etwas daran zu ändern.

Das ursprüngliche Designteam einigte sich auf einige einfache Ziele. Sie wollten kein Neuland in Versionskontrollmethodik betreten, sondern einfach CVS reparieren. Sie beschlossen, dass Subversion dieselben Merkmale und dasselbe Entwicklungsmodell wie CVS haben solle, wobei die Fehler von CVS aber nicht noch einmal gemacht werden sollten. Und obwohl es nicht als ein hundertprozentiger Ersatz für CVS gedacht war, sollte es dennoch ähnlich genug sein, so dass ein leichter Wechsel für einen CVS-Anwender möglich wäre.

Nach vierzehn Monaten Programmierung wurde Subversion am 31. August 2001 „selbstbewirtend“, d.h., die Subversion-Entwickler hörten auf, CVS für den Quellcode von Subversion zu verwenden und benutzten stattdessen Subversion.

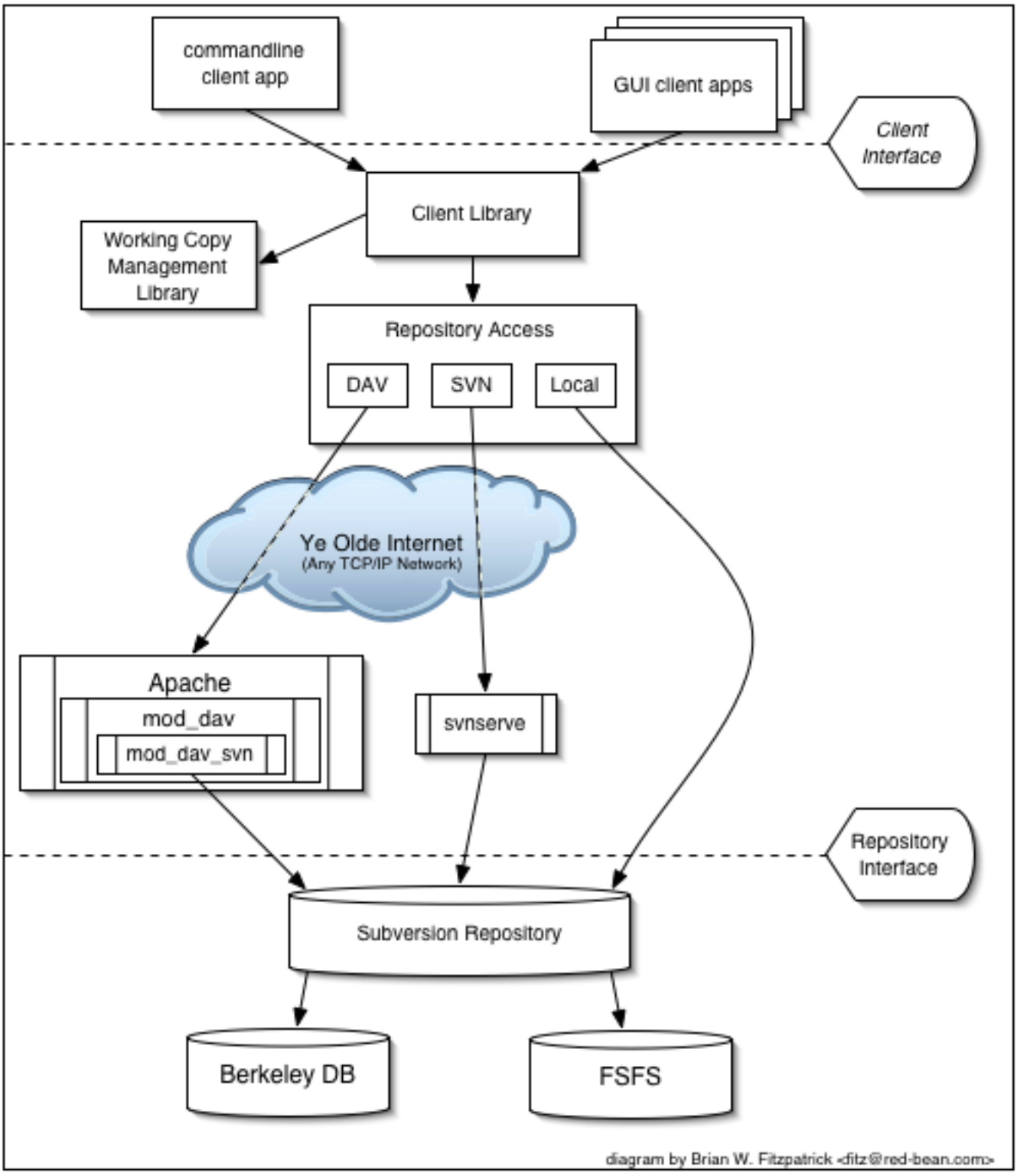
Obwohl CollabNet das Projekt startete und immer noch einen großen Batzen der Arbeit finanziert (sie zahlen die Gehälter einiger Vollzeit-Subversion-Entwickler), läuft Subversion wie die meisten Open-Source-Projekte, geführt von einer Anzahl lockerer, transparenter Regeln, die die Meritokratie fördern. Im Jahr 2009 arbeitete CollabNet mit den Subversion-Entwicklern auf das Ziel hin, das Subversion-Projekt in die Apache Software Foundation (ASF) zu integrieren, eine der bekanntesten Kollektiven für Open-Source-Projekte auf der Welt. Subversions technische Wurzeln, Gemeinschaftswerte und Entwicklungspraktiken passten perfekt zur ASF, von deren Mitgliedern viele bereits aktiv an Subversion mitgewirkt haben. Anfang 2010 war Subversion vollständig in die Familie der wichtigsten ASF Projekte aufgenommen, verlegte seine Webpräsenz nach http://subversion.apache.org und wurde in „Apache Subversion“ umbenannt.

[2] Oder wie es ein Freund ausdrückt: „Eine Fliege mit einem Buick erschlagen.“
[3] CollabNet Enterprise Edition ist seitdem durch eine neue Produktlinie namens CollabNet TeamForge ersetzt worden.

Die Architektur von Subversion

Abbildung 1, „Die Architektur von Subversion“ illustriert einen „kilometerhohen“ Blick auf das Design von Subversion.

Abbildung 1. Die Architektur von Subversion



An einem Ende ist das Projektarchiv von Subversion, das die gesamten versionierten Daten enthält. Am anderen Ende ist Ihr Subversion-Client-Programm, das die lokale Spiegelung von Teilen dieser versionierten Daten verwaltet. Zwischen den entgegengesetzten Enden befinden sich mehrere Wege über die Projektarchiv-Zugriffsschicht (RA-Schicht). Einige davon gehen über Computernetzwerke und über Netzwerkserver, die dann auf das Projektarchiv zugreifen. Andere wiederum lassen das Netz links liegen und greifen direkt auf das Projektarchiv zu.

Die Komponenten von Subversion

Sobald es installiert ist, hat Subversion eine Anzahl verschiedener Teile. Es folgt ein schneller Überblicki über das, was Sie bekommen. Lassen Sie sich nicht beunruhigen, sollten die kurzen Beschreibungen Sie dazu veranlassen, sich am Kopf zu kratzen – es gibt in diesem Buch *eine Menge* weiterer Seiten, die dem Ziel gewidmet sind, diese Verwirrung zu lindern.

svn
: Das Kommandozeilenprogramm

svnversion
: Ein Programm, das den Zustand einer Arbeitskopie (durch Revisionen der vorliegenden Objekte) berichtet

svnlook
: Ein Werkzeug zur direkten Untersuchung eines Subversion-Projektarchivs

svnadmin
: Ein Werkzeug zum Erstellen, Verändern oder Reparieren eines Projektarchivs

mod_dav_svn
: Ein Plug-In-Modul für den Apache-HTTP-Server, wird benötigt, um das Projektarchiv über ein Netzwerk verfügbar zu machen

svnserve
: Ein spezielles Server-Programm, dass als Hintergrundprozess laufen oder von SSH aufgerufen werden kann; eine weitere Möglichkeit, das Projektarchiv über ein Netzwerk verfügbar zu machen

svndumpfilter
: Ein Programm zum Filtern von Subversion-Projektarchiv-Dump-Streams

svnsync
: Ein Programm zum inkrementellen Spiegeln eines Projektarchivs über ein Netzwerk

Was gibt es Neues in Subversion

Die erste Auflage dieses Buchs wurde von O'Reilly Media im Jahr 2004 veröffentlicht, kurz nachdem Subversion die 1.0 erreicht hatte. Seitdem hat das Subversion-Projekt weiterhin neue Hauptversionen der Software herausgegeben. Hier ist eine kurze Zusammenfassung der umfangreicheren Änderungen seit Subversion 1.0. Beachten Sie, dass es keine komplette Liste ist; um alle Details zu sehen, besuchen Sie die Subversion-Website bei http://subversion.apache.org.

Subversion 1.1 (September 2004)
: Release 1.1 führte FSFS ein, eine Projektarchiv-Speicheroption, die auf Dateien basiert. Obwohl das Berkeley-DB-Backend immer noch weitverbreitet ist und unterstützt wird, ist FSFS mittlerweile wegen der niedrigen Einstiegshürde und des minimalen Wartungsbedarfs die Standard-Auswahl für neu erzeugte Projektarchivs. Ebenfalls kam mit diesem Release die Möglichkeit, symbolische Links unter Versionskontrolle zu stellen, das automatische Maskieren von URLs und eine sprachabhängige Benutzerschnittstelle.

Subversion 1.2 (Mai 2005)
: Mit Release 1.2 konnten serverseitige Sperren auf Dateien erzeugt und somit der Commit-Zugriff für bestimmte Ressourcen serialisiert werden. Während Subversion immer noch grundsätzlich ein gleichzeitiges Versionskontrollsystem ist, können bestimmte Arten binärer Dateien (z.B. Kunstobjekte) nicht zusammengeführt werden. Die Sperrmöglichkeit stillt den Bedarf, solche Ressourcen zu versionieren und zu schützen. Zusammen mit dem Sperren kam auch eine vollständige WebDAV-Auto-Versionierungs-Implementierung, die es erlaubt, Subversion-Projektarchivs als Netzwerkverzeichnisse einzuhängen. Schließlich begann Subversion 1.2 einen neuen, schnelleren binären

Differenzalgorithmus zu verwenden, um alte Versionen von Dateien zu komprimieren und hervorzuholen.

Subversion 1.3 (Dezember 2005)
: Release 1.3 brachte pfadbasierte Autorisierungskontrolle für den **svnserve**-Server, was einem Merkmal entsprach, das vorher nur im Apache-Server vorzufinden war. Der Apache-Server wiederum bekam einige neue eigene Logging-Features, und die Subversion-API-Bindings für andere Sprachen machten auch große Sprünge vorwärts.

Subversion 1.4 (September 2006)
: Release 1.4 führte ein völlig neues Werkzeug – **svnsync** – ein, um eine Einbahn-Replizierung von Projektarchivs über das Netz vornehmen zu können. Größere Teile der Arbeitskopie-Metadaten wurden überarbeitet, so dass nicht mehr XML benutzt wurde (was sich in erhöhter Geschwindigkeit auf Client-Seite niederschlug), während das Berkeley-DB-Projektarchiv-Backend die Fähigkeit erhielt, sich nach einem Server-Crash automatisch wiederherzustellen.

Subversion 1.5 (Juni 2008)
: Release 1.5 brauchte viel länger als vorige Releases, doch das Hauptfeature war gigantisch: Halbautomatische Verfolgung des Verzweigens und Zusammenführens. Dies war eine riesige Wohltat für Anwender und schob Subversion weit jenseits der Fähigkeiten von CVS und in die Reihen kommerzieller Mitbewerber wie Perforce und ClearCase. Subversion 1.5 führte auch eine große Anzahl anderer, benutzerorientierter Features ein, wie die interaktive Auflösung von Dateikonflikten, partielle Checkouts, client-seitige Verwaltung von Änderungslisten, eine starke neue Syntax für External-Definitionen und SASL-Authentifizierungsunterstützung für den **svnserve**-Server.

Subversion 1.6 (March 2009)
: Release 1.6 fuhr damit fort, das Verzweigen und Zusammenführen robuster zu machen, indem Baumkonflikte eingeführt wurden. Auch an anderen bestehenden Funktionen wurden Verbesserungen vorgenommen: weitergehende interaktive Optionen zur Konfliktauflösung, rekursives Entfernen und vollständige Unterstützung des Ausschliessens fur unvollständige Checkouts, dateibasierte Definition von Externals sowie Protokollierungsunterstützung ähnlich wie bei **mod_dav_svn**. Auch für den Kommandozeilen-Client wurde eine neue Kurzschreibweise zum Referenzieren von Subversion-Projektarchiv.URLs eingeführt.

Publikum

Dieses Buch ist für computerkundige Leute geschrieben, die mit Subversion ihre Daten verwalten wollen. Obwohl Subversion unter verschiedenen Betriebssystemen läuft, ist die primäre Benutzerschnittstelle kommandozeilenbasiert. Dieses Kommandozeilenwerkzeug (**svn**) und einige zusätzliche Hilfsprogramme stehen im Mittelpunkt dieses Buches.

Aus Gründen der Vereinheitlichung gehen die Beispiele in diesem Buch davon aus, dass der Leser ein unixähnliches Betriebssystem benutzt und mit Unix und Kommandozeilenschnittstellen verhältnismäßig gut zurechtkommt. Nichtsdestotrotz läuft **svn** auch unter anderen Betriebssystemen als Unix, etwa Microsoft Windows. Bis auf ein paar Ausnahmen, wie z.B. die Verwendung umgekehrter Schrägstriche (\) statt Schrägstrichen (/) als Pfadtrenner, sind die Ein- und Ausgaben dieses Werkzeugs unter Windows identisch zur Unix-Version.

Die meisten Leser sind wahrscheinlich Programmierer oder Systemadministratoren, die Änderungen an Quellcode verfolgen müssen. Das ist der am meisten verbreitete Einsatzzweck von Subversion, so dass alle Beispiele in diesem Buch auf diesem Szenario beruhen. Doch Subversion kann gleichwohl dazu benutzt werden, Änderungen an allerlei Arten von Informationen zu verwalten – Bilder, Musik, Datenbanken, Dokumentation usw. Für Subversion sind alle Daten einfach Daten.

Obwohl dieses Buch unter der Annahme geschrieben worden ist, dass der Leser noch nie ein Versionskontrollsystem benutzt hat, haben wir auch versucht, für Anwender von CVS (und anderen Systemen) den Sprung zu Subversion so schmerzlos wie möglich zu machen. Ab und zu werden in Randnotizen andere Versionskontrollsysteme erwähnt, und Anhang B, *Subversion für CVS-Benutzer* fasst viele der Unterschiede zwischen CVS und Subversion zusammen.

Es sei angemerkt, dass es sich bei den Quelltexten in diesem Buch nur um Beispiele handelt. Obwohl sie sich mit den passenden Compiler-Aufrufen übersetzen ließen, sollen sie lediglich ein besonderes Szenario illustrieren und nicht als Vorlage für guten Programmierstil oder gute Programmierpraxis dienen.

Wie dieses Buch zu lesen ist

Technische Bücher stehen immer vor einem bestimmten Dilemma: ob sie *von-oben* oder *von-unten* Lernenden entgegenkommen sollen. Ein von-oben Lernender bevorzugt es, Dokumentation zu lesen oder zu überfliegen und dabei einen groben Überblick über das Funktionieren des Systems zu erhalten, bevor er beginnt, die Software zu verwenden. Ein von-unten Lernender ist eine Person, für die „Lernen durch Ausprobieren“ gilt, jemand, der in die Software eintauchen möchte, um beim

Ausprobieren herauszufinden, wie sie funktioniert, und wenn nötig Abschnitte im Buch nachschlägt. Die meisten Bücher werden für die eine oder andere Art dieser Personen geschrieben, wobei dieses Buch zweifellos den von-oben Lernenden entgegenkommt. (Und wenn Sie gerade diesen Abschnitt lesen, sind Sie wahrscheinlich selber ein von-oben Lernender!) Verzweifeln Sie jedoch nicht, falls Sie ein von-unten Lerner sind. Während dieses Buch als eine breite Betrachtung der Themen rund um Subversion gestaltet ist, beinhaltet jeder Abschnitt eine reichhaltige Auswahl an Beispielen, die sie ausprobieren können. Die Ungeduldigen, die einfach weitermachen wollen, können sofort zu Anhang A, *Subversion-Einführung für einen schnellen Start* springen.

Ungeachtet Ihrer Lernmethode zielt dieses Buch darauf ab, für Menschen unterschiedlicher Herkunft nützlich zu sein – von Menschen ohne vorherige Erfahrung mit Versionskontrolle bis hin zu erfahrenen Systemadministratoren. Je nach Ihrer Herkunft können bestimmte Kapitel mehr oder weniger wichtig für Sie sein. Was nun folgt, kann als „Leseempfehlung“ für verschiedene Typen von Lesern betrachtet werden:

Erfahrene Systemadministratoren
: Die Annahme ist, dass Sie wahrscheinlich bereits Versionskontrolle verwendet haben und darauf brennen, möglichst schnell einen Subversion-Server zum Laufen zu bekommen. Kapitel 5, *Verwaltung des Projektarchivs* und Kapitel 6, *Konfiguration des Servers* zeigen, wie Sie Ihr erstes Projektarchiv erzeugen und es über das Netz verfügbar machen können. Danach sind Kapitel 2, *Grundlegende Benutzung* und Anhang B, *Subversion für CVS-Benutzer* die schnellsten Wege zum Lernen des Subversion-Clients.

Neulinge
: Wahrscheinlich hat Ihr Administrator Subversion bereits aufgesetzt, und Sie möchten nun lernen, wie man den Client benutzt. Falls Sie noch nie ein Versionskontrollsystem benutzt haben, ist Kapitel 1, *Grundlegende Konzepte* eine unbedingt notwendige Einführung in die Konzepte der Versionskontrolle. Kapitel 2, *Grundlegende Benutzung* ist eine Führung durch den Subversion-Client.

Fortgeschrittene
: Ob Sie ein Benutzer oder ein Administrator sind, letztendlich wird Ihr Projekt anwachsen. Sie werden lernen wollen, wie man fortgeschrittene Dinge mit Subversion machen kann, etwa Zweige verwenden und Zusammenführungen durchführen (Kapitel 4, *Verzweigen und Zusammenführen*), wie Subversions Unterstützung von Eigenschaften (Kapitel 3, *Fortgeschrittene Themen*) zu benutzen ist, wie Laufzeitoptionen konfiguriert werden können (Kapitel 7, *Subversion an Ihre Bedürfnisse anpassen*) und vieles mehr. Diese Kapitel sind zunächst nicht kritisch, jedoch sollten Sie sie lesen, sobald Sie mit den Grundlagen vertraut sind.

Entwickler
: Unter der Annahme, dass Sie bereits mit Subversion vertraut sind und es nun entweder erweitern oder neue Software basierend auf einem seiner zahlreichen APIs erstellen möchten, ist Kapitel 8, *Subversion integrieren* genau das, was sie suchen.

Das Buch schließt mit einer Referenz – Kapitel 9, *Die vollständige Subversion Referenz* ist ein Referenzhandbuch für alle Befehle von Subversion, und die Anhänge behandeln eine Anzahl nützlicher Themen. Dies sind die Kapitel, zu denen Sie sehr wahrscheinlich zurückkehren werden, wenn Sie dieses Buch beendet haben.

Aufbau dieses Buchs

Hier sind die folgenden Kapitel und ihr Inhalt aufgeführt:

Kapitel 1, *Grundlegende Konzepte*
: Erklärt die Grundlagen von Versionskontrolle und unterschiedliche Versionierungsmodelle sowie das Projektarchiv von Subversion, Arbeitskopien und Revisionen.

Kapitel 2, *Grundlegende Benutzung*
: Ein Spaziergang durch den Tag eines Subversion-Anwenders. Es zeigt, wie ein Subversion-Client verwendet wird, um Daten zu bekommen, zu verändern und abzuliefern.

Kapitel 3, *Fortgeschrittene Themen*
: Behandelt komplexere Eigenschaften, denen Benutzer letztendlich begegnen werden, wie etwa versionierte Metadaten, Dateisperren und Peg-Revisionen.

Kapitel 4, *Verzweigen und Zusammenführen*

Behandelt Zweige, Zusammenführungen und Etikettieren inklusive empfohlener Vorgehensweisen beim Verzweigen und Zusammenführen, übliche Szenarien, wie Änderungen wieder rückgängig gemacht werden können und wie einfach von einem Zweig zum nächsten gewechselt werden kann.

Kapitel 5, *Verwaltung des Projektarchivs*
: Beschreibt die Grundlagen des Subversion-Projektarchivs, wie man ein Projektarchiv anlegt, konfiguriert und wartet sowie die Tools, die man hierfür benutzen kann

Kapitel 6, *Konfiguration des Servers*
: Erklärt, wie man einen Subversion-Server konfiguriert und unterschiedliche Arten auf ein Projektarchiv zuzugreifen: `HTTP`, das `svn`-Protokoll und über die lokale Festplatte. Behandelt werden hier auch die Authentifizierung, die Autorisierung und der anonyme Zugriff.

Kapitel 7, *Subversion an Ihre Bedürfnisse anpassen*
: Untersucht die Subversion-Client-Konfigurationsdateien, die Handhabung internationalisierter Texte und wie man externe Tools zur Zusammenarbeit mit Subversion bringt.

Kapitel 8, *Subversion integrieren*
: Beschreibt die Interna von Subversion, das Subversion-Dateisystem und die Verwaltungsbereiche der Arbeitskopie aus der Sicht eines Programmierers. Hier wird auch gezeigt, wie die veröffentlichten APIs in einem Programm verwendet werden, das Subversion benutzt.

Kapitel 9, *Die vollständige Subversion Referenz*
: Erklärt detailreich jeden Unterbefehl von **svn**, **svnadmin** und **svnlook** mit vielen Beispielen für die ganze Familie.

Anhang A, *Subversion-Einführung für einen schnellen Start*
: Für die Ungeduldigen eine Anleitung im Schnelldurchlauf für die Installation und die sofortige Benutzung. Seien Sie gewarnt!

Anhang B, *Subversion für CVS-Benutzer*
: Behandelt die Ähnlichkeiten und Unterschiede zwischen Subversion und CVS mit etlichen Vorschlägen, wie man sich all die schlechten Angewohnheiten aus jahrelangem CVS-Gebrauch wieder abgewöhnen kann. Dies beinhaltet Subversion-Revisionsnummern, versionierte Verzeichnisse, Offline-Tätigkeiten, **update** und **status**, Zweige, Tags, Metadaten, Konfliktauflösung und Authentifizierung.

Anhang C, *WebDAV und Autoversionierung*
: Beschreibt die Details zu WebDAV und DeltaV und wie man sein Subversion-Projektarchiv konfiguriert, damit es als freigegebenes DAV-Laufwerk schreibbar in das Dateisystem eingehängt werden kann.

Anhang D, *Copyright*
: Eine Kopie der Creative Commons Attribution License, unter der dieses Buch lizenziert ist.

Dieses Buch ist frei

Dieses Buch startete aus Dokumentationsschnipseln von Entwicklern des Subversion-Projektes, die in einem Werk gebündelt und umgeschrieben wurden. Insofern war es immer schon unter einer freien Lizenz (siehe Anhang D, *Copyright*). Tatsächlich wurde das Buch unter den Augen der Öffentlichkeit geschrieben, ursprünglich als Teil des Subversion Projektes selbst. Das bedeutet zweierlei:

- Sie werden stets die neueste Version dieses Buchs im eigenen Subversion-Projektarchiv finden.
- Sie können an diesem Buch Änderungen vornehmen und es wie auch immer weiter verteilen – es unterliegt einer freien Lizenz. Ihre einzige Verpflichtung besteht darin, den Hinweis auf die ursprünglichen Autoren beizubehalten. Natürlich würden wir es bevorzugen, wenn Sie Rückmeldungen und Verbesserungen der Subversion-Entwicklergemeinde zukommen ließen, anstatt Ihre Privatversion zu verteilen.

Die Homepage der Entwicklungs- und Übersetzungsaktivitäten auf freiwilliger Basis ist http://svnbook.red-bean.com. Dort finden Sie Links auf die neuesten Releases und mit Tags versehene Versionen des Buchs in verschiedenen Formaten ebenso wie eine Anleitung, auf das Subversion-Projektarchiv des Buchs zuzugreifen (dort lebt sein Quellcode im DocBook-XML-Format). Rückmeldungen sind willkommen – ja sogar erwünscht. Bitte senden Sie alle Kommentare, Beschwerden und

Patches für die Sourcen des Buchs an <svnbook-dev@red-bean.com>.

Danksagungen

Dieses Buch wäre nicht möglich (und auch nicht sehr nützlich) wenn es Subversion nicht gäbe. Dafür möchten die Autoren Brian Behrendorf danken sowie CollabNet für die Vision, solch ein riskantes und ehrgeiziges Open-Source-Projekt zu finanzieren; Jim Blandy für den ursprünglichen Namen von Subversion und sein Design – wir lieben Dich, Jim; Karl Fogel, dafür, dass er so ein guter Freund und Leiter der Gemeinde ist, in dieser Reihenfolge. [4]

Dank an O'Reilly und das Team der professionellen Redakteure, die uns geholfen haben, diesen Text in unterschiedlichen Ständen seiner Evolution zu verbessern: Chuck Toporek, Linda Mui, Tatiana Apandi, Mary Brady und Mary Treseler. Eure Geduld und Unterstützung waren enorm.

Schließlich danken wir den zahllosen Menschen, die zu diesem Buch durch informelle Rezensionen, Vorschläge, und Fehlerbehebungen beigetragen haben. Der Ausdruck und die Wartung einer vollständigen Liste dieser Leute wäre an dieser Stelle nicht praktikabel. Dennoch mögen deren Namen für immer in der Versionskontrollgeschichte dieses Buches weiterleben!

[4] Oh, und Danke, Karl, dafür, dass du zu viel zu tun hattest, um das Buch selbst zu schreiben.

Kapitel 1. Grundlegende Konzepte

Das Kapitel ist eine kurze, lockere Einführung in Subversion und seinem Ansatz zu Versionskontrolle. Wir besprechen die grundlegenden Konzepte von Versionskontrolle und arbeiten uns in die Richtung von Subversion und dessen spezifischen Ideen und zeigen einfache Beispiele zur Anwendung.

Obwohl die Beispiele in diesem Kapitel Leute zeigen, die gemeinsam an Quellcode arbeiten, sei daran erinnert, dass Subversion alle möglichen Arten von Datensammlungen verwalten kann – es beschränkt sich nicht darauf, Entwicklern zu helfen.

Grundlagen der Versionskontrolle

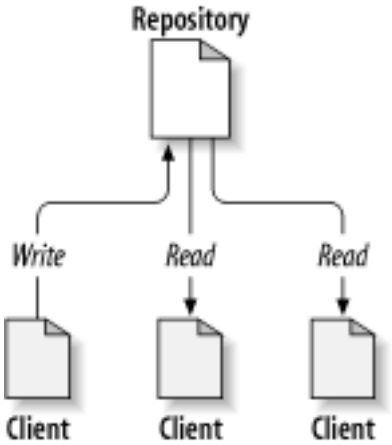
Ein Versionskontrollsystem (oder Revisionskontrollsystem) ist ein System, das inkrementelle Versionen (oder Revisionen) von Dateien und, in manchen Fällen, Verzeichnissen über die Zeit hinweg verfolgt. Natürlich ist es für sich nicht sehr interessant, die verschiedenen Versionen eines Anwenders (oder einer Gruppe von Anwendern) zu verfolgen. Was ein Versionskontrollsystem nützlich macht, ist die Tatsache, dass es Ihnen erlaubt, die Änderungen zu untersuchen, die zu jeder dieser Versionen geführt haben und es ermöglicht, erstere jederzeit wieder aufzurufen.

In diesem Abschnitt werden wir einige Komponenten und Konzepte von Versionskontrollsystemen auf ziemlich hohem Niveau vorstellen. Wir werden uns auf moderne Versionskontrollsysteme beschränken – in der heutigen vernetzten Welt hat es wenig Sinn, Versionskontrollsystemen eine Berechtigung einzuräumen, die nicht über Netze hinweg arbeiten können.

Das Projektarchiv

Im Kern eines Versionskontrollsystems ist ein Projektarchiv, das der zentrale Speicher der Daten dieses Systems ist. Das Projektarchiv speichert Informationen gewöhnlicherweise in Form eines *Dateisystembaumes*, eine Hierarchie aus Dateien und Verzeichnissen. Eine beliebige Anzahl von *Clients* verbindet sich mit dem Projektarchiv und liest oder schreibt diese Dateien. Durch den Schreibvorgang, macht ein Client Informationen für andere verfügbar. Durch den Lesevorgang bekommt der Client Informationen von anderen zur Verfügung gestellt. Abbildung 1.1, „Ein typisches Client/Server System“ verdeutlicht das.

Abbildung 1.1. Ein typisches Client/Server System



Warum ist das interessant? Bis zu diesem Punkt hört sich das wie die Definition eines typischen Datei-Servers an. Und tatsächlich, das Projektarchiv *ist* eine Art von Datei-Server, aber nicht von der Art, die Sie kennen. Was das Subversion-Projektarchiv so speziell macht ist, dass es sich während die Dateien im Projektarchiv geändert werden jede Version jener Dateien merkt.

Wenn ein Client Daten aus dem Projektarchiv liest, bekommt der Client üblicherweise nur die letzte Version des Dateisystem-Baumes zu sehen. Was ein Versionskontrollsystem aber interessant macht, ist darüber hinaus die Fähigkeit, vorherige Zustände des Dateibaums aus dem Projektarchiv abzurufen. Ein Versionskontrollsystem kann historische Fragen stellen, wie „Was beinhaltete das Verzeichnis am letzten Mittwoch?“ und „Wer war die Person, die als letztes die Datei geändert hat und welche

Änderungen hat sie gemacht?“. Diese Art von Fragen sind die Grundlage eines Versionskontrollsystems.

Die Arbeitskopie

Der Wert eines Versionskontrollsystems rührt von der Tatsache her, dass es Versionen von Dateien und Verzeichnissen verfolgt, doch der Rest des Software-Universums arbeitet nicht auf „Versionen von Dateien und Verzeichnissen“. Die meisten Programme wissen, wie mit einer *einzelnen* Version eines bestimmten Dateityps umgegangen wird. Wie arbeitet also ein Anwender eines Versionskontrollsystems konkret mit einem abstrakten – und oft entfernten – Projektarchiv voll mit mehreren Versionen verschiedener Dateien? Wie schaffen es seine oder ihre Textbearbeitungs-Software, Präsentations-Software, Quelltexteditoren, Web-Design-Software oder sonstigen Programme, die alle nur mit einfachen Dateien umgehen können, Zugriff auf solche Dateien zu bekommen? Die Antwort findet sich im Versionskontroll-Konstrukt, das als *Arbeitskopie* bekannt ist.

Eine Arbeitskopie ist buchstäblich eine lokale Kopie einer bestimmten Version der vom VCS verwalteten Anwenderdaten mit der der Anwender frei arbeiten kann. Arbeitskopien[1] sehen für andere Software aus wie alle anderen lokalen Verzeichnisse voller Dateien, so dass diese Programme nicht „versionskontroll-bewusst“ sein müssen, um die Daten zu lesen und zu schreiben. Die Aufgabe, die Arbeitskopie zu verwalten und Änderungen an ihrem Inhalt zum und vom Projektarchiv zu übergeben, fällt genau der Client-Software des Versionskontrollsystems zu.

Versionierungsmodelle

Wenn die primäre Mission eines Versionskontrollsystems darin besteht, die unterschiedlichen Versionen digitaler Informationen über die Zeit hinweg zu verfolgen, liegt eine sehr nahe sekundäre Mission darin, das kollaborative Bearbeiten und Teilen dieser Daten zu ermöglichen. Jedoch verwenden unterschiedliche Systeme auch unterschiedliche Strategien, um dies zu bewerkstelligen. Aus einer Reihe von Gründen ist es wichtig, diese Unterschiede zu verstehen. Zunächst hilft es dabei, bestehende Versionskontrollsysteme zu vergleichen und gegenüberzustellen, falls Ihnen andere Systeme begegnen, die Subversion ähneln. Darüber hinaus wird es Ihnen helfen, Subversion effektiver zu benutzen, da Subversion selbst eine Reihe unterschiedlicher Arbeitsweisen unterstützt.

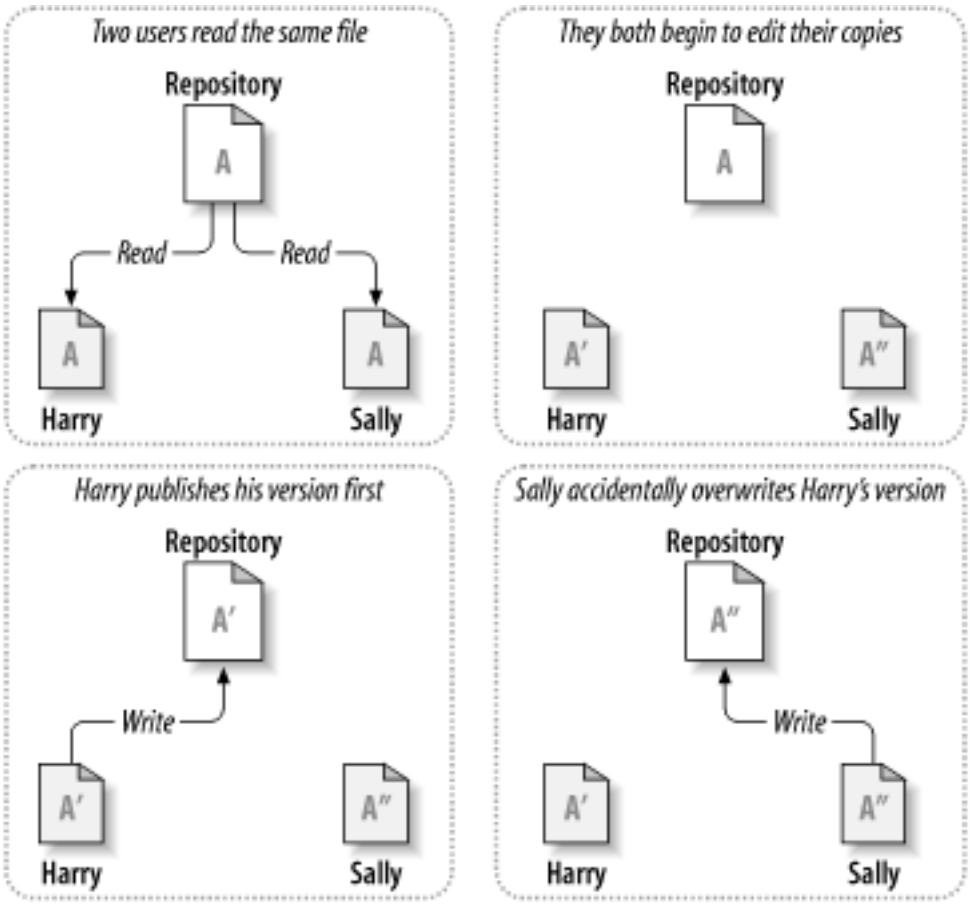
Das Problem verteilter Dateizugriffe

Alle Versionskontrollsysteme haben das gleiche fundamentale Problem zu lösen: Wie soll es Anwendern erlaubt werden, Informationen zu teilen, aber sie davor zu bewahren, sich gegenseitig auf die Füße zu treten? Es ist allzu einfach, die Änderungen eines anderen im Projektarchiv zu überschreiben.

Stellen Sie sich einmal folgendes Szenario in Abbildung 1.2, „Das zu vermeidende Problem“ vor: Zwei Kollegen, Harry und Sally, haben sich entschieden, dieselbe Datei zur gleichen Zeit zu bearbeiten. Harry speichert seine Änderungen zuerst im Projektarchiv, es ist aber möglich, dass Sally nur einige Augenblicke später seine Datei mit ihrer überschreibt. Harrys Änderungen der Datei sind zwar nicht für immer verloren (da das System jede Änderung aufzeichnet), aber alle seine Änderungen sind in Sallys später gespeicherter Version der Datei nicht vorhanden, da Sally diese Änderungen noch gar nicht kannte. Das heißt, dass Harrys Arbeit doch verloren ist, zumindest in der neuesten Version der Datei, und das vermutlich aus Versehen. Eine solche Situation wollen wir auf alle Fälle vermeiden.

Abbildung 1.2. Das zu vermeidende Problem

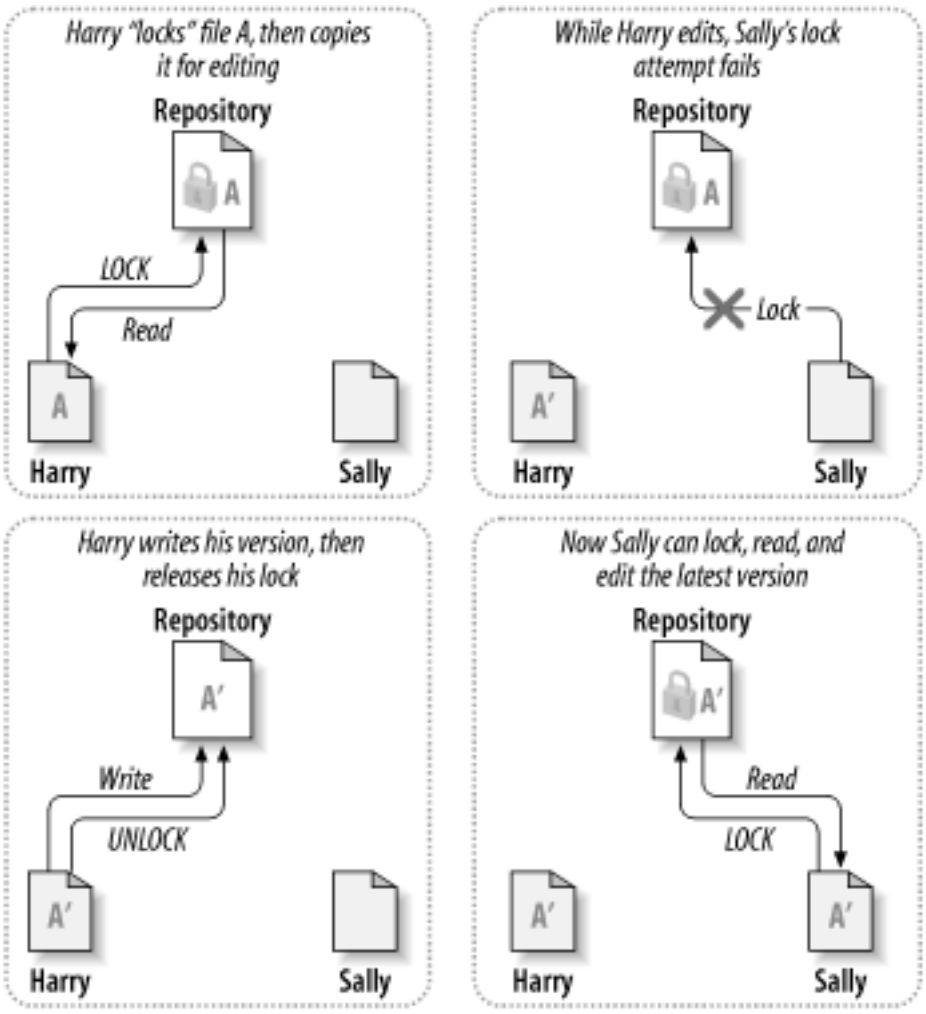
[1] Der Begriff „Arbeitskopie“ kann allgemein auf die lokale Instanz einer jeden Dateiversion angewendet werden. Die meisten Leute verwenden den Begriff aber, wenn sie sich auf einen kompletten Verzeichnisbaum beziehen, der Dateien und Verzeichnisse enthält, die vom Versionskontrollsystem verwaltet werden.



Die Sperren-Ändern-Entsperren-Lösung

Viele Versionskontrollsysteme verwenden ein *Sperren-Ändern-Entsperren*-Modell um zu verhindern, dass verschiedene Autoren sich gegenseitig die Änderungen löschen. Bei diesem Modell erlaubt das Projektarchiv nur jeweils einem Programmierer den Zugriff auf eine Datei. Harry müsste also die Datei sperren, ehe er anfängt, seine Änderungen einzugeben. Wenn Harry die Datei gesperrt hat, kann Sally sie nicht ebenfalls sperren und daher auch nichts ändern. Sie kann die Datei in der Zeit nur lesen und darauf warten, dass Harry mit seiner Arbeit fertig ist und die Datei entsperrt. Abbildung 1.3, „Die Sperren-Ändern-Entsperren-Lösung“

Abbildung 1.3. Die Sperren-Ändern-Entsperren-Lösung



Das Problem bei einem Sperren-Ändern-Entsperren-Modell liegt in seinen Beschränkungen, die oft zu schier unüberwindlichen Hindernissen führen können.

- *Das Sperren kann zu administrativen Problemen führen.* Vielleicht sperrt Harry eine Datei und vergisst dann, sie zu entsperren. In der Zwischenzeit sind Sally, die ebenfalls Änderungen an dieser Datei durchführen will, die Hände gebunden. Und dann geht Harry in Urlaub. Nun muss Sally sich an einen Administrator wenden, um die Datei entsperrt zu bekommen. Das Ergebnis sind unnötige Verzögerungen und vergeudete Zeit.

- *Das Sperren kann zu einer unnötigen Serialisierung führen.* Was ist, wenn Harry z. B. den Anfang einer Textdatei bearbeiten will, während Sally einfach nur das Ende ändern möchte? Diese Änderungen würden sich überhaupt nicht gegenseitig beeinflussen und könnten problemlos gleichzeitig durchgeführt werden, vorausgesetzt, sie würden anschließend vernünftig zusammengefasst. Es gibt in dieser Situation keinen Grund, der Reihe nach zu arbeiten.

- *Das Sperren kann zu einem falschen Gefühl von Sicherheit führen.* Angenommen Harry sperrt und bearbeitet Datei A, während Sally gleichzeitig Änderungen an Datei B durchführt. Was ist, wenn A und B voneinander abhängig sind und die jeweiligen Änderungen nicht kompatibel sind? Plötzlich funktioniert das Zusammenspiel zwischen A und B nicht mehr. Das System des Sperrens hat dieses Problem nicht verhindert, doch hat es fälschlicherweise zu einem Gefühl der Sicherheit geführt. Es ist leicht, sich vorzustellen, dass Harry und Sally der Meinung waren, dass jeder von ihnen eine eigenständige, voneinander unabhängige Änderung durchgeführt hat und dass das Sperren dazu geführt hat, dass sie ihre inkompatiblen Änderungen nicht vorher miteinander besprochen haben. Sperren ist oft ein Ersatz für echte Kommunikation.

Die Kopieren–Ändern–Zusammenfassen-Lösung

Subversion, CVS und viele andere Versionskontrollsysteme benutzen ein *Kopieren–Ändern–Zusammenfassen*-Modell als Alternative zum Sperren. In diesem Modell verbindet sich jeder Client der Anwender mit dem Projektarchiv und erzeugt eine

persönliche Arbeitskopie. Dann arbeiten die Anwender gleichzeitig und unabhängig voneinander an ihren privaten Kopien. Am Ende werden dann alle Einzelkopien zu einer neuen, aktuellen Version zusammengeführt. Das Versionskontrollsystem hilft oft bei dieser Zusammenführung, aber letztlich ist der Mensch dafür verantwortlich, das es korrekt abläuft.

Hier ist ein Beispiel: Harry und Sally haben sich jeweils eine eigene Arbeitskopie des im Projektarchiv vorhandenen Projektes geschaffen. Beide arbeiten nun an der selben Datei A innerhalb ihrer jeweiligen Kopien. Sally speichert ihre Version zuerst im Projektarchiv ab. Wenn Harry später ebenfalls versucht, seine Änderungen zu speichern, informiert ihn das Projektarchiv, das seine Datei A nicht mehr aktuell ist. Das bedeutet, dass seitdem er sich seine Kopie erschaffen hat, sind irgendwelche Änderungen aufgetreten. Also bittet Harry seinen Client darum, diese neuen Änderungen in seine Arbeitskopie der Datei A einzuarbeiten. Die Möglichkeit besteht, dass Sallys Änderungen mit seinen nicht überlappen, wenn er also alle Änderungen eingearbeitet hat, kann er seine Arbeitskopie zurück in das Projektarchiv speichern. Die Abbildungen Abbildung 1.4, „„Kopieren – Ändern – Zusammenfassen“ - Lösung“ und Abbildung 1.5, „„Kopieren – Ändern – Zusammenfassen“ - Lösung (Fortsetzung)“ zeigen diesen Prozess.

Abbildung 1.4. „Kopieren – Ändern – Zusammenfassen“ - Lösung

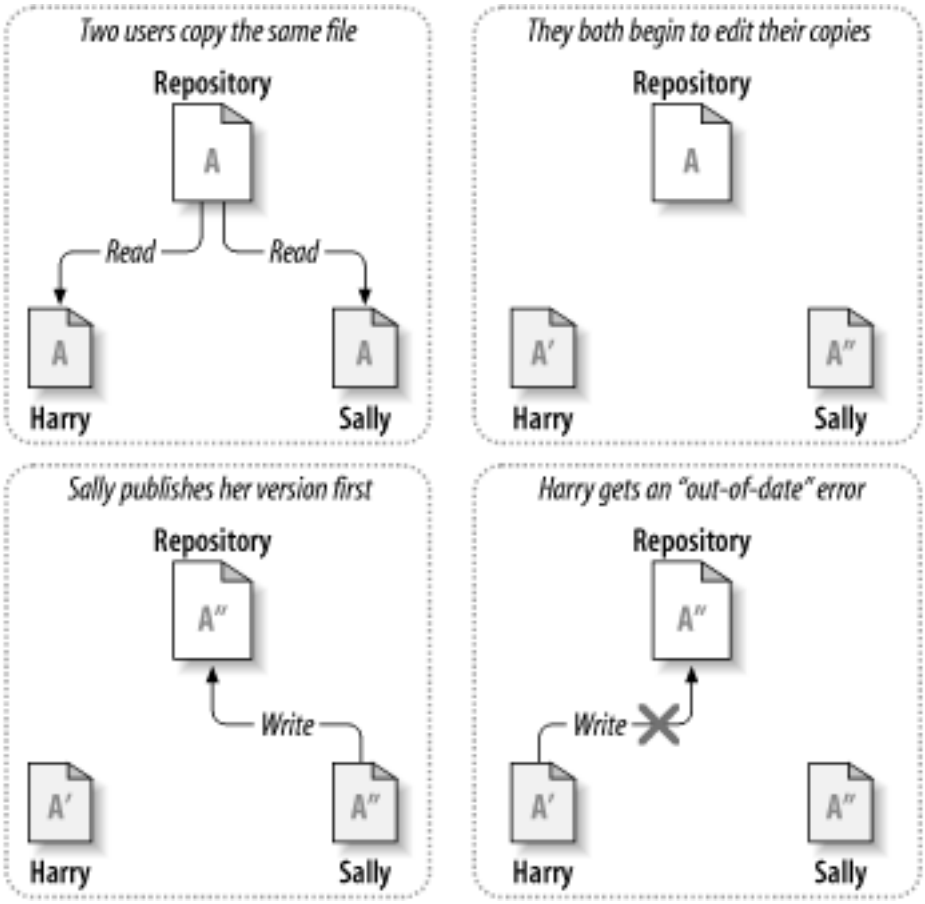
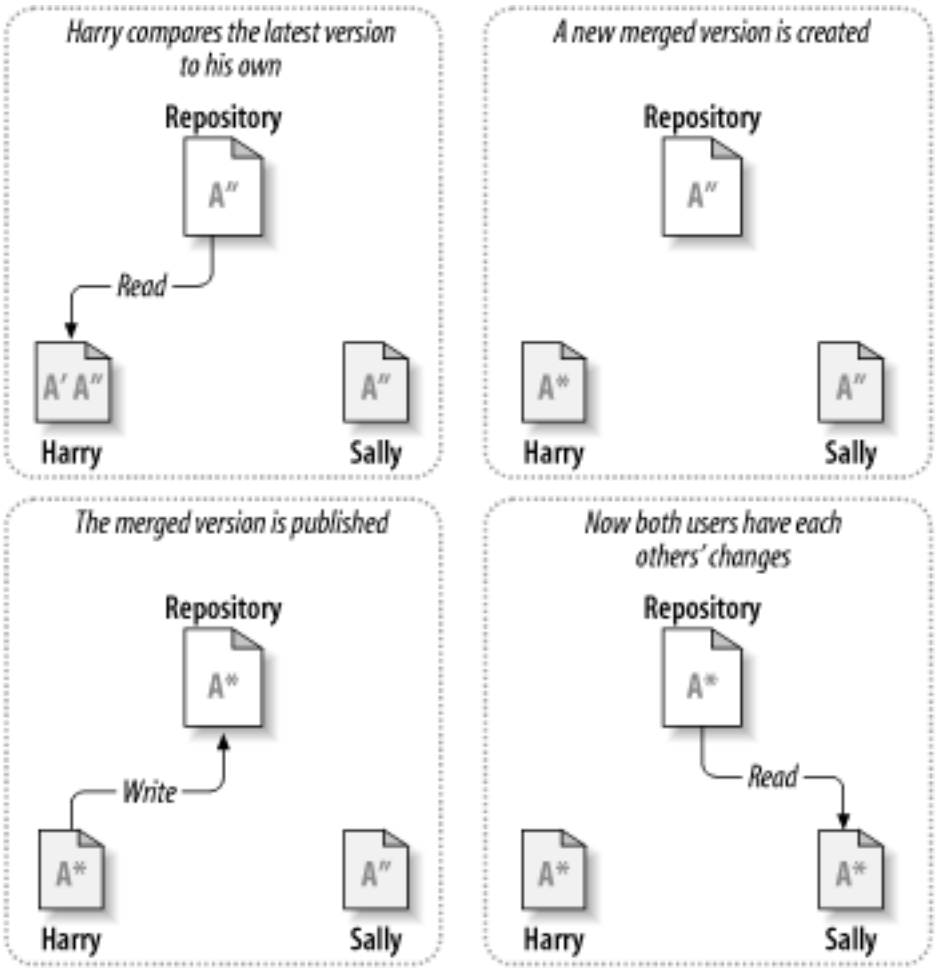


Abbildung 1.5. „Kopieren – Ändern – Zusammenfassen“ - Lösung (Fortsetzung)



Was aber passiert, *wenn* Sallys Änderungen mit Harrys kollidieren? Diese Situation wird *Konflikt* genannt und ist normalerweise kein allzu großes Problem. Wenn Harry Sallys Änderungen in seine Datei einpflegen lassen will, werden in seiner Datei die miteinander in Konflikt stehenden Änderungen gekennzeichnet, er kann sämtliche Änderungen sehen und manuell zwischen ihnen wählen. Das Programm löst solche Konfliktsituationen nicht automatisch, nur Menschen sind in der Lage, die Probleme zu erkennen und die nötigen intelligenten Änderungen durchzuführen. Wenn Harry die Konfliktsituationen – vielleicht nach einer kurzen Diskussion mit Sally – gelöst hat, kann er seine Datei problemlos ins Projektarchiv speichern.

Dieses *Kopieren-Ändern-Zusammenfassen*-Modell (engl. copy-modify-merge model) klingt vielleicht ein wenig chaotisch, in der Praxis aber läuft es völlig glatt. Die einzelnen Anwender können parallel arbeiten, ohne einander in die Quere zu kommen oder unnötig warten zu müssen. Wenn sie an den selben Dateien arbeiten, zeigt es sich meistens, dass ihre jeweiligen Änderungen einander überhaupt nicht stören, wirkliche Konflikte sind selten. Und die Zeit, die es beansprucht, eine solche Konfliktsituation zu lösen, ist meist wesentlich kürzer als der Zeitverlust, der durch das Sperren auftritt.

Am Ende läuft alles auf einen kritischen Faktor hinaus: Kommunikation zwischen den Anwendern. Wenn diese Kommunikation eher spärlich abläuft, häufen sich sowohl semantische als auch syntaktische Konflikte. Kein System kann Anwender dazu zwingen, vernünftig miteinander zu kommunizieren und kein System kann semantische Konflikte erkennen. Also hat es auch keinen Sinn, sich in dem falschen Gefühl von Sicherheit zu wiegen, dass das Sperren Konflikte irgendwie vermeiden könnte. In der Praxis verringert das System des Sperrens mehr als andere die Produktivität.

Wann das Sperren notwendig ist

Obwohl das Sperren-Ändern-Entsperren-Modell im Allgemeinen als schädlich für die Zusammenarbeit empfunden wird, ist es mitunter angebracht.

Das Kopieren-Ändern-Zusammenführen-Modell beruht auf der Annahme, dass Dateien kontextbezogen zusammenführbar sind – d.h., die Mehrzahl der Dateien im Projektarchiv sind zeilenorientierte Textdateien (wie z.B. Programm-Quelltext). Aber für Dateien in Binärformaten, wie Grafiken und Klänge, ist es oftmals nicht möglich,

konfliktäre Änderungen zusammenzuführen. In diesen Fällen müssen die Benutzer tatsächlich eine strikte Schrittfolge beim Ändern der Datei einhalten. Ohne geordneten Zugriff wird schließlich jemand seine Zeit auf Änderungen verschwenden, die längst verworfen wurden.

Auch wenn Subversion vorrangig ein Kopieren-Ändern-Zusammenführen-System ist, erkennt es die Notwendigkeit des Sperrens vereinzelter Dateien an und bietet dafür die Mechanismen. Wir diskutieren dieses Feature in „Sperren“.

Versionskontrolle nach Art von Subversion

Wir haben bereits erwähnt, dass Subversion ein modernes, netzbewusstes Versionskontrollsystem. Wie wir in „Grundlagen der Versionskontrolle“ beschrieben haben (unser Versionskontroll-Überblick auf hoher Ebene), dient ein Projektarchiv als Kern-Speichermechanismus für die versionierten Daten von Subversion, und über Arbeitskopien kommunizieren Anwender und ihre Software mit diesen Daten. In diesem Abschnitt werden wir damit beginnen, die besonderen Vorgehensweisen von Subversion bei der Implementierung von Versionskontrolle vorzustellen.

Subversion Projektarchive

Subversion implementiert das Konzept eines Projektarchivs für Versionskontrolle so, wie es jedes andere moderne Versionskontrollsystem auch machen würde. Im Gegensatz zu einer Arbeitskopie ist ein Subversion-Projektarchiv ein abstraktes Gebilde, das sich fast ausschließlich über die eigenen Subversion-Bibliotheken und -Werkzeuge manipulieren lässt. Da die meisten Interaktionen eines Anwenders mit Subversion die Benutzung des Subversion-Clients einbeziehen und im Kontext einer Arbeitskopie vollzogen werden, wird sich ein großer Teil dieses Buches mit dem Subversion-Projektarchiv und dessen Bearbeitung beschäftigen. Für die Feinheiten des Projektarchivs, siehe allerdings Kapitel 5, *Verwaltung des Projektarchivs*.

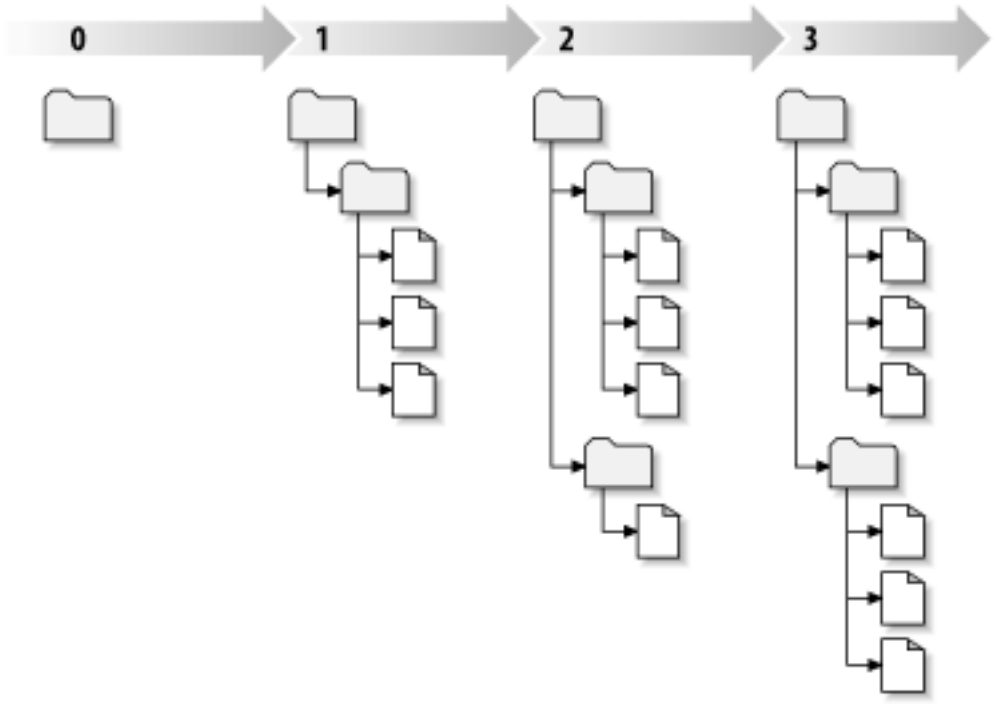
Revisionen

Ein Subversion-Client übergibt eine (d.h., übermittelt die Änderungen an einer) beliebigen Anzahl von Dateien und Verzeichnissen als eine einzige atomare Transaktion. Eine atomare Transaktion bedeutet: entweder es gehen alle Änderungen in das Projektarchiv oder keine. Angesichts von Programmabstürzen, Systemabstürzen, Netzproblemen oder anderer Benutzeraktionen hält Subversion an dieser Atomizität fest.

Jedes Mal wenn das Projektarchiv eine Übertragung annimmt, wird ein neuer Zustand des Dateisystem-Baums erzeugt, der *Revision* genannt wird. Jeder Revision wird eine einmalige natürliche Zahl zugewiesen, die um eins größer ist als die Vorgänger-Revision. Die anfängliche Revision eines frisch erzeugten Projektarchivs bekommt die Nummer 0 und besteht lediglich aus einem leeren Wurzelverzeichnis.

Abbildung 1.6, „Änderungen am Baum im Verlauf der Zeit“ zeigt, wie man sich das Projektarchiv vorstellen kann. Stellen Sie sich eine Reihe von Revisionsnummern vor, die bei 0 startet und von links nach rechts wächst. Jede Revisionsnummer hat einen Dateisystem-Baum unter sich hängen, der ein „Schnappschuss“ des Projektarchivs nach einer Übertragung ist.

Abbildung 1.6. Änderungen am Baum im Verlauf der Zeit



Globale Revisionsnummern

Anders als die meisten Versionskontrollsysteme werden die Revisionsnummern von Subversion auf *den kompletten Projektarchiv-Baum* anstatt auf einzelne Dateien angewendet. Jede Revisionsnummer wählt einen kompletten Baum aus; ein bestimmter Zustand nach der Übertragung einer Änderung. Man kann sich auch vorstellen, dass Revision N den Zustand des Projektarchiv-Dateisystems nach der n-ten Übertragung repräsentiert. Wenn Subversion-Benutzer von „Revision 5 von `foo.c`“ sprechen, meinen sie tatsächlich „`foo.c` so wie es in Revision 5 aussieht“. Beachten Sie, dass sich im Allgemeinen die Revisionen N und M einer Datei *nicht* notwendigerweise unterscheiden! Viele andere Versionskontrollsysteme verwenden dateibezogene Revisionsnummern, so dass dieses Konzept zunächst ungewöhnlich aussieht. (Ehemalige CVS-Benutzer sollten sich für weitergehende Informationen Anhang B, *Subversion für CVS-Benutzer* ansehen.)

Projektarchive adressieren

Subversion-Client-Programme verwenden URLs, um Dateien und Verzeichnisse in Subversion-Projektarchivs zu identifizieren. Meistens benutzen diese URLs die Standardsyntax, die es erlaubt, Servernamen und Portnummern als Teil des URL zu spezifizieren.

- http://svn.example.com/svn/project
- http://svn.example.com:9834/repos

Die Subversion-Projektarchiv-URLs sind nicht beschränkt auf den Typ `http://`. Da Subversion mehrere unterschiedliche Kommunikationswege zwischen seinen Clients und Servern anbietet, unterscheiden sich die zur Adressierung des Projektarchivs verwendeten URLs auf eine subtile Art, abhängig davon, welcher Zugriffsmechanismus zum Projektarchiv angewendet werden soll. Tabelle 1.1, „Projektarchiv-Zugriffs-URLs“ beschreibt, wie unterschiedliche URL Schemata auf die verfügbaren Zugriffsmethoden abgebildet werden. Details über die Serveroptionen von Subversion finden Sie unter Kapitel 6, *Konfiguration des Servers*.

Tabelle 1.1. Projektarchiv-Zugriffs-URLs

Schema	Zugriffsmethode
`file:///`	Direkter Zugriff auf das Projektarchiv (auf lokaler Platte)
`http://`	Zugriff über das WebDAV-Protokoll auf Apache-Server, die Subversion abhandeln können
`https://`	Wie `http://`, jedoch mit SSL-Verschlüsselung
`svn://`	Zugriff über ein besonderes Protokoll auf einen `svnserve`-Server
`svn+ssh://`	Wie `svn://`, jedoch über einen SSH Tunnel

Allerdings gibt es einige bemerkenswerte Feinheiten, wie Subversion mit URLs umgeht. Beispielsweise dürfen URLs, die die `file://`-Zugriffsmethode enthalten (für lokale Projektarchive verwendet), gemäß Konvention entweder den Servernamen `localhost` oder gar keinen Servernamen enthalten:

- file:///var/svn/repos
- file://localhost/var/svn/repos

Darüber hinaus müssen Benutzer des `file://` Schemas auf Windows-Plattformen eine inoffizielle „Standard“-Syntax verwenden falls auf Projektarchive auf derselben Maschine aber auf einem anderen Laufwerk zugegriffen werden soll. Beide der folgenden URL-Pfad-Syntaxen funktionieren, wobei `X` das Laufwerk ist, wo das Projektarchiv liegt:

- file:///X:/var/svn/repos
- file:///X|/var/svn/repos

Beachten Sie, dass ein URL Schrägstriche verwendet, selbst wenn die übliche (nicht-URL) Form eines Pfades unter Windows rückwärtige Schrägstriche verwendet. Beachten Sie ebenfalls, bei der Verwendung des Formats `file:///X|/` den URL in Anführungsstriche einzuschließen, damit der senkrechte Strich nicht als Pipe-Symbol interpretiert wird.

Sie können die `file://` URLs von Subversion nicht in einem normalen Web-Browser auf die Art und Weise verwenden wie andere `file://` URLs. Falls Sie versuchen, einen `file://` URL in einem gewöhnlichen Web-Browser anzusehen, wird der Inhalt der Datei von der angegebenen Stelle direkt aus dem Dateisystem gelesen und angezeigt. Allerdings befinden sich die Daten von Subversion in einem virtuellen Dateisystem (siehe „Projektarchiv-Schicht“), und der Browser wird nicht mit diesem Dateisystem umzugehen wissen.

Der Subversion-Client wandelt URLs nach Bedarf automatisch um, wie es auch ein Web-Browser macht. So wird beispielsweise der URL `http://host/path with space/project/españa` – der sowohl Leerzeichen als auch Zeichen aus dem höheren ASCII-Bereich enthält – automatisch von Subversion so interpretiert als ob sie `http://host/path%20with%20space/project/espa%C3%B1a` geschrieben hätten. Falls der URL Leerzeichen enthält, stellen Sie sicher, ihn auf der Kommandozeile in Anführungszeichen zu setzen, so dass Ihre Shell alles als ein einzelnes Argument für das Programm behandelt.

Es gibt eine erwähnenswerte Ausnahme von der Regel, wie Subversion URLs behandelt, die in vielen Kontexten auch auf die Behandlung lokaler Pfade anwendbar ist. Falls die letzte Pfadkomponente des URL oder lokalen Pfades einen Klammeraffen (@) enthält, müssen Sie eine besondere Syntax verwenden – in „Peg- und operative Revisionen“ beschrieben – damit Subversion diese Ressource passend ansprechen kann.

In Subversion 1.6 wurde eine neue Notation mit Zirkumflex (^) als Kurzschreibweise für „der URL des Wurzelverzeichnisses des Projektarchivs“ eingeführt. Sie können beispielsweise `^/tags/bigsandwich/` verwenden, um sich auf den URL des Verzeichnisses `/tags/bigsandwich` im Wurzelverzeichnis des Projektarchivs zu beziehen. Beachten Sie, dass dieser URL nur dann funktioniert, wenn Ihre aktuelles Arbeitsverzeichnis eine Arbeitskopie ist – der Kommandozeilen-Client kennt den URL des Projektarchiv-Wurzelverzeichnisses, da er sich die Metadaten der Arbeitskopie ansieht. Beachten Sie auch, dass Sie `^/` statt nur `^` verwenden (mit dem abschließenden Schrägstrich), wenn Sie sich auf das Wurzelverzeichnis des Projektarchivs beziehen möchten.

Subversion-Arbeitskopien

Eine Subversion-Arbeitskopie ist ein gewöhnlicher Verzeichnisbaum auf Ihrem lokalen System, der eine Ansammlung von Dateien enthält. Sie können diese Dateien nach belieben bearbeiten, und wenn es sich um Quelltexte handelt, können Sie hieraus Ihr Programm auf die übliche Weise compilieren. Ihre Arbeitskopie ist Ihr privater Arbeitsbereich: nie wird Subversion weder die Änderungen von anderen einpflegen, noch Ihre eigenen Änderungen anderen zur Verfügung stellen, bis Sie es ausdrücklich dazu auffordern. Sie können sogar mehrere Arbeitskopien desselben Projektes haben.

Nachdem Sie einige Änderungen an den Dateien Ihrer Arbeitskopie gemacht und sichergestellt haben, dass sie funktionieren, stellt Ihnen Subversion Befehle zur Verfügung, um Ihre Änderungen den anderen, die an Ihrem Projekt mitarbeiten, „publik“ zu machen (indem es ins Projektarchiv schreibt). Wenn die anderen ihre Änderungen veröffentlichen, stellt Ihnen Subversion Befehle zur Verfügung, um diese Änderungen in Ihr Arbeitsverzeichnis einzupflegen (indem es aus dem Projektarchiv liest).

Eine Arbeitskopie verfügt darüber hinaus über einige zusätzliche Dateien, die von Subversion erzeugt und gepflegt werden, um es bei diesen Befehlen zu unterstützen. Insbesondere enthält jedes Verzeichnis Ihrer Arbeitskopie ein Unterverzeichnis namens `.svn`, auch bekannt als das *Verwaltungsverzeichnis* der Arbeitskopie. Die Dateien in jedem Verwaltungsverzeichnis helfen Subversion dabei, zu erkennen, welche Dateien unveröffentlichte Änderungen enthalten und welche Dateien hinsichtlich der Arbeit anderer veraltet sind.

Obwohl `.svn` der allgemeine Standardname für das Subversion-Verwaltungsverzeichnis ist, könnten Windows mit dem ASP.NET Web-Application-Framework bekommen, das Zugriff auf Verzeichnisse verbietet, deren Namen mit einem Punkt (`.`) beginnt. Mit Rücksicht auf Anwender in solchen Situationen verwendet Subversion stattdessen `_svn` als Namen für das Verwaltungsverzeichnis, falls es eine Variable namens `SVN_ASP_DOT_NET_HACK` in seiner Arbeitsumgebung findet. Für dieses Buch gelten durchgängig alle Referenzen auf `.svn` ebenfalls für `_svn`, falls dieser „ASP.NET Hack“ angewendet wird.

Wie die Arbeitskopie funktioniert

Für jede Datei eines Arbeitsverzeichnis merkt sich Subversion (neben anderen Dingen) zwei essentielle Informationen:

- Auf welcher Revision Ihre Arbeitsdatei aufbaut (das wird die *Arbeitsrevision* der Datei genannt)
- Ein Zeitstempel, der festhält, wann die lokale Kopie das letzte Mal vom Projektarchiv aktualisiert wurde.

Mit diesen Informationen kann Subversion durch Kommunikation mit dem Projektarchiv feststellen, in welchem der folgenden Zustände sich eine Arbeitsdatei befindet:

Unverändert und aktuell
: Die Datei im Arbeitsverzeichnis ist unverändert, und keinerlei Änderungen an der Datei sind seit der Arbeitsrevision an das Projektarchiv übergeben worden. Ein **svn commit** der Datei würde nichts machen, und ein **svn update** der Datei auch nicht.

Lokal geändert und aktuell
: Die Datei wurde im Arbeitsverzeichnis geändert, und keinerlei Änderungen an der Datei sind seit der letzten Aktualisierung an das Projektarchiv übergeben worden. Es gibt lokale Änderungen, die noch nicht an das Projektarchiv übergeben worden sind, so dass ein **svn commit** der Datei Ihre Änderungen erfolgreich veröffentlichen würde, und ein **svn update** der Datei nichts tun würde.

Unverändert und veraltet
: Die Datei wurde im Arbeitsverzeichnis nicht geändert, jedoch im Projektarchiv. Die Datei sollte aktualisiert werden, damit sie bezüglich der letzten öffentlichen Revision aktuell ist. Ein **svn commit** der Datei würde nichts machen, und ein **svn update** der Datei würde die letzten Änderungen in Ihre Arbeitskopie einbringen.

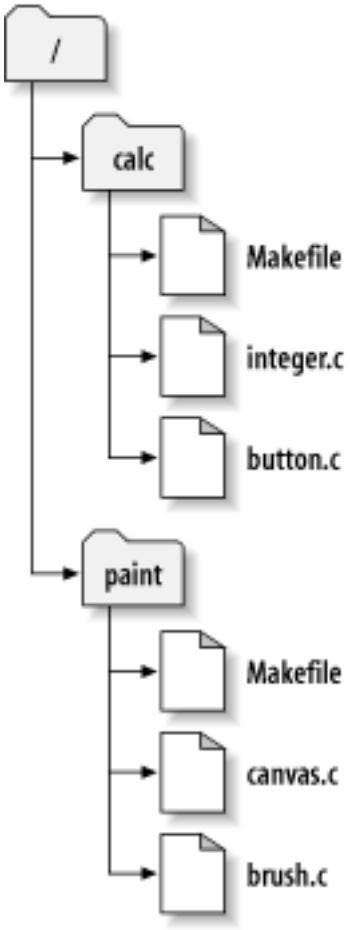
Lokal geändert und veraltet
: Die Datei wurde sowohl im Arbeitsverzeichnis als auch im Projektarchiv geändert. Ein **svn commit** der Datei würde mit einem „out-of-date“ Fehler abbrechen. Die Datei sollte erst aktualisiert werden; ein **svn update** Befehl würde versuchen, die öffentlichen mit den lokalen Änderungen zusammenzuführen. Wenn Subversion diese Zusammenführung nicht plausibel automatisch durchführen kann, wird die Auflösung des Konflikts dem Benutzer überlassen.

Grundlegende Interaktionen der Arbeitskopie

Oft enthält ein typisches Subversion-Projektarchiv die Dateien (oder den Quelltext) für verschiedene Projekte; für gewöhnlich ist jedes Projekt ein Unterverzeichnis im Dateisystembaum des Projektarchivs. Bei dieser Anordnung entspricht die Arbeitskopie eines Benutzers gewöhnlich einem bestimmten Unterverzeichnis des Projektarchivs.

Nehmen wir zum Beispiel an, Sie haben ein Projektarchiv, das zwei Software-Projekte beinhaltet, `paint` und `calc`. Jedes Projekt ist in einem eigenen Hauptverzeichnis abgelegt, wie in Abbildung 1.7, „Das Dateisystem des Projektarchivs“ dargestellt.

Abbildung 1.7. Das Dateisystem des Projektarchivs



Um eine Arbeitskopie zu erhalten, muss zunächst irgendein Teilbaum des Projektarchivs *ausgecheckt* werden (check out). (Der Begriff *check out* hört sich an, als habe es etwas mit dem Sperren oder Reservieren von Ressourcen zu tun, hat es aber nicht; es erzeugt lediglich eine Arbeitskopie des Projektes für Sie.) Wenn Sie zum Beispiel `/calc` auschecken, bekommen Sie eine Arbeitskopie wie diese:

```
$ svn checkout http://svn.example.com/repos/calc
A    calc/Makefile
A    calc/integer.c
A    calc/button.c
Ausgecheckt, Revision 56.

$ ls -A calc
Makefile  button.c integer.c .svn/
```

Die Liste der As am linken Rand zeigt an, dass Subversion Ihrer Arbeitskopie eine Anzahl von Objekten hinzufügt (Add). Sie haben nun eine persönliche Kopie des Verzeichnisses `/calc` im Projektarchiv, mit einem zusätzlichen Eintrag – `.svn` – das, wie bereits erwähnt, die besonderen Informationen enthält, die Subversion benötigt.

Angenommen, Sie nehmen Änderungen an `button.c` vor. Da sich das Verzeichnis `.svn` den ursprünglichen Änderungszeitpunkt und den Inhalt der Datei merkt, kann Subversion erkennen, dass Sie die Datei verändert haben. Trotzdem veröffentlicht Subversion Ihre Änderungen solange nicht, bis Sie es ausdrücklich hierzu auffordern. Der Vorgang des Veröffentlichens von Änderungen über das Projektarchiv ist gemeinhin bekannter als *commit* (oder *check in*).

Um Ihre Änderungen anderen gegenüber zu veröffentlichen, können Sie den Subversion-Befehl **svn commit** verwenden:

```
$ svn commit button.c -m "Tippfehler in button.c korrigiert"
Sende          button.c
Übertrage Daten .
Revision 6 übertragen.
```

Nun sind Ihre Änderungen an `button.c` dem Projektarchiv überstellt, mitsamt einer Notiz, die Ihre Änderung beschreibt (nämlich, dass Sie einen Tippfehler beseitigt haben). Wenn eine andere Benutzerin eine Arbeitskopie von `/calc` auscheckt, wird sie Ihre Änderungen in der letzten Version der Datei sehen können.

angenommen, Sie haben eine Mitarbeiterin, Sally, die eine Arbeitskopie von `/calc` gleichzeitig mit Ihnen ausgecheckt hat. Wenn Sie Ihre Änderung an `button.c` übertragen, bleibt Sallys Arbeitskopie unverändert; Subversion ändert Arbeitskopien nur auf Wunsch des Benutzers.

Um ihr Projekt auf den neuesten Stand zu bringen, kann Sally Subversion dazu auffordern, ihre Arbeitskopie zu aktualisieren, indem sie den Befehl **svn update** verwendet. Das bringt sowohl Ihre als auch alle anderen Änderungen die übertragen wurden seit sie ausgecheckt hatte in ihre Arbeitskopie.

```
$ pwd
/home/sally/calc

$ ls -A
Makefile button.c integer.c .svn/

$ svn update
U    button.c
Aktualisiert zu Revision 57.
```

Die Ausgabe des **svn update** Befehls zeigt, dass Subversion den Inhalt von `button.c` aktualisiert hat (Update). Beachten Sie, dass Sally nicht angeben musste, welche Dateien zu aktualisieren sind; Subversion benutzt die Informationen aus dem `.svn` Verzeichnis und darüber hinaus weitere Informationen im Projektarchiv, um zu entscheiden, welche Dateien auf den neuesten Stand gebracht werden müssen.

Arbeitskopien mit gemischten Revisionen

Als allgemeingültiges Prinzip versucht Subversion, so flexibel wie möglich zu sein. Eine besondere Ausprägung der Flexibilität ist die Fähigkeit, eine Arbeitskopie bestehend aus Dateien und Verzeichnissen mit einer Mischung unterschiedlicher Revisionsnummern zu haben. Subversions Arbeitskopien entsprechen nicht jederzeit einer einzigen Revision des Projektarchivs; sie können Dateien aus mehreren unterschiedlichen Revisionen enthalten. Nehmen wir z.B. an, Sie checken sich eine Arbeitskopie einer Datei aus einem Projektarchiv aus, deren neueste Revision 4 ist:

```
calc/
  Makefile:4
  integer.c:4
  button.c:4
```

In diesem Augenblick entspricht Ihre Arbeitskopie exakt der Revision im Projektarchiv. Sie machen jetzt allerdings eine Änderung an `button.c` und bringen diese Änderung mit einer Übertragung ins Projektarchiv. Angenommen, dass keine weiteren Übertragungen vorgenommen wurden, wird Ihre Übertragung die Revision 5 im Projektarchiv erzeugen, und Ihre Arbeitskopie sieht so aus:

```
calc/
  Makefile:4
  integer.c:4
  button.c:5
```

Angenommen, zu diesem Zeitpunkt macht Sally eine Übertragung für eine Änderung an `integer.c` und erzeugt Revision 6. Wenn Sie **svn update** verwenden, um Ihre Arbeitskopie zu aktualisieren, sieht sie so aus:

```
calc/
  Makefile:6
  integer.c:6
  button.c:6
```

Sallys Änderung an `integer.c` erscheint in Ihrer Arbeitskopie, und Ihre Änderung ist immer noch in `button.c`. In diesem Beispiel ist der Text von `Makefile` in den Revisionen 4, 5 und 6 identisch, jedoch markiert Subversion die Arbeitskopie von `Makefile` mit Revision 6, um zu zeigen, dass es noch aktuell ist. Wenn Sie also ein sauberes Update von der Wurzel Ihrer Arbeitskopie her machen, sollte sie im Allgemeinen genau einer Revision im Projektarchiv entsprechen.

Aktualisierungen und Übertragungen sind getrennt

Eine der grundlegenden Regeln von Subversion ist, dass eine Aktion, die in das Projektarchiv schreibt keine Aktion zur Folge hat, die aus dem Projektarchiv liest und umgekehrt. Wenn Sie bereit sind, neue Änderungen an das Projektarchiv zu übergeben, heißt das noch lange nicht, dass Sie auch die Änderungen anderer haben möchten. Und wenn Sie noch an Änderungen arbeiten, sollte **svn update** elegant die Änderungen aus dem Projektarchiv mit Ihren Änderungen zusammenführen anstatt Sie dazu zu zwingen, Ihre Änderungen zu veröffentlichen.

Der hauptsächliche Nebeneffekt dieser Regel ist, dass eine Arbeitskopie zusätzlich buchhalten muss, um sowohl gemischte Revisionen zu verfolgen als auch diese Mischung vertragen zu können. Die Tatsache, dass auch Verzeichnisse selbst versioniert sind, verkompliziert die Sache nur.

Nehmen wir zum Beispiel an, Ihre Arbeitskopie besteht komplett aus Revision 10. Sie bearbeiten die Datei `foo.html` und führen ein **svn commit** aus, das die Revision 15 im Projektarchiv erzeugt. Nach der erfolgreichen Übertragung würden viele neue Benutzer erwarten, dass die gesamte Arbeitskopie auf Revision 15 stehe, was aber nicht der Fall ist! Alle möglichen Änderungen können sich zwischen Revision 10 und 15 im Projektarchiv zugetragen haben. Der Client weiß nichts über diese Änderungen im Projektarchiv, da Sie noch nicht **svn update** aufgerufen haben, und **svn commit** zieht keine Änderungen herein. Wenn andererseits **svn commit** automatisch Änderungen hereinziehen würde, könnte die gesamte Arbeitskopie auf Revision 15 gebracht werden – doch dann wäre die grundlegende Regel verletzt, dass Lesen und Schreiben getrennte Aktionen sind. Deshalb ist das einzig Sichere, das der Subversion-Client tun kann, die eine Datei – `foo.html` – als zur Revision 15 gehörig zu kennzeichnen. Der Rest der Arbeitskopie verbleibt bei Revision 10. Nur durch **svn update** können die neuesten Änderungen hereingezogen und die gesamte Arbeitskopie als Revision 15 gekennzeichnet werden.

Gemischte Revisionen sind normal

Tatsache ist, dass *jedes Mal* wenn Sie **svn commit** aufgerufen haben, die Arbeitskopie aus irgendeiner Mischung von Revisionen besteht. Die Sachen, die Sie eben ins Projektarchiv gebracht haben, werden mit höheren Revisionsnummern gekennzeichnet als alles andere. Nach einigen Übertragungen (ohne zwischenzeitliche Updates) ist Ihre Arbeitskopie eine

Riesenmischung von Revisionen. Selbst wenn Sie die einzige Person sind, die das Projektarchiv benutzt, werden sie dieses Phänomen bemerken. Um Ihre Mischung aus Arbeitsrevisionen untersuchen zu können, verwenden Sie den Befehl **svn status** mit der Option `--verbose` (`-v`; siehe „Verschaffen Sie sich einen Überblick über Ihre Änderungen“ für weitergehende Informationen).

Oft ist neuen Benutzern überhaupt nicht bewusst, das ihre Arbeitskopie gemischte Revisionen beinhaltet. Das kann zur Verwirrung führen, weil viele Client-Programme empfindlich auf die Revision des Objektes reagieren, das sie untersuchen. Beispielsweise wird der **svn log**-Befehl verwendet, um die Historie der Änderungen einer Datei oder eines Verzeichnisses darzustellen (siehe „Erzeugung einer Liste der Änderungsgeschichte“). Wenn der Benutzer diesen Befehl auf ein Objekt in der Arbeitskopie anwendet, erwartet er, die gesamte Historie des Objektes zu sehen. Wenn jedoch die Arbeitsrevision des Objektes ziemlich alt ist (oftmals weil lange Zeit kein **svn update** aufgerufen wurde), wird die Historie der *älteren* Version des Objekts angezeigt.

Gemischte Revisionen sind nützlich

Wenn Ihr Projekt hinreichend komplex ist, werden Sie entdecken, dass es manchmal ganz nett sein kann, Teile Ihrer Arbeitskopie *zurückzudatieren* (oder auf eine ältere Version als die vorliegende zu aktualisieren); wie das gemacht wird, wird in Kapitel 2, *Grundlegende Benutzung* gezeigt. Vielleicht möchten Sie eine ältere Version eines Teilmoduls in einem Unterverzeichnis testen, oder Sie möchten herausbekommen, wann ein Fehler das erste Mal in einer Datei auftauchte. Dies ist der „Zeitmaschinen“-Aspekt eines Versionskontrollsystems – die Eigenschaft, die es ermöglicht, irgendeinen Teil Ihrer Arbeitskopie zeitlich nach vorne oder nach hinten zu verschieben.

Gemischte Revisionen haben ihre Grenzen

Wie auch immer Sie gemischte Revisionen in Ihrer Arbeitskopie verwenden, diese Flexibilität hat ihre Grenzen.

Erstens kann die Löschung einer Datei oder eines Verzeichnisses nicht an das Projektarchiv übergeben werden, wenn die Datei oder das Verzeichnis nicht ganz aktuell ist. Falls eine neuere Version im Projektarchiv existiert, wird Ihr Löschversuch abgelehnt, um zu vermeiden, dass Sie versehentlich Änderungen löschen, die Sie noch nicht gesehen haben.

Zweitens können Sie keine Änderungen an Metadaten eines Verzeichnisses an das Projektarchiv übergeben, wenn das Verzeichnis nicht ganz aktuell ist. In Kapitel 3, *Fortgeschrittene Themen* werden Sie lernen, wie man „Eigenschaften“ an Objekte hängt. Die Arbeitskopie eines Verzeichnisses definiert eine bestimmte Menge von Einträgen und Eigenschaften, so dass eine Eigenschafts-Änderung an einem veralteten Verzeichnis Eigenschaften zerstören kann, die Sie noch nicht gesehen haben.

Zusammenfassung

In diesem Kapitel haben wir eine Anzahl fundamentaler Konzepte von Subversion behandelt:

- Wir haben die Begriffe zentrales Projektarchiv, Arbeitskopie und Reihe von Revisionsbäumen des Projektarchivs eingeführt.
- Wir haben einige einfache Beispiele gesehen, wie zwei Mitarbeiter Subversion verwenden können, um gegenseitig Änderungen auszutauschen, indem das „kopieren-verändern-zusammenführen“-Modell benutzt wird.
- Wir haben ein wenig darüber geredet, wie Subversion Informationen in einer Arbeitskopie verfolgt und verwaltet.

An dieser Stelle sollten Sie eine gute Vorstellung haben, wie Subversion ganz allgemein arbeitet. Mit diesem Kenntnisstand sollten Sie in der Lage sein, das nächste Kapitel anzugehen, das ein detaillierter Rundgang durch die Befehle und Eigenschaften von Subversion ist.

Kapitel 2. Grundlegende Benutzung

Theorie ist nützlich, doch deren Anwendung ist der pure Spaß. Lassen Sie uns nun zu den Details von Subversion kommen. Wenn Sie das Ende dieses Kapitels erreicht haben, werden Sie in der Lage sein, alle Aufgaben zu erledigen, die sich bei der normalen täglichen Arbeit mit Subversion stellen. Sie werden damit beginnen, Ihre Dateien in Subversion einzupflegen, gefolgt von einem initialen Checkout Ihres Codes. Dann werden wir bei unserem Rundgang zeigen, wie Änderungen gemacht und diese Änderungen untersucht werden. Sie werden auch sehen, wie Sie die Änderungen anderer in Ihre Arbeitskopie bringen, untersuchen, und sich durch eventuell auftretende Konflikte arbeiten können.

Dieses Kapitel ist nicht als erschöpfende Liste aller Befehle von Subversion gedacht ist – es ist eher eine Einführung in die gebräuchlichsten Aufgaben von Subversion, denen Sie begegnen werden. Dieses Kapitel setzt voraus, dass Sie Kapitel 1, *Grundlegende Konzepte* gelesen und verstanden haben und dass Sie mit dem allgemeinen Subversion-Modell vertraut sind. Für eine vollständige Referenz aller Befehle, siehe Kapitel 9, *Die vollständige Subversion Referenz*.

Dieses Kapitel geht weiter davon aus, dass der Leser Informationen sucht, wie er auf grundlegende Art mit einem bestehenden Subversion-Projektarchiv interagieren kann. Kein Projektarchiv bedeutet, keine Arbeitskopie; keine Arbeitskopie bedeutet, nicht viel Interesse an diesem Kapitel. Es gibt viele Orte im Internet, die freie oder preiswerte Bewirtungsdienste für Subversion-Projektarchive anbieten. Oder, falls Sie es bevorzugen sollten, Ihre eigenen Projektarchive einzurichten und zu verwalten, schauen Sie sich Kapitel 5, *Verwaltung des Projektarchivs* an. Erwarten Sie aber nicht, dass die Beispiele in diesem Kapitel funktionieren, ohne dass der Anwender Zugriff auf ein Subversion-Projektarchiv hat.

Zum Schluss sei noch gesagt, dass jede Subversion-Operation, die über ein Netzwerk mit dem Projektarchiv Kontakt aufnimmt, möglicherweise erfordert, dass sich der Anwender authentifiziert. Der Einfachheit halber vermeiden unsere Beispiele über das gesamte Kapitel hinweg die Darstellung und Erörterung der Authentifizierung. Beachten Sie, dass sie wahrscheinlich dazu gezwungen werden, dem Server zumindest einen Anwendernamen und ein Passwort anzugeben, falls Sie das hier erlangte Wissen auf einer echten Subversion-Instanz anwenden möchten. Für eine detaillierte Beschreibung, wie Subversion Authentifizierung und Client-Berechtigungsnachweise behandelt, siehe „Client-Zugangsdaten“

Hilfe!

Es bedarf keiner Erwähnung, dass dieses Buch existiert, um als Quelle für Information und Hilfe neuen und alten Subversion-Anwendern zu dienen. Allerdings ist die Kommandozeile von Subversion praktischerweise selbst-dokumentierend, was die Notwendigkeit herabsetzt, das Buch aus dem (hölzernen, virtuellen oder sonstigen) Regal hervorzuholen. Der Befehl **svn help** ist Ihr Einstieg zu dieser eingebauten Dokumentation:

```
$ svn help
Aufruf: svn <Unterbefehl> [Optionen] [Parameter]
Subversion-Kommandozeilenclient, Version 1.6.13.
Geben Sie »svn help <Unterbefehl>« ein, um Hilfe zu einem Unterbefehl
          zu erhalten.
Geben Sie »svn --version« ein, um die Programmversion und die Zugriffsmodule
     oder »svn --version --quiet«, um nur die Versionsnummer zu sehen.

Die meisten Unterbefehle akzeptieren Datei- und/oder Verzeichnisparameter,
wobei die Verzeichnisse rekursiv durchlaufen werden. Wenn keine Parameter
angegeben werden, durchläuft der Befehl das aktuelle Verzeichnis rekursiv.

Verfügbare Unterbefehle:
   add
   blame (praise, annotate, ann)
   cat
…
```

Wie in der vorangegangenen Ausgabe beschrieben, bekommen Sie Hilfe zu einem bestimmten Unterbefehl mit **`svn help`** ***`UNTERBEFEHL`***. Subversion antwortet mit der vollständigen Aufrufbeschreibung für diesen Unterbefehl mitsamt seiner Syntax, den Optionen und dem Verhalten:

```
$ svn help help
help (?, h): Beschreibt die Anwendung dieses Programms und seiner Unterbefehle.
Aufruf: help [UNTERBEFEHL...]

Globale Optionen:
  --username PAR           : Benutzername PAR angeben
  --password PAR           : Passwort PAR angeben
…
```

Optionen und Schalter und Flags, oh Mann!

Der Subversion-Kommandozeilen-Client besitzt zahlreiche Befehlsmodifizierer. Manche Leute nennen diese Dinge „Schalter“ oder „Flags“ – in diesem Buch nennen wie sie „Optionen“. Sie finden die von einem **svn**-Unterbefehl unterstützten Optionen, neben einer Reihe von Optionen, die global von allen Unterbefehlen unterstützt werden, am Ende der eingebauten Aufrufbeschreibung für diesen Unterbefehl.

Die Optionen von Subversion haben zwei unterschiedliche Formen: Kurzoptionen bestehen aus einem Bindestrich gefolgt von einem einzelnen Buchstaben, und Langoptionen bestehen aus zwei Bindestrichen gefolgt von mehreren Buchstaben und Bindestrichen (z.B. `-s` bzw. `--dies-ist-eine-langoption`). Jede Option besitzt mindestens ein Langformat. Einige, etwa die Option `--changelist`, verfügen über einen abgekürzten Alias im Langformat (hier `--cl`). Nur bestimmte Optionen, typischerweise die am meisten gebrauchten, haben zusätzlich ein Kurzformat. Um die Klarheit in diesem Buch zu bewahren, benutzen wir in den Beispielen die Langform, doch wenn Optionen beschrieben werden, die eine Kurzform besitzen, nennen wir sowohl die Langform (der Klarheit wegen) als auch die Kurzform (um sie sich leichter merken zu können). Sie sollten die Form benutzen, mit der Sie am besten zurechtkommen, jedoch versuchen Sie nicht, beide gleichzeitig zu verwenden.

Viele Unix-basierte Distributionen von Subversion enthalten Handbuchseiten, die mit dem Programm **man** aufgerufen werden können, doch jene beinhalten in der Regel nur Verweise auf andere Quellen eigentlicher Hilfe, etwa die Webpräsenz des Projektes und derjenigen, die dieses Buch bewirtet. Daneben bieten mehrere Firmen Hilfe und Unterstützung für Subversion an, üblicherweise als Mischung von webbasierten Diskussionsforen und entgeltlicher Beratung. Und selbstverständlich bietet das Internet einen Hort von einem Jahrzehnt an Diskussionen zum Thema Subversion, der nur darauf wartet, von Ihrer Lieblings-Suchmaschine geborgen zu werden. Hilfe zu Subversion ist nie zu weit weg.

Wie Sie Daten in Ihr Projektarchiv bekommen

Sie können neue Dateien auf zweierlei Weisen in das Subversion-Projektarchiv bekommen: **svn import** und **svn add**. Wir werden **svn import** jetzt und **svn add** später in diesem Kapitel besprechen, wenn wir einen typischen Tag mit Subversion durchnehmen.

Importieren von Dateien und Verzeichnissen

Mit dem **svn import**-Befehl kann ein unversionierter Verzeichnisbaum schnell in ein Projektarchiv kopiert werden, wobei benötigte Zwischenverzeichnisse nach Bedarf angelegt werden. **svn import** erfordert keine Arbeitskopie und pflegt Ihre Dateien sofort in das Projektarchiv ein. Typischerweise verwenden Sie diesen Befehl, wenn bereits ein Verzeichnisbaum besteht, den Sie aber in einem Subversion-Projektarchiv pflegen möchten. Zum Beispiel:

```
$ svn import mytree file:///var/svn/newrepos/some/project \
             http://svn.example.com/svn/repo/some/project \
             -m "Erstimport"
Hinzufügen     mytree/foo.c
Hinzufügen     mytree/bar.c
Hinzufügen     mytree/subdir
Hinzufügen     mytree/subdir/quux.h

Revision 1 übertragen.
```

Im vorstehenden Beispiel wurde der Inhalt des Verzeichnisses `mytree` in das Verzeichnis `some/project` des Projektarchivs abgelegt. Beachten Sie, dass Sie dieses neue Verzeichnis nicht erst anlegen mussten – **svn import** erledigt das für Sie. Unmittelbar nach der Übergabe können Sie Ihre Daten im Projektarchiv sehen:

```
$ svn list http://svn.example.com/svn/repo/some/project
bar.c
foo.c
subdir/
$
```

Beachten Sie, dass nach dem Import das ursprüngliche Verzeichnis *nicht* in eine Arbeitskopie umgewandelt wird. Um auf diesen Daten auf eine versionierte Art und Weise arbeiten zu können, müssen Sie noch eine Arbeitskopie aus diesem Baum erzeugen.

Empfohlene Aufteilung des Projektarchivs

Subversion bietet äußerste Flexibilität, was die Anordnung Ihrer Daten betrifft. Da es einfach Verzeichnisse und Dateien versioniert und keinem dieser Objekte eine bestimmte Bedeutung zuschreibt, können Sie die Daten in Ihrem Projektarchiv auf beliebige Weise anordnen. Leider bedeutet diese Flexibilität auch, dass Sie sich „ohne Karte leicht verirren“ können, wenn Sie versuchen, sich in verschiedenen Subversion-Projektarchiven zurechtzufinden, die eine komplett unterschiedliche und unvorhersehbare Anordnung der in ihnen vorhandenen Daten haben.

Um dieser Verwirrung entgegenzuwirken, empfehlen wir Ihnen, dass Sie einer Konvention zur Gestaltung des Projektarchivs (bereits vor langer Zeit eingeführt, während der Entstehung der Subversion-Projektes) folgen, in der eine handvoll strategisch benannter Verzeichnisse des Subversion-Projektarchivs eine nützliche Aussage über die sich darin befindlichen Daten machen. Die meisten Projekte besitzen eine erkennbare „Hauptlinie“ oder *Trunk* der Entwicklung, einige „Zweige“ (engl. *branches*), die abweichende Kopien von Entwicklungslinien darstellen und einige *Tags*, die benannte stabile Momentaufnahmen einer bestimmten Entwicklungslinie sind. Also empfehlen wir zunächst, dass jedes Projekt eine erkennbare *Projektwurzel* im Projektarchiv hat, ein Verzeichnis unter dem sich die gesamte versionierte Information des Projektes befindet, und zwar nur dieses Projektes. Zweitens schlagen wir vor, dass jede Projektwurzel ein Unterverzeichnis `trunk` für die Hauptentwicklungslinie hat, ein Unterverzeichnis `branches`, in dem bestimmte Zweige (oder Sammlungen von Zweigen) erstellt werden und ein Unterverzeichnis `tags`, in dem bestimmte Tags (oder Sammlungen von Tags) angelegt werden. Selbstverständlich kann die Wurzel des Projektarchivs auch als Projektwurzel dienen, falls das Projektarchiv nur ein einziges Projekt beheimatet.

Hier sind ein paar Beispiele:

```
$ svn list file:///var/svn/single-project-repo
trunk/
branches/
tags/
$ svn list file:///var/svn/multi-project-repo
project-A/
project-B/
$ svn list file:///var/svn/multi-project-repo/project-A
trunk/
branches/
tags/
$
```

Wir werden in Kapitel 4, *Verzweigen und Zusammenführen* viel mehr über Tags und Zweige reden. Details und Ratschläge zum Einrichten von Projektarchiven bei mehreren Projekten finden Sie in „Aufbau des Projektarchivs“. Schließlich erörtern wir Projektwurzeln näher in „Planung der Organisation Ihres Projektarchivs“.

Was steckt in einem Namen?

Subversion gibt sich alle Mühe, nicht die Art der Daten einzuschränken, die Sie unter Versionskontrolle setzen können. Der Inhalt von Dateien und Werte von Eigenschaften werden als binäre Daten gespeichert und übermittelt, und „Datei-Inhalts-Typ“ sagt Ihnen, wie Sie Subversion darauf hinweisen, dass „Text“-Operationen für eine bestimmte Datei keinen Sinn ergeben. Trotzdem gibt es einige wenige Stellen, an denen Subversion Einschränkungen für gespeicherte Informationen vorsieht.

Subversion behandelt intern bestimmte Dateneinheiten – z.B. Namen von Eigenschaften, Pfadnamen und Protokollmitteilungen – als UTF-8-kodiertes Unicode. Das heißt aber nicht, dass all Ihre Interaktionen mit Subversion in UTF-8 erfolgen müssen. Im Allgemeinen werden Subversion-Clients die Umwandlungen zwischen UTF-8 und dem auf Ihrem Computer verwendeten Kodiersystem großzügig und transparent vornehmen, sofern eine solche Umwandlung sinnvollerweise durchgeführt werden kann (was bei den meisten gebräuchlichsten Kodierungen heutzutage der Fall ist).

Darüber hinaus werden Pfadnamen sowohl bei WebDAV-Übertragungen als auch in einigen der Steuerdateien von Subversion als XML-Attributwerte verwendet. Das bedeutet, dass Pfadnamen nur aus zulässigen XML (1.0) Zeichen bestehen dürfen. Subversion verbietet ebenfalls `TAB`-, `CR`- und `LF`-Zeichen in Pfadnamen, um zu verhindern, das Pfade in Vergleichen oder bei Befehlsausgaben, wie **svn log** oder **svn status** zerrissen werden.

Obwohl es sich anhört, als müsse man sich eine Menge merken, sind diese Einschränkungen selten ein Problem. Solange Ihre Locale-Einstellungen kompatibel zu UTF-8 sind und Sie keine Kontrollzeichen in den Pfadnamen verwenden, sollten Sie keine Probleme habe, mit Subversion zu kommunizieren. Der Kommandozeilen-Client bietet Ihnen noch ein wenig Extrahilfe – um „korrekte“ Versionen für den internen Gebrauch zu erzeugen, maskiert er bei Bedarf automatisch illegale Zeichen in URL-Pfaden, die Sie eingeben.

Erstellen einer Arbeitskopie

In den meisten Fällen werden Sie ein Subversion-Projektarchiv zu nutzen beginnen, indem Sie einen *Checkout* Ihres Projektes vornehmen. Das Auschecken eines Verzeichnisses aus dem Projektarchivs erzeugt eine Arbeitskopie dieses Verzeichnisses auf Ihrem lokalen Rechner. Falls nicht anderweitig angegeben, enthält diese Kopie die jüngste (d.h. zuletzt erzeugte oder geänderte) im Subversion-Projektarchiv aufgefundene Version des Verzeichnisses und seiner Kinder:

```
$ svn checkout http://svn.example.com/svn/repo/trunk
A    trunk/README
A    trunk/INSTALL
A    trunk/src/main.c
A    trunk/src/header.h
…
Ausgecheckt, Revision 8810.
```

Obwohl im vorangehenden Beispiel das Trunk-Verzeichnis ausgecheckt wird, können Sie ebenso leicht irgendein tiefer befindliches Unterverzeichnis aus einem Projektarchiv auschecken, indem Sie den URL dieses Unterverzeichnisses als URL für den Checkout angeben:

```
$ svn checkout http://svn.example.com/svn/repo/trunk/src
A    src/main.c
A    src/header.h
A    src/lib/helpers.c
…
Ausgecheckt, Revision 8810.
$
```

Da Subversion ein *Kopieren-Ändern-Zusammenführen*-Modell (copy-modify-merge model) statt eines *Sperren-Ändern-Entsperren*-Modells (lock-modify-unlock) verwendet (siehe „Versionierungsmodelle“), können Sie sofort damit beginnen, Änderungen an den Dateien und Verzeichnissen Ihrer Arbeitskopie vorzunehmen. Ihre Arbeitskopie ist wie jede beliebige andere Ansammlung aus Dateien und Verzeichnissen auf Ihrem System. Sie können die Dateien darin bearbeiten, umbenennen, sogar die komplette Arbeitskopie löschen und vergessen.

Obwohl sich Ihre Arbeitskopie „wie jede beliebige andere Ansammlung aus Dateien und Verzeichnissen auf Ihrem System“ verhält, können Sie zwar beliebig Dateien editieren, doch Sie müssen Subversion über *alles andere* was Sie tun in Kenntnis setzen. Wenn Sie z.B. ein Objekt in der Arbeitskopie kopieren oder verschieben möchten, sollten Sie **svn copy** oder **svn move** verwenden statt der Kopier- oder Verschiebebefehle Ihres Betriebssystems. Wir werden darauf später im Kapitel näher eingehen.

Sofern Sie nicht bereit sind, das Hinzufügen einer neuen Datei oder eines neuen Verzeichnisses oder Änderungen an bestehenden Objekten an das Projektarchiv zu übergeben, besteht keine Notwendigkeit, dem Subversion-Server mitzuteilen, dass Sie irgendetwas gemacht haben.

Was hat es mit dem Verzeichnis .svn auf sich?

Jedes Verzeichnis in einer Arbeitskopie beinhaltet einen Verwaltungsbereich – ein Verzeichnis namens `.svn`. Normalerweise wird dieses Unterverzeichnis nicht vom Befehl zum Auflisten des Verzeichnisinhaltes angezeigt, trotzdem ist es ein wichtiges Verzeichnis. Egal, was Sie machen, löschen oder ändern Sie nichts im Verwaltungsbereich! Subversion verwendet jenes Verzeichnis und seinen Inhalt, um Ihre Arbeitskopie zu verwalten.

Sollten Sie versehentlich das Verzeichnis `.svn` löschen, besteht die einfachste Lösung des Problems darin, das komplette darüber liegende Verzeichnis zu löschen (mit dem Betriebssystem-Löschbefehl, nicht mit **svn delete**) und dann wiederum **`svn update`** von einem darüber liegenden Verzeichnis aus aufzurufen. Der Subversion-Client wird das von Ihnen gelöschte Verzeichnis herunterladen und dabei auch einen neuen `.svn`-Bereich anlegen.

Beachten Sie, dass in den vorangegangenen Beispielen Subversion eine Arbeitskopie in einem Verzeichnis angelegt hat, dessen Name der letzten Komponente des Checkout-URLs entsprach. Das passiert nur aus reiner Anwenderfreundlichkeit, wenn dem Befehl **svn checkout** lediglich der URL zum Auschecken übergeben wird. Der Kommandozeilen-Client von Subversion bietet Ihnen dennoch zusätzlich die Möglichkeit, den Namen des lokalen Verzeichnisses anzugeben, den Subversion zum Anlegen der Arbeitskopie verwenden soll. Zum Beispiel:

```
$ svn checkout http://svn.example.com/svn/repo/trunk my-working-copy
A    my-working-copy/README
A    my-working-copy/INSTALL
A    my-working-copy/src/main.c
A    my-working-copy/src/header.h
…
Ausgecheckt, Revision 8810.
$
```

Falls das von Ihnen angegebene lokale Verzeichnis noch nicht existiert, ist das in Ordnung: **svn checkout** legt es für Sie an.

Der grundlegende Arbeitszyklus

Subversion hat zahlreiche Features, Optionen und noch jede Menge Schnickschnack, aber für die tägliche Arbeit ist die Wahrscheinlichkeit groß, nur wenig davon zu benutzen. In diesem Abschnitt gehen wir durch die gebräuchlichsten Dinge, die Sie während des Tagesgeschäftes mit Subversion machen werden.

Der typische Arbeitszyklus sieht so aus:

1. *Aktualisieren Sie Ihre Arbeitskopie.* Das bedingt die Verwendung des Befehls **svn update**.

2. *Nehmen Sie Ihre Änderungen vor*. Die häufigsten Änderungen, die Sie machen werden, sind Bearbeitungen des Inhalts Ihrer bestehenden Dateien. Doch manchmal müssen Sie Dateien und Verzeichnisse hinzufügen, entfernen und verschieben – die Befehle **svn add**, **svn delete**, **svn copy** sowie **svn move** bewerkstelligen derartige strukturelle Änderungen in der Arbeitskopie.

3. *Überprüfen Sie Ihre Änderungen.* Die Befehle **svn status** und **svn diff** sind entscheidend beim Überprüfen der von Ihnen in der Arbeitskopie vorgenommenen Änderungen.

4. *Beheben Sie Ihre Fehler.* Niemand ist vollkommen, und so kann es passieren, dass Sie beim Überprüfen Ihrer Änderungen etwas entdecken, was nicht richtig ist. Manchmal ist es am einfachsten, einfach erneut von vorne zu beginnen. Der Befehl **svn revert** stellt den ungeänderten Zustand einer Datei oder eines Verzeichnisses wieder her.

5. *Lösen Sie etwaige Konflikte auf (arbeiten Sie die Änderungen anderer ein).* Während der Zeit, die Sie benötigen, um Änderungen vorzunehmen und zu überprüfen, hätten andere ebenfalls Änderungen machen und sie veröffentlichen können. Sie sollten deren Änderungen in Ihre Arbeitskopie integrieren, um Szenarien bedingt durch Veralterung zu vermeiden, die möglicherweise entstehen, wenn Sie Ihre Änderungen veröffentlichen wollen. Auch hier hilft Ihnen der Befehl **svn update** weiter. Sollte das zu lokalen Konflikten führen, müssen Sie jene mithilfe des Befehls **svn resolve** auflösen.

6. *Veröffentlichen (übergeben) Sie Ihre Änderungen.* Der Befehl **svn commit** überträgt Ihre Änderungen in das Projektarchiv, in dem sie, falls sie angenommen werden, die neueste Version aller Dinge erstellen, die Sie modifiziert haben. Nun können auch andere Ihre Arbeit sehen.

Aktualisieren Sie Ihre Arbeitskopie

Wenn Sie in einem Projekt arbeiten, das über mehrere Arbeitskopien modifiziert wird, sollten Sie Ihre Arbeitskopie aktualisieren, damit Sie die Änderungen aus anderen Arbeitskopien seit Ihrer letzten Aktualisierung mitbekommen. Es kann sich dabei um Änderungen handeln, die andere Mitarbeiter aus dem Team gemacht haben, oder aber Änderungen, die Sie selbst von einem anderen Computer gemacht haben. Um Ihre Daten zu schützen, erlaubt es Ihnen Subversion nicht, neue Änderungen auf veraltete Dateien und Verzeichnisse anzuwenden, so dass Sie am besten die neuesten Versionen aller Ihrer Projektdateien und -verzeichnisse haben, bevor Sie selbst irgendwelche Änderungen vornehmen.

Verwenden Sie **svn update**, um Ihre Arbeitskopie mit der letzten Revision im Projektarchiv zu synchronisieren:

```
$ svn update
U    foo.c
U    bar.c
Aktualisiert zu Revision 2.
```

In diesem Fall sieht es so aus, dass jemand Änderungen sowohl an `foo.c` als auch an `bar.c` übergeben hat, seit Sie das letzte Mal aktualisiert haben, und Subversion hat Ihre Arbeitskopie aktualisiert, damit sie beide Änderungen enthält.

Wenn der Server über **svn update** Änderungen an Ihre Arbeitskopie schickt, wird ein Buchstabencode neben jedem Objekt angezeigt, um Ihnen anzuzeigen, was Subversion gemacht hat, um die Arbeitskopie auf den neuesten Stand zu bringen. Zur Bedeutung der Buchstaben, rufen Sie **`svn help update`** auf oder schauen sich svn update (up) an.

Nehmen Sie Ihre Änderungen vor

Nun können Sie loslegen und Änderungen an Ihrer Arbeitskopie vornehmen. Sie können zwei Arten von Änderungen an Ihrer Arbeitskopie machen: *Dateiänderungen* und *Baumänderungen*. Sie brauchen Subversion nicht mitteilen, dass Sie beabsichtigen, eine Datei zu ändern; machen Sie einfach Ihre Änderungen mit Ihrem Editor, Textverarbeitungsprogramm, Grafikprogramm oder was Sie sonst normalerweise benutzen. Subversion merkt automatisch, welche Dateien verändert wurden; darüber hinaus behandelt es binäre Dateien ebenso einfach wie Textdateien – und ebenso effizient. Davon unterscheiden sich Baumänderungen, die Änderungen an der Verzeichnisstruktur nach sich ziehen. Zu solchen Änderungen zählen das Hinzufügen und Entfernen von Dateien, das Umbenennen von Dateien oder Verzeichnissen sowie das Kopieren von Dateien und Verzeichnissen an andere Orte. Für Baumänderungen verwenden Sie Subversion-Operationen, um Dateien und Verzeichnisse zum Löschen, Hinzufügen, Kopieren oder Verschieben „einzuplanen“. Diese Änderungen können sofort in Ihrer Arbeitskopie stattfinden, jedoch wird im Projektarchiv nichts hinzugefügt oder gelöscht bevor Sie die Änderungen übergeben haben.

Versionierung symbolischer Links

Auf Nicht-Windows-Systemen kann Subversion den besonderen Dateityp *symbolischer Link* (oder *Symlink*) versionieren. Ein Symlink ist eine Datei, die sich wie eine transparente Referenz auf ein anderes Objekt im Dateisystem verhält, und somit Programmen die Möglichkeit bietet, indirekt auf dieses Objekt zuzugreifen, indem sie Operationen auf dem Symlink ausführen.

Wenn ein Symlink in Subversion übergeben wird, merkt sich Subversion sowohl, dass die Datei eigentlich ein Symlink ist, als auch das Objekt, auf das der Symlink „zeigt“. Wenn dieser Symlink auf einem Nicht-Windows-System in einer anderen Arbeitskopie ausgecheckt wird, rekonstruiert Subversion aus dem versionierten Symlink einen echten Symlink auf Dateisystemebene. Jedoch beeinträchtigt das keineswegs die Benutzbarkeit von Arbeitskopien auf Systemen wie Windows, die keine Symlinks unterstützen. Auf diesen Systemen erzeugt Subversion einfach eine Textdatei, deren Inhalt der Pfad ist, auf den der ursprüngliche Symlink gezeigt hat. Obwohl diese Datei unter Windows nicht als Symlink verwendet werden kann, hindert es Windows-Benutzer nicht an der Ausübung anderer Tätigkeiten mit Subversion.

Hier ist ein Überblick der fünf Subversion-Unterbefehle, die Sie am häufigsten benutzen werden, um Änderungen am Verzeichnisbaum vorzunehmen:

`svn add FOO`

Verwenden Sie diesen Befehl, um die Datei, das Verzeichnis oder den symbolischen Link `FOO` zum Hinzufügen in das Projektarchiv vormerken. Wenn Sie das nächste Mal übergeben, wird `FOO` ein Kind seines Elternverzeichnisses. Beachten Sie, dass alles unterhalb von `FOO` zum Hinzufügen vorgemerkt wird, falls `FOO` ein Verzeichnis ist. Falls Sie nur `FOO` selber hinzufügen möchten, geben Sie die Option `--depth=empty` an.

`svn delete FOO`

Verwenden Sie das, um die Datei, das Verzeichnis oder den symbolischen Link `FOO` zum Löschen aus dem Projektarchiv vormerken. `FOO` wird sofort aus der Arbeitskopie entfernt, falls es eine Datei oder ein Link ist. Falls `FOO` ein Verzeichnis ist, wird es nicht gelöscht, sondern zum Löschen vorgemerkt. Wenn Sie Ihre Änderungen übergeben, wird das gesamte Verzeichnis `FOO` aus der Arbeitskopie und dem Projektarchiv entfernt. [1]

`svn copy FOO BAR`

Erzeuge ein neues Objekt `BAR` als Duplikat von `FOO` und merke `BAR` automatisch zum Hinzufügen vor. Wird bei der nächsten Übergabe `BAR` dem Projektarchiv hinzugefügt, wird die Historie der Kopie mit aufgezeichnet (so wie sie ursprünglich in `FOO` war). **svn copy** erzeugt keine Zwischenverzeichnisse, sofern nicht die Option `--parents` angegeben wird..

`svn move FOO BAR`

Dieser Befehl macht genau das gleiche wie **`svn copy FOO BAR; svn delete FOO`**. D.h., `BAR` wird zum Hinzufügen als Kopie von `FOO` und `FOO` selbst zum Löschen vorgemerkt. **svn move** erzeugt keine Zwischenverzeichnisse, sofern nicht die Option `--parents` angegeben wird.

`svn mkdir FOO`

Dieser Befehl macht genau das gleiche wie **`mkdir FOO; svn add FOO`**. D.h., ein neues Verzeichnis namens `FOO` wird angelegt und zum Hinzufügen vorgemerkt.

Ändern des Projektarchivs ohne Arbeitskopie

Subversion *bietet* Wege, um Änderungen am Verzeichnisbaum ohne explizite Übergabe unmittelbar in das Projektarchiv zu übertragen. Im einzelnen können besondere Anwendungen von **svn mkdir**, **svn copy**, **svn move** und **svn delete** sowohl direkt auf Projektarchiv-URLs als auch auf Pfaden in der Arbeitskopie arbeiten. Selbstverständlich macht **svn import**, wie bereits erwähnt, immer Änderungen direkt am Projektarchiv.

[1] Selbstverständlich wird nichts jemals vollständig aus dem Projektarchiv gelöscht – lediglich aus seiner `HEAD`-Revision. Sie können weiterhin auf das gelöschte Objekt in früheren Revisionen zugreifen. Sollten Sie den Wunsch haben, das Objekt wiederauferstehen zu lassen, so das es wieder in `HEAD` vorhanden ist, siehe „Zurückholen gelöschter Objekte“.

Es gibt Vor- und Nachteile bei URL-basierten Operationen. Ein offensichtlicher Vorteil davon ist Geschwindigkeit: das Auschecken einer Arbeitskopie, die Sie noch nicht haben, nur um eine scheinbar einfache Aktion durchzuführen, bedeutet unverhältnismäßige Kosten. Ein Nachteil ist, dass Sie im Allgemeinen auf eine einzelne Operation oder einen einzelnen Operationstypen beschränkt sind, wenn Sie direkt auf URLs arbeiten. Schließlich liegt der Hauptvorteil einer Arbeitskopie darin, dass sie sich als eine Art „Bereitstellungsraum“ verwenden lässt. Sie können sicherstellen, dass die Änderungen, die Sie übergeben möchten, im Gesamtzusammenhang Ihres Projektes einen Sinn ergeben, bevor sie tatsächlich in das Projektarchiv übergeben werden. Und selbstverständlich können diese vorbereiteten Änderungen so einfach oder so kompliziert wie notwendig sein, und dennoch bei der Übergabe in einer einzigen neuen Revision münden.

Überprüfen Sie Ihre Änderungen

Sobald Sie mit Ihren Änderungen fertig sind, müssen Sie sie ins Projektarchiv bringen; es ist normalerweise eine gute Idee, sich die Änderungen zuvor noch einmal anzusehen. Dadurch, dass Sie die Änderungen noch einmal begutachten, können Sie eine genauere *Protokollnachricht* schreiben (eine menschenlesbare Beschreibung der übergebenen Änderungen, die neben ihnen im Projektarchiv gespeichert wird). Es könnte auch sein, dass Sie feststellen, versehentlich eine Datei geändert zu haben, und dass Sie vor der Übergabe diese Änderung rückgängig machen müssen. Zusätzlich bietet sich hierbei eine gute Gelegenheit, die Änderungen vor der Veröffentlichung noch einmal genau durchzugehen. Sie können sich mit **svn status** einen Überblick über Ihre Änderungen verschaffen und mit **svn diff** die Änderungen im Detail anzeigen lassen.

Guck mal, Mutti, kein Netzwerk!

Sie können die Befehle **svn status**, **svn diff** und **svn revert** ohne Netzzugriff verwenden, selbst wenn das Projektarchiv im Netz *ist*. Das macht es leicht, Ihre in Arbeit befindlichen Änderungen zu verwalten, wenn Sie irgendwo ohne Netzverbindung sind oder aus anderen Gründen das Projektarchiv über das Netz nicht erreichen können.

Subversion bewerkstelligt das, indem es private Zwischenspeicher der ursprünglichen, unveränderten Versionen jeder versionierten Datei innerhalb des Verwaltungsbereichs der Arbeitskopie vorhält. Das erlaubt es Subversion, lokale Änderungen an diesen Dateien *ohne Netzzugriff* anzuzeigen – und rückgängig zu machen. Darüber hinaus erlaubt dieser Cache („Text-Base“ genannt) Subversion bei einer Übergabe, die lokalen Änderungen des Benutzers als komprimiertes *Delta* (oder „Differenz“) gegenüber der unveränderten Version zum Server zu senden. Diesen Cache zu haben, bedeutet einen riesigen Vorteil – selbst wenn Sie eine schnelle Internet-Verbindung haben, ist es viel schneller, nur die Änderungen an einer Datei statt die vollständige Datei an den Server zu übermitteln.

Verschaffen Sie sich einen Überblick über Ihre Änderungen

Um einen Überblick über Ihre Änderungen zu bekommen, verwenden Sie den Befehl **svn status**. Wahrscheinlich werden Sie den Befehl **svn status** häufiger benutzen als alle anderen Subversion-Befehle.

Da die Ausgabe des Befehls **cvs status** so geräuschvoll war, und **cvs update** nicht nur eine Aktualisierung vornimmt sondern auch den Status Ihrer lokalen Änderungen meldet, haben sich die meisten Anwender von CVS angewöhnt, **cvs update** zum Anzeigen Ihrer Meldungen zu verwenden. In Subversion sind die Aktualisierungs- und Statusmeldefunktionen vollständig getrennt. Zu Details, siehe „Unterscheidung zwischen Status und Update“

Wenn Sie **`svn status`** ohne zusätzliche Argumente ganz oben in Ihrer Arbeitskopie aufrufen, erfasst und meldet es alle Datei- und Verzeichnisbaumänderungen, die Sie gemacht haben.

```
$ svn status
?       scratch.c
A       stuff/loot
A       stuff/loot/new.c
D       stuff/old.c
M       bar.c
$
```

In seinem Standard-Ausgabemodus zeigt **svn status** sieben Spalten mit Zeichen, gefolgt von mehreren Leerzeichen und einem Datei- oder Verzeichnisnamen an. Die erste Spalte gibt Aufschluss über den Zustand einer Datei oder eines Verzeichnisses und/oder des entsprechenden Inhalts. Einige der häufigsten Codes, die **svn status** anzeigt, sind:

`? item`
: Die Datei, das Verzeichnis oder der symbolische Link `item` ist nicht unter Versionskontrolle.

`A item`
: Die Datei, das Verzeichnis oder der symbolische Link `item` ist zum Hinzufügen in das Projektarchiv vorgemerkt.

`C item`
: Die Datei `item` befindet sich in einem Konfliktzustand. D.h., Änderungen, die vom Server bei einer Aktualisierung empfangen wurden, überlappen sich mit lokalen Änderungen, die Sie in Ihrer Arbeitskopie haben (und konnten beim Aktualisieren nicht automatisch aufgelöst werden). Sie müssen den Konflikt auflösen, bevor Sie Ihre Änderungen in das Projektarchiv übergeben können.

`D item`
: Die Datei, das Verzeichnis oder der symbolische Link `item` ist zum Löschen im Projektarchiv vorgemerkt.

`M item`
: Der Inhalt der Datei `item` ist geändert worden.

Wenn Sie einen speziellen Pfad an **svn status** übergeben, bekommen Sie nur Informationen über dieses Objekt:

```
$ svn status stuff/fish.c
D       stuff/fish.c
```

svn status hat auch eine `--verbose`-Option (`-v`), die Ihnen den Zustand *jedes* Objektes in der Arbeitskopie anzeigt, selbst wenn es sich nicht geändert hat:

```
$ svn status -v
M               44        23    sally     README
                44        30    sally     INSTALL
M               44        20    harry     bar.c
                44        18    ira       stuff
                44        35    harry     stuff/trout.c
D               44        19    ira       stuff/fish.c
                44        21    sally     stuff/things
A                0         ?     ?        stuff/things/bloo.h
                44        36    harry     stuff/things/gloo.c
```

Dies ist das „lange Format“ der Ausgabe von **svn status**. Die Buchstaben in der ersten Spalte bedeuten dasselbe wie vorher, jedoch zeigt die zweite Spalte die Arbeitsrevision des Objektes an. Die dritte und vierte Spalte zeigen die Revision der letzten Änderung an und wer es geändert hat.

Keiner der vorangegangenen Aufrufe von **svn status** stellt eine Verbindung zum Projektarchiv her – sie berichten lediglich, das, was über die Objekte in der Arbeitskopie aus den Aufzeichnungen im Verwaltungsbereich der Arbeitskopie hervorgeht, sowie aus den Zeitstempeln und dem Inhalt geänderter Dateien. Manchmal ist es jedoch dienlich, zu sehen, welche der Objekte in Ihrer Arbeitskopie seit Ihrer letzten Aktualisierung im Projektarchiv geändert wurden. Dafür bietet **svn status** die Option `--show-updates` (`-u`), die eine Verbindung zum Projektarchiv herstellt, und Informationen darüber bereitstellt, was nicht mehr aktuell ist:

```
$ svn status -u -v
M      *        44        23    sally     README
```

```
M               44        20    harry     bar.c
      *         44        35    harry     stuff/trout.c
D               44        19    ira       stuff/fish.c
A                0         ?     ?        stuff/things/bloo.h
  Status bezogen auf Revision:   46
```

Beachten Sie die zwei Sternchen im vorangegangenen Beispiel: Wenn Sie an dieser Stelle **`svn update`** aufrufen würden, erhielten Sie Änderungen an `README` und `trout.c`. Das gibt Ihnen einige sehr wichtige Informationen: Da eins dieser Objekte auch von Ihnen lokal geändert wurde (die Datei `README`) müssen Sie vor der Übergabe aktualisieren, um die Änderungen an `README` vom Server mitzubekommen, oder das Projektarchiv wird Ihre Übergabe ablehnen, da sie nicht aktuell ist. Wir werden das später detailliert erörtern

svn status kann viel mehr Informationen über Dateien und Verzeichnisse in Ihrer Arbeitskopie anzeigen als wir hier gezeigt haben – für eine erschöpfende Beschreibung von **svn status** und dessen Ausgabe, rufen Sie **`svn help status`** auf oder schauen Sie unter svn status (stat, st).

Untersuchen Sie die Details Ihrer lokalen Änderungen

Eine andere Möglichkeit, Ihre Änderungen zu untersuchen, ist es, den Befehl **svn diff** zu verwenden, der Unterschiede im Dateiinhalt anzeigt. Wenn Sie **`svn diff`** ganz oben in Ihrer Arbeitskopie ohne Argumente aufrufen, gibt Subversion die von Ihnen gemachten Änderungen an menschenlesbaren Dateien in Ihrer Arbeitskopie aus. Jene Änderungen werden in einem Format namens *unified diff* angezeigt, welches Änderungen als „Brocken“ (oder „Schnipsel“) des Dateiinhalts anzeigt, wobei jeder Textzeile ein Zeichencode vorangestellt wird: ein Leerzeichen, das bedeutet, dass die Zeile nicht geändert wurde, ein Minus (`-`), das bedeutet, dass die Zeile aus der Datei entfernt wurde oder ein Plus (`+`), das bedeutet, dass die Zeile der Datei hinzugefügt wurde. Im Kontext des Befehls **svn diff** zeigen Ihnen diese Minus- und Pluszeilen, wie die Zeilen vor bzw. nach Ihren Änderungen aussahen.

Hier ein Beispiel:

```
$ svn diff
Index: bar.c
===================================================================
--- bar.c (revision 3)
+++ bar.c (working copy)
@@ -1,7 +1,12 @@
+#include <sys/types.h>
+#include <sys/stat.h>
+#include <unistd.h>
+
+#include <stdio.h>

 int main(void) {
-  printf("Sixty-four slices of American Cheese...\n");
+  printf("Sixty-five slices of American Cheese...\n");
 return 0;
 }

Index: README
===================================================================
--- README (revision 3)
+++ README (working copy)
@@ -193,3 +193,4 @@
+Note to self:  pick up laundry.

Index: stuff/fish.c
===================================================================
--- stuff/fish.c (revision 1)
+++ stuff/fish.c (working copy)
-Welcome to the file known as 'fish'.
-Information on fish will be here soon.
```

```
Index: stuff/things/bloo.h
===================================================================
--- stuff/things/bloo.h (revision 8)
+++ stuff/things/bloo.h (working copy)
+Here is a new file to describe
+things about bloo.
```

Der Befehl **svn diff** erzeugt diese Ausgabe, indem er Ihre Arbeitsdateien mit den „unveränderten“ Kopien in der Text-Base vergleicht. Dateien, die zum Hinzufügen vorgemerkt sind, werden vollständig als hinzugefügter Text dargestellt, und Dateien, die zum Löschen vorgemerkt sind, werden vollständig als gelöschter Text dargestellt. Die Ausgabe von **svn diff** ist kompatibel zum Programm **patch**. Das Programm **patch** liest und verwendet *Patch-Dateien* (oder „Patches“), wobei es sich um Dateien handelt, die Unterschiede an einer oder mehreren Dateien beschreiben. Daher können Sie die in Ihrer Arbeitskopie vorgenommenen Änderungen mit jemanden teilen, ohne die Änderungen erst übergeben zu müssen, indem Sie aus der umgeleiteten Ausgabe des Befehls **svn diff** eine Patch-Datei erzeugen:

```
$ svn diff > patchfile
$
```

Subversion verwendet seinen eingebauten diff-Algorithmus, der standardmäßig das unified-diff-Format benutzt. Falls Sie die Ausgabe von diff in einem anderen Format haben möchten, geben Sie ein externes diff-Programm mit der `--diff-cmd`-Option an, und übergeben Sie ihm beliebige Flags mit der Option `--extensions` (`-x`). Sie möchten beispielsweise, dass Subversion die Berechnung und Anzeige der Unterschiede an das GNU-Programm **diff** delegiert, wobei die lokalen Änderungen an der Datei `foo.c` im Kontext-Format ausgegeben (einer anderen Darstellung der Unterschiede) und Unterschiede in Groß- und Kleinschreibung des Dateiinhalts ignoriert werden:

```
$ svn diff --diff-cmd /usr/bin/diff -x "-i" foo.c
…
$
```

Beheben Sie Ihre Fehler

Angenommen, Sie stellen beim Ansehen der Ausgabe von **svn diff** fest, dass alle Änderungen, die Sie an einer bestimmten Datei gemacht haben, fehlerhaft waren. Vielleicht hätten Sie die Datei überhaupt nicht ändern sollen, oder es wäre einfacher, noch einmal bei Null anzufangen. Sie könnten die Datei erneut bearbeiten und alle Änderungen rückgängig machen. Sie könnten versuchen, eine Kopie der Datei im Ursprungszustand zu bekommen und deren Inhalt über Ihre Änderungen zu kopieren. Sie könnten versuchen, diese Änderungen erneut mit **`patch -R`** rückwärts anzuwenden. Es gibt wahrscheinlich noch andere Herangehensweisen, die Sie ausprobieren könnten.

In Subversion bedarf das Rückgängigmachen und bei Null anfangen glücklicherweise nicht derartiger Akrobatik. Verwenden Sie einfach den Befehl **svn revert**:

```
$ svn status README
M       README
$ svn revert README
Rückgängig gemacht: »README«
$ svn status README
$
```

In diesem Beispiel hat Subversion die Datei wieder so her, wie sie vor der Änderung war, indem sie mit der unveränderten Version der Datei aus dem Cache der Text-Base überschrieben wurde. Beachten Sie aber auch, dass **svn revert** *jegliche*

vorgemerkten Operationen rückgängig machen kann – z.B. könnten Sie sich entscheiden, eine neue Datei erst gar nicht hinzufügen zu wollen:

```
$ svn status new-file.txt
?       new-file.txt
$ svn add new-file.txt
A         new-file.txt
$ svn revert new-file.txt
Rückgängig gemacht: »new-file.txt«
$ svn status new-file.txt
?       new-file.txt
$
```

Oder vielleicht haben Sie die Datei versehentlich aus der Versionsverwaltung gelöscht:

```
$ svn status README

$ svn delete README
D         README

$ svn revert README
Rückgängig gemacht: »README«

$ svn status README
$
```

Der Befehl **svn revert** bietet die Rettung für unvollkommene Menschen. Er kann Ihnen jede Menge Zeit und Energie einzusparen helfen, die ansonsten beim manuellen Rückgängigmachen entstanden wäre, oder noch schlimmer, wenn Sie Ihre Arbeitskopie durch eine frische hätten ersetzen müssen, um noch einmal neu anzufangen.

Lösen Sie etwaige Konflikte auf

Wir haben bereits gesehen, wie **`svn status -u`** Konflikte vorhersagen kann, jedoch müssen wir uns noch um diese Konflikte kümmern. Konflikte können jederzeit auftreten, wenn Sie versuchen, Änderungen aus dem Projektarchiv mit Ihrer Arbeitskopie zusammenzuführen oder zu integrieren (in einem sehr allgemeinen Sinn). Bis hierher wissen Sie, dass **svn update** genau diese Art von Szenario hervorruft – der eigentliche Zweck dieses Befehls ist es, Ihre Arbeitskopie mit dem Projektarchiv auf einen Stand zu bringen, indem alle Änderungen seit Ihrer letzten Aktualisierung mit Ihrer Arbeitskopie zusammengeführt werden. Wie teilt Ihnen Subversion nun diese Konflikte mit, und wie gehen Sie damit um?

Angenommen, Sie rufen **svn update** auf und sehen diese interessante Ausgabe:

```
$ svn update
U    INSTALL
G    README
Konflikt in »bar.c« entdeckt.
Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
         (mc) eigene konfliktbehaftete Datei, (tc) fremde konfliktbehaftete Datei,
         (s) alle Optionen anzeigen:
```

Die Codes `U` und `G` sind kein Grund zur Beunruhigung; diese Dateien haben die Änderungen aus dem Projektarchiv sauber aufgenommen. Eine mit `U` markierte Datei enthält keine lokalen Änderungen, wurden jedoch mit Änderungen aus dem Projektarchiv aktualisiert. Eine mit `G` markierte Datei enthielt zwar lokale Änderungen, die Änderungen aus dem Projektarchiv haben sich aber nicht damit überschnitten.

Die nächsten paar Zeilen sind allerdings interessant. Zunächst teilt Ihnen Subversion mit, dass es beim Versuch, ausstehende

Server-Änderungen in die Datei `bar.c` hineinzubringen, festgestellt hat, dass einige dieser Änderungen mit lokalen Änderungen kollidieren, die Sie an dieser Datei in Ihrer Arbeitskopie vorgenommen, jedoch noch nicht übergeben haben. Vielleicht hat jemand dieselbe Textzeile wie Sie geändert. Wie dem auch sei, Subversion markiert diese Datei umgehend als konfliktbehaftet. Dann fragt es Sie, wie Sie mit diesem Problem umgehen möchten, indem es Ihnen interaktiv einen Bearbeitungsschritt zur Auflösung des Konfliktes anbietet. Die am häufigsten verwendeten Optionen werden angezeigt, aber Sie können auch alle Optionen sehen, wenn Sie *s* eintippen:

```
…
Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
         (mc) eigene konfliktbehaftete Datei, (tc) fremde konfliktbehaftete Datei,
         (s) alle Optionen anzeigen: s

  (e)  editieren          - zusammengeführte Datei in einem Editor ändern
  (df) voller Diff        - alle Änderungen in der zusammengeführten Datei anzeigen
  (r)  aufgelöst          - akzeptieren der zusammengeführten Version der Datei
  (dc) Konflikte anzeigen - alle Konflikte anzeigen (die zusammengeführte Version
                            ignorieren)
  (mc) mine-conflict      - eigene Version für alle Konflikte akzeptieren (das selbe)
  (tc) theirs-conflict    - fremde Version für alle Konflikte akzeptieren (das selbe)

  (mf) volle eigene Datei - die eigene Version der kompletten Datei akzeptieren
                            (selbst Nicht-Konflikte)
  (tf) volle fremde Datei - die fremde Version der kompletten Datei akzeptieren
                            (das selbe)
  (p)  zurückstellen      - den Konflikt erst später auflösen
  (l)  starten            - Starten eines externen Programms zur Konfliktauflösung
  (s)  alle anzeigen      - diese Liste anzeigen

Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
         (mc) eigene konfliktbehaftete Datei, (tc) fremde konfliktbehaftete Datei,
         (s) alle Optionen anzeigen:
```

Bevor wir im Detail erklären, was jede Option bedeutet, gehen wir noch mal eben die Optionen durch.

`(e) editieren`
: Die Konfliktdatei im bevorzugten Editor, wie in der Umgebungsvariablen `EDITOR` angegeben, öffnen.

`(df) voller Diff`
: Die Unterschiede zwischen der Basisrevision und der Konfliktdatei im unified-diff-Format anzeigen.

`(r) aufgelöst`
: Nach dem Bearbeiten einer Datei teilen Sie **svn** mit, dass Sie die Konflikte in der Datei aufgelöst haben und der aktuelle Inhalt übernommen werden soll.

`(dc) Konflikte anzeigen`
: Zeigt alle konfliktbehafteten Regionen der Datei an, wobei erfolgreich zusammengeführte Änderungen ignoriert werden.

`(mc) mine-conflict`
: Die neuen vom Server erhaltenen Änderungen verwerfen, die in Konflikt mit Ihren lokalen Änderungen an der zu überprüfenden Datei stehen. Jedoch werden alle nicht in Konflikt stehenden Änderungen vom Server für diese Datei angenommen und zusammengeführt.

`(tc) theirs-conflict`
: Alle lokalen Änderungen an der zu überprüfenden Datei verwerfen, die in Konflikt zu Änderungen vom Server stehen. Jedoch werden alle nicht in Konflikt stehenden lokalen Änderungen an dieser Datei beibehalten.

`(mf) volle eigene Datei`
: Alle neuen vom Server erhaltenen Änderungen für die zu überprüfende Datei verwerfen, aber alle Ihre lokalen Änderungen an dieser Datei beibehalten.

`(tf) volle fremde Datei`

Alle Ihre lokalen Änderungen für die zu überprüfende Datei verwerfen und nur die neuen vom Server erhaltenen Änderungen für diese Datei verwenden.

`(p) zurückstellen`
Die Datei im Konfliktzustand lassen, um nach Abschluss der Aktualisierung die Konfliktauflösung durchzuführen.

`(l) starten`
Ein externes Programm zur Konfliktauflösung starten. Das setzt Vorbereitungen voraus.

`(s) alle anzeigen`
Die Liste aller bei der interaktiven Konfliktauflösung möglichen Befehle anzeigen.

Wir werden diese Befehle nun detaillierter behandeln, wobei sie nach Funktionalität gruppiert werden.

Interaktive Begutachtung der Konflikte

Bevor Sie entscheiden, wie Sie einen Konflikt beseitigen wollen, wollen Sie wahrscheinlich genau sehen, worin der Konflikt besteht. Zwei der bei der interaktiven Aufforderung zur Verfügung stehenden Befehle können Ihnen dabei helfen. Der erste ist der Befehl „voller Diff“ (**df**), der alle lokalen Änderungen an der zu begutachtenden Datei und die in Konflikt stehenden Regionen anzeigt:

```
…
Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
         (mc) eigene konfliktbehaftete Datei, (tc) fremde konfliktbehaftete Datei,
         (s) alle Optionen anzeigen: df
--- .svn/text-base/sandwich.txt.svn-base      Tue Dec 11 21:33:57 2007
+++ .svn/tmp/tempfile.32.tmp     Tue Dec 11 21:34:33 2007
@@ -1 +1,5 @@
-Just buy a sandwich.
+<<<<<<< .mine
+Go pick up a cheesesteak.
+=======
+Bring me a taco!
+>>>>>>> .r32
…
```

Die erste Zeile des diff-Inhalts zeigt den vorherigen Inhalt der Arbeitskopie (die BASE-Revision), die nächste Zeile beinhaltet Ihre Änderung und die letzte Zeile ist die Änderung, die soeben vom Server empfangen worden ist (*gewöhnlich* die HEAD-Revision).

Der zweite Befehl ist ähnlich wie der erste, jedoch zeigt der Befehl „Konflikte anzeigen“ (**dc**) nur die in Konflikt stehenden Regionen an statt aller Änderungen an der Datei. Darüber hinaus verwendet dieser Befehl ein etwas anderes Darstellungsformat für die Konfliktregionen, das es Ihnen auf einfache Weise erlaubt, den Dateiinhalt dieser Regionen in den drei möglichen Zuständen zu vergleichen: original und unbearbeitet, mit Ihren Änderungen, wobei die in Konflikt stehenden Änderungen vom Server ignoriert werden und mit den Änderungen vom Server, wobei Ihre lokalen Änderungen rückgängig gemacht wurden.

Nachdem Sie die durch diesen Befehl bereitgestellten Informationen geprüft haben, können Sie den nächsten Schritt in Angriff nehmen.

Interaktive Konfliktauflösung

Es gibt mehrere unterschiedliche Wege, um Konflikte interaktiv aufzulösen – von denen Ihnen zwei erlauben, Änderungen selektiv zusammenzuführen und zu editieren und die restlichen, die es Ihnen erlauben, einfach eine Version der Datei auszuwählen und weiterzumachen.

Falls Sie eine beliebige Kombination Ihrer lokalen Änderungen auswählen wollen, können Sie den Befehl „editieren“ (**e**) verwenden, um die Datei mit den Konfliktmarken manuell in einem Texteditor (nach den Anweisungen in „Verwendung externer Editoren“ konfiguriert) bearbeiten. Nachdem Sie die Datei bearbeitet haben und mit Ihren Änderungen zufrieden sind, können Sie Subversion mitteilen, das sich die bearbeitete Datei nicht mehr im Konfliktzustand befindet, indem Sie den Befehl

„resolve“ (`r`) benutzen.

Egal, was Ihnen wahrscheinlich Ihr Unix-Snob erzählen wird, die manuelle Bearbeitung der Datei in Ihrem Lieblings-Texteditor ist die Low-Tech Variante zur Beseitigung von Konflikten (siehe „Manuelle Konfliktauflösung“ für die Vorgehensweise). Aus diesem Grund bietet Subversion den Befehl „starten“ (**`l`**) zum Starten eines schicken graphischen Werkzeugs zum Zusammenführen (siehe „External merge“).

Falls Sie entscheiden, dass Sie keine Änderungen zusammenzuführen brauchen, sondern lediglich eine der beiden Dateiversionen akzeptieren wollen, können Sie entweder Ihre Änderungen (auch „meine“) mit dem „mine-full“-Befehl (**`mf`**) oder die der Anderen mit dem „theirs-full“-Befehl (**`tf`**) auswählen.

Zum Schluss gibt es noch ein Paar von Kompromissoptionen. Die Befehle „mine-conflict“ (**`mc`**) und „theirs-conflict“ (**`tc`**) sagen Subversion, es soll Ihre lokalen Änderungen bzw. die Änderungen vom Server zum Auflösen aller Konflikte innerhalb der Datei heranziehen. Im Gegensatz zu den Befehlen „volle eigene Datei“ und „volle fremde Datei“ bewahren beide Befehle jedoch Ihre lokalen Änderungen und die Änderungen vom Server an den Stellen der Datei, an denen keine Konflikte entdeckt wurden.

Aufschieben der Konfliktauflösung

Das hört sich vielleicht an wie ein passender Abschnitt zur Vermeidung von Ehestreitigkeiten, doch es geht immer noch um Subversion; also lesen Sie weiter. Falls Sie eine Aktualisierung vornehmen und ein Konflikt auftaucht, den Sie nicht begutachten oder auflösen können, ermöglicht Ihnen das Eingeben von **`p`** die Konfliktauflösung Datei für Datei aufzuschieben, wenn Sie **`svn update`** aufrufen. Falls Sie bereits vorher wissen, dass Sie aktualisieren wollen, ohne Konflikte aufzulösen, können Sie die Option `--non-interactive` an **svn update** übergeben, und jede Datei mit Konflikten wird automatisch mit einem `C` gekennzeichnet.

Das `C` (für „Conflicted“) bedeutet, dass die Änderungen vom Server sich mit Ihren eigenen überschneiden, und Sie nach Abschluss der Aktualisierung manuell aus den Änderungen wählen müssen. Wenn Sie eine Konfliktauflösung verschieben, macht **svn** typischerweise drei Dinge, um Ihnen bei der Konfliktauflösung zu helfen:

- Subversion gibt ein `C` während der Aktualisierung aus und merkt sich, dass die Datei in einem Konfliktzustand ist.
- Falls Subversion die Datei als geeignet zum Zusammenführen ansieht, fügt es *Konfliktmarken* – besondere Zeichenketten, die die Konfliktregion begrenzen – in die Datei ein, um die überlappenden Bereiche besonders hervorzuheben. (Subversion verwendet die Eigenschaft `svn:mime-type`, um festzustellen, ob sich die Datei kontextuell zeilenweise zusammenführen lässt. Siehe „Datei-Inhalts-Typ“, um mehr zu erfahren.)
- Für jede Datei mit Konflikten stellt Subversion drei zusätzliche unversionierte Dateien in Ihre Arbeitskopie:

  `filename.mine`
  Dies ist Ihre Datei aus der Arbeitskopie bevor Sie die Aktualisierung begannen. Diese Datei beinhaltet Ihre lokalen Änderungen sowie Konfliktmarkierungen. (Falls Subversion diese Datei als nicht-zusammenführbar erachtet, wird die `.mine`-Datei nicht erstellt, da sie identisch mit der Datei der Arbeitskopie wäre.)

  `filename.r`*`OLDREV`*
  Dies ist die Datei, wie sie in der `BASE`-Revision aussah, d.h., die unmodifizierte Revision der Datei in Ihrer Arbeitskopie *bevor* Sie die Aktualisierung begonnen haben, wobei *`OLDREV`* die Nummer der `BASE`-Revision ist.

  `filename.r`*`NEWREV`*
  Dies ist die Datei, die Ihr Subversion-Client soeben durch die Aktualisierung Ihrer Arbeitskopie vom Server erhalten hat, wobei *`NEWREV`* der Revisionsnummer entspricht, auf der Sie aktualisiert haben (`HEAD`, falls nichts anderes angegeben wurde).

Beispielsweise ändert Sally die Datei `sandwich.txt`, übergibt diese Änderungen jedoch noch nicht. In der Zwischenzeit übergibt Harry Änderungen an derselben Datei. Sally aktualisiert Ihre Arbeitskopie vor der Übergabe und bekommt einen Konflikt, den sie verschiebt:

```
$ svn update
Konflikt in »sandwich.txt« entdeckt.
Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
```

```
        (mc) eigene konfliktbehaftete Datei, (tc) fremde konfliktbehaftete Datei
        (s) alle Optionen anzeigen: p
C  sandwich.txt
Aktualisiert zu Revision 2.
Konfliktübersicht:
  Textkonflikte: 1
$ ls -1
sandwich.txt
sandwich.txt.mine
sandwich.txt.r1
sandwich.txt.r2
```

An dieser Stelle erlaubt Subversion Sally *nicht*, die Datei `sandwich.txt` an das Projektarchiv zu übergeben, solange die drei temporären Dateien nicht entfernt werden:

```
$ svn commit -m "Add a few more things"
svn: Übertragen schlug fehl (Details folgen):
svn: Übertragung abgebrochen: »/home/sally/svn-work/sandwich.txt« bleibt im Konflikt
```

Falls Sie eine Konfliktauflösung aufgeschoben haben, müssen Sie den Konflikt auflösen, bevor Ihnen Subversion erlaubt, Ihre Änderungen in das Projektarchiv einzustellen. Sie werden dafür den **svn resolve**-Befehl mit einem von mehreren Argumenten für die `--accept`-Option aufrufen.

Falls Sie die Dateiversion vor Ihren Änderungen haben möchten, wählen Sie das *`base`*-Argument.

Falls Sie die Version möchten, die nur Ihre Änderungen enthält, wählen Sie das *`mine-full`*-Argument.

Falls Sie die Version möchten, die Ihre letzte Aktualisierung vom Server gezogen hat (und somit Ihre Änderungen vollständig verwerfen wollen), wählen Sie das Argument *`theirs-full`*.

Wenn Sie jedoch frei aus Ihren Änderungen und den Änderungen vom Server wählen möchten, führen Sie den konfliktbehafteten Text „händisch“ zusammen (indem Sie die Konfliktmarken in der Datei begutachten und editieren) und wählen das *`working`*-Argument.

svn resolve entfernt die drei temporären Dateien und akzeptiert die Version, die Sie mit der `--accept`-Option angeben. Subversion betrachtet die Datei nun als nicht mehr konfliktbehaftet:

```
$ svn resolve --accept working sandwich.txt
Konflikt von »sandwich.txt« aufgelöst
```

Manuelle Konfliktauflösung

Das manuelle Auflösen von Konflikten kann ganz schön einschüchternd sein, wenn Sie es das erste Mal versuchen; jedoch kann es mit etwas Übung so leicht werden, wie vom Fahrrad zu fallen.

Hier ist ein Beispiel. Aufgrund einer schlechten Absprache bearbeiten Sie und Ihre Mitarbeiterin Sally gleichzeitig die Datei `sandwich.txt`. Sally übergibt ihre Änderungen an das Projektarchiv, und sobald Sie versuchen, Ihre Arbeitskopie zu aktualisieren, erhalten Sie einen Konflikt und müssen `sandwich.txt` bearbeiten, um den Konflikt aufzulösen. Zunächst wollen wir uns die Datei einmal ansehen:

```
$ cat sandwich.txt
Top piece of bread
Mayonnaise
```

```
Lettuce
Tomato
Provolone
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

Die Zeichenketten aus Kleiner-als-Zeichen, Gleichheitszeichen und Größer-als-Zeichen sind Konfliktmarken und gehören nicht zu den eigentlichen Daten, die in Konflikt stehen. Im Allgemeinen werden Sie sicherstellen wollen, dass die Konflikte aus der Datei entfernt werden, bevor sie das nächste Mal eine Übergabe durchführen. Der Text zwischen den ersten beiden Marken besteht aus den Änderungen, die Sie im Konfliktbereich vorgenommen haben:

```
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
```

Der Text zwischen der zweiten und der dritten Marke ist der Text aus Sallys Übergabe:

```
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
```

Für gewöhnlich werden Sie nicht einfach die Konfliktmarken mitsamt der Änderungen von Sally löschen wollen – sie wird furchtbar überrascht sein, wenn das Sandwich kommt und nicht das drauf ist, was sie wollte. Hier ist der Zeitpunkt gekommen, zu dem Sie zum Telefon greifen oder durch das Büro gehen und Sally erklären, dass man in einem italienischen Delikatessenladen kein Sauerkraut bekommt.[2] Sobald Sie sich über die zu übergebenden Änderungen einig sind, können Sie Ihre Datei bearbeiten und die Konfliktmarken entfernen:

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
Salami
Mortadella
Prosciutto
Creole Mustard
Bottom piece of bread
```

Verwenden Sie jetzt **svn resolve**, und Sie sind bereit, Ihre Änderungen an das Projektarchiv zu übergeben:

[2] Und wenn Sie danach fragen, wird man Sie wahrscheinlich auf einer Schiene aus der Stadt tragen.

```
$ svn resolve --accept working sandwich.txt
Konflikt von »sandwich.txt« aufgelöst
$ svn commit -m "Mach weiter mit meinem Sandwich, vergiss Sallys Änderungen."
```

Beachten Sie, dass **svn resolve**, anders als die meisten anderen Befehle, die wir in diesem Kapitel behandeln, erwartet, dass Sie ausdrücklich alle Dateien aufzählen, deren Konflikt Sie beseitigt haben. Auf alle Fälle sollten Sie sorgfältig vorgehen und **svn resolve** nur verwenden, falls Sie sicher sind, den Konflikt in Ihrer Datei beseitigt zu haben – sobald die temporären Dateien entfernt sind, lässt Subversion zu, dass Sie die Datei in das Projektarchiv stellen, selbst wenn sie noch Konfliktmarken enthält.

Falls Sie mal bei der Bearbeitung der konfliktbehafteten Datei verwirrt sein sollten, können Sie jederzeit in den drei Dateien nachsehen, die Subversion für Sie in der Arbeitskopie bereitstellt – dazu gehört auch Ihre Datei vor der Aktualisierung. Sie können sogar ein Zusammenführungs-Werkzeug eines Drittanbieters verwenden, um diese drei Dateien zu untersuchen.

Verwerfen Ihrer Änderungen zugunsten einer aktualisierten Revision aus dem Projektarchiv

Falls Sie einen Konflikt erhalten und entscheiden, dass Sie Ihre Änderungen verwerfen wollen, können Sie **`svn resolve --accept theirs-full`** ***`CONFLICTED-PATH`*** aufrufen, und Subversion wird Ihre Änderungen ignorieren und die temporären Dateien entfernen:

```
$ svn update
Konflikt in »sandwich.txt« entdeckt.
Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
         (mc) eigene konfliktbehaftete Datei, (tc) fremde konfliktbehaftete Datei
         (s) alle Optionen anzeigen: p
C    sandwich.txt
Aktualisiert zu Revision 2.
$ ls sandwich.*
sandwich.txt  sandwich.txt.mine  sandwich.txt.r2  sandwich.txt.r1
$ svn resolve --accept theirs-full sandwich.txt
Konflikt von »sandwich.txt« aufgelöst
$
```

Die Verwendung von svn revert

Falls Sie sich entscheiden, Ihre Änderungen zu verwerfen und erneut mit der Bearbeitung zu beginnen (ob nach einem Konflikt oder sonst zu jeder Zeit), machen Sie einfach Ihre Änderungen rückgängig:

```
$ svn revert sandwich.txt
Rückgängig gemacht: »sandwich.txt«
$ ls sandwich.*
sandwich.txt
$
```

Beachten Sie, dass Sie beim Rückgängigmachen einer konfliktbehafteten Datei nicht **svn resolve** zu verwenden brauchen.

Übergeben Ihrer Änderungen

Endlich! Sie haben die Bearbeitung abgeschlossen, Sie haben alle Änderungen vom Server eingearbeitet, und Sie sind bereit, Ihre Änderungen an das Projektarchiv zu übergeben.

Der Befehl **svn commit** schickt all Ihre Änderungen zum Projektarchiv. Wenn Sie eine Änderung übergeben, müssen Sie einen

Protokolleintrag erstellen, der die Änderung beschreibt. Dieser Eintrag wird mit der von Ihnen erzeugten neuen Revision verknüpft. Wenn Ihr Eintrag kurz ist, können Sie ihn mit der Option `--message` (`-m`) in der Kommandozeile angeben:

```
$ svn commit -m "Anzahl Käsescheiben korrigiert."
Sende          sandwich.txt
Übertrage Daten .
Revision 3 übertragen.
```

Falls Sie jedoch Ihren Protokolleintrag während der Arbeit in irgendeiner Textdatei erstellen möchten, können Sie Subversion mitteilen, sich den Eintrag aus dieser Datei zu holen, indem Sie ihren Namen mit der Option `--file` (`-F`) angeben:

```
$ svn commit -F logmsg
Sende          sandwich.txt
Übertrage Daten .
Revision 4 übertragen.
```

Sollten Sie vergessen, entweder die Option `--message` (`-m`) oder die Option `--file` (`-F`) anzugeben, startet Subversion automatisch Ihren Lieblingseditor (siehe die Information zu `editor-cmd` in „Config“), damit Sie einen Protokolleintrag erstellen können.

Wenn Sie gerade in Ihrem Editor einen Eintrag schreiben und sich entschließen, die Übergabe abzubrechen, können Sie einfach Ihren Editor beenden, ohne die Änderungen zu sichern. Falls Sie den Eintrag bereits gesichert haben sollten, löschen Sie einfach den gesamten Text, sichern Sie erneut und brechen dann ab:

```
$ svn commit

Logmeldung unverändert oder nicht angegeben
A)bbrechen, Weitermac)hen, E)ditieren:
a
$
```

Das Projektarchiv weiß nicht, ob Ihre Änderung im Ganzen einen Sinn ergeben, es ist ihm auch egal; es überprüft lediglich, ob nicht irgendjemand anderes irgendeine derselben Dateien geändert hat wie Sie, als Sie mal weggeschaut haben. Falls jemand das gemacht *hat*, wird die gesamte Übergabe mit einer Meldung fehlschlagen, dass eine oder mehrere Ihrer Dateien nicht mehr aktuell sind:

```
$ svn commit -m "Noch eine Regel hinzufügen"
Sende          rules.txt
svn: Übertragen schlug fehl (Details folgen):
svn: Datei »/rules.txt« ist veraltet
…
```

(Der genaue Wortlaut dieser Fehlermeldung hängt vom verwendeten Netzwerkprotokoll und vom Server ab, doch die Bedeutung ist in allen Fällen gleich.)

Zu diesem Zeitpunkt müssen Sie **`svn update`** aufrufen, sich um eventuelle Zusammenführungen oder Konflikte kümmern und die Übergabe erneut versuchen.

Das deckt den grundlegenden Arbeitszyklus für die Verwendung von Subversion ab. Subversion bietet viele andere Möglichkeiten, die Sie benutzen können, um Ihr Projektarchiv und Ihre Arbeitskopie zu verwalten, doch der größte Teil Ihrer

täglichen Arbeit mit Subversion wird lediglich die in diesem Kapitel behandelten Befehle berühren. Wir werden jedoch noch ein paar mehr Befehle behandeln, die Sie ziemlich oft verwenden werden.

Geschichtsforschung

Ihr Subversion-Projektarchiv ist wie eine Zeitmaschine. Es legt einen Eintrag für jede jemals übergebene Änderung an und erlaubt Ihnen, diese Geschichte durch die Untersuchung sowohl ehemaliger Datei- und Verzeichnisversionen als auch der begleitenden Metadaten zu erforschen. Mit einem einzigen Subversion-Befehl können Sie das Projektarchiv genauso auschecken (oder eine bestehende Arbeitskopie wiederherstellen), wie es zu einem beliebigen Zeitpunkt oder einer Revisionsnummer in der Vergangenheit war. Allerdings möchten Sie manchmal nur in die Vergangenheit *spähen* anstatt dorthin zu *gehen*.

Es gibt mehrere Befehle, die Sie mit historischen Daten aus dem Projektarchiv versorgen können:

svn diff
: Zeigt die Details einer bestimmten Änderung auf Zeilenebene

svn log
: Zeigt Ihnen grobe Informationen: Mit Revisionen verknüpfte Protokolleinträge zu Datum und Autor und welche Pfade sich in jeder Revision geändert haben.

svn cat
: Holt eine Datei hervor wie sie mit einer bestimmten Revisionsnummer einmal ausgesehen hat und zeigt sie auf dem Bildschirm an

svn list
: Zeigt die Dateien in einem Verzeichnis für eine gewünschte Revision an

Detaillierte Untersuchung der Änderungsgeschichte

svn diff ist uns bereits begegnet – es zeigt Dateiunterschiede im unified-diff-Format; wir verwendeten es, um die lokalen Änderungen an unserer Arbeitskopie anzuzeigen, bevor wir sie dem Projektarchiv übergaben.

Tatsächlich stellt sich heraus, dass es *drei* verschiedene Verwendungsmöglichkeiten für **svn diff** gibt:

- zum Untersuchen lokaler Änderungen
- zum Vergleichen Ihrer Arbeitskopie mit dem Projektarchiv
- zum Vergleichen von Projektarchiv-Revisionen

Untersuchen lokaler Änderungen

Wie wir gesehen haben, vergleicht der Aufruf von **`svn diff`** ohne Optionen die Arbeitsdateien mit den zwischengespeicherten „ursprünglichen“ Kopien im `.svn`-Bereich:

```
$ svn diff
Index: rules.txt
===================================================================
--- rules.txt (revision 3)
+++ rules.txt (working copy)
@@ -1,4 +1,5 @@
 Be kind to others
 Freedom = Responsibility
 Everything in moderation
-Chew with your mouth open
+Chew with your mouth closed
```

```
+Listen when others are speaking
$
```

Vergleichen der Arbeitskopie mit dem Projektarchiv

Wird eine einzelne Nummer mit `--revision` (`-r`) übergeben, wird die Arbeitskopie mit der angegebenen Revision im Projektarchiv verglichen:

```
$ svn diff -r 3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 3)
+++ rules.txt (working copy)
@@ -1,4 +1,5 @@
 Be kind to others
 Freedom = Responsibility
 Everything in moderation
-Chew with your mouth open
+Chew with your mouth closed
+Listen when others are speaking
$
```

Vergleichen von Projektarchiv-Revisionen

Werden zwei Revisionsnummern durch einen Doppelpunkt getrennt mit `--revision` (`-r`) übergeben, werden die beiden Revisionen direkt miteinander verglichen:

```
$ svn diff -r 2:3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 2)
+++ rules.txt (revision 3)
@@ -1,4 +1,4 @@
 Be kind to others
-Freedom = Chocolate Ice Cream
+Freedom = Responsibility
 Everything in moderation
 Chew with your mouth open
$
```

Eine bequemere Möglichkeit, eine Revision mit der Vorgänger-Revision zu vergleichen, bietet die Verwendung der Option `--change` (`-c`):

```
$ svn diff -c 3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 2)
+++ rules.txt (revision 3)
@@ -1,4 +1,4 @@
 Be kind to others
-Freedom = Chocolate Ice Cream
+Freedom = Responsibility
 Everything in moderation
 Chew with your mouth open
```

```
$
```

Zu guter Letzt können Sie Revisionen im Projektarchiv auch dann vergleichen, falls Sie gar keine Arbeitskopie auf Ihrem lokalen Rechner haben, indem Sie einfach den entsprechenden URL auf der Kommandozeile angeben:

```
$ svn diff -c 5 http://svn.example.com/repos/example/trunk/text/rules.txt
…
$
```

Erzeugung einer Liste der Änderungsgeschichte

Um Informationen über den Werdegang einer Datei oder eines Verzeichnisses zu bekommen, benutzen Sie den Befehl **svn log**. **svn log** versorgt Sie mit einem Eintrag, der Auskunft darüber gibt, wer Änderungen an einer Datei oder einem Verzeichnis gemacht hat, in welcher Revision die Änderung stattfand, zu welcher Zeit und welchem Datum die Revision entstand sowie – falls verfügbar – dem die Übergabe begleitenden Protokolleintrag:

```
$ svn log
------------------------------------------------------------------------
r3 | sally | 2008-05-15 23:09:28 -0500 (Thu, 15 May 2008) | 1 line

include-Zeilen hinzugefügt und Anzahl der Käsescheiben korrigiert.
------------------------------------------------------------------------
r2 | harry | 2008-05-14 18:43:15 -0500 (Wed, 14 May 2008) | 1 line

main()-Methoden hinzugefügt.
------------------------------------------------------------------------
r1 | sally | 2008-05-10 19:50:31 -0500 (Sat, 10 May 2008) | 1 line

Erstimport
------------------------------------------------------------------------
```

Beachten Sie, dass die Protokolleinträge standardmäßig in *umgekehrter zeitlicher Reihenfolge* ausgegeben werden. Falls Sie eine andere Folge von Revisionen in einer bestimmten Anordnung oder nur eine einzige Revision sehen möchten, übergeben Sie die Option `--revision` (`-r`):

Tabelle 2.1. Häufige Protokollanfragen

Befehl	Beschreibung
`svn log -r 5:19`	Anzeige der Protokolleinträge für die Revisionen 5 bis 19 in chronologischer Reihenfolge
`svn log -r 19:5`	Anzeige der Protokolleinträge für die Revisionen 5 bis 19 in umgekehrt chronologischer Reihenfolge
`svn log -r 8`	Anzeige des Protokolleintrags nur für Revision 8

Sie können sich auch die Protokollgeschichte einer einzigen Datei oder eines einzigen Verzeichnisses ansehen. Zum Beispiel:

```
$ svn log foo.c
…
$ svn log http://foo.com/svn/trunk/code/foo.c
```

...

Diese Befehle zeigen *nur* Protokolleinträge für die Revisionen, in der sich die Arbeitsdatei (oder URL) geändert hat.

Warum zeigt mir svn log nicht, was ich gerade übergeben habe?

Wenn Sie Ihre Änderungen an das Projektarchiv übergeben und sofort **`svn log`** ohne Argumente eingeben, wird Ihnen vielleicht auffallen, dass Ihre letzte Änderung nicht in der Liste der Protokolleinträge auftaucht. Das liegt an der Kombination des Verhaltens von **svn commit** und dem Standardverhalten von **svn log**. Wenn Sie Änderungen an das Projektarchiv übergeben, erhöht **svn** zunächst nur die Revision der Dateien (und Verzeichnisse) die es übernimmt, so dass das Elternverzeichnis normalerweise auf der älteren Revision verbleibt (siehe „Aktualisierungen und Übertragungen sind getrennt“ für die Erklärung, warum das so ist). **svn log** holt dann standardmäßig die Geschichte des Verzeichnisses in der gegenwärtigen Revision, und so kommt es, dass Sie die neu übergebenen Änderungen nicht sehen. Die Lösung besteht entweder in einer Aktualisierung Ihrer Arbeitskopie oder indem Sie dem Befehl **svn log** ausdrücklich mit der Option `--revision` (`-r`) eine Revisionsnummer mitgeben.

Wenn Sie noch mehr Informationen über eine Datei oder ein Verzeichnis benötigen, können Sie **svn log** auch die Option `--verbose` (`-v`) mitgeben. Weil Ihnen Subversion erlaubt, Dateien und Verzeichnisse zu kopieren und zu verschieben, ist es wichtig, Pfadänderungen im Dateisystem mitzuverfolgen. Daher beinhaltet bei dieser Option die Ausgabe von **svn log** eine Liste veränderter Pfade in einer Revision:

```
$ svn log -r 8 -v
------------------------------------------------------------------------
r8 | sally | 2008-05-21 13:19:25 -0500 (Wed, 21 May 2008) | 1 line
Geänderte Pfade:
   M /trunk/code/foo.c
   M /trunk/code/bar.h
   A /trunk/code/doc/README

Die Unterraumwinde gefrozzelt.

------------------------------------------------------------------------
```

svn log akzeptiert ebenfalls die Option `--quiet` (`-q`), die den Protokolleintrag unterdrückt. Zusammen mit der Option `--verbose` zeigt es nur die Namen der geänderten Dateien an.

Warum gibt mir svn log eine leere Antwort?

Nach ein wenig Arbeit mit Subversion werden die meisten Benutzer so etwas begegnen:

```
$ svn log -r 2
------------------------------------------------------------------------
$
```

Auf den ersten Blick sieht es aus wie ein Fehler. Aber seien Sie daran erinnert, dass, während Revisionen über das gesamte Projektarchiv zählen, **svn log** auf einem Pfad im Projektarchiv arbeitet. Wenn Sie keinen Pfad angeben, verwendet Subversion das aktuelle Arbeitsverzeichnis als Standardargument. Deshalb zeigt Subversion Ihnen einen leeren Protokolleintrag, falls Sie in einem Unterverzeichnis Ihrer Arbeitskopie arbeiten und versuchen, sich den Protokolleintrag einer Revision anzusehen, in dem sich weder dieses Verzeichnis noch irgendein Unterverzeichnis darin geändert hat. Falls Sie sehen wollen, was sich in der Revision geändert hat, versuchen Sie **svn log** direkt auf den obersten URL Ihres Projektarchivs zeigen zu lassen, wie in **`svn log -r 2 ^/`**.

Stöbern im Projektarchiv

Wenn Sie **svn cat** und **svn list** verwenden, können Sie sich verschiedene Revisionen von Dateien und Verzeichnissen ansehen, ohne die Revision Ihrer Arbeitskopie ändern zu müssen. Tatsächlich brauchen Sie dafür nicht einmal eine Arbeitskopie.

svn cat

Falls Sie eine frühere Version einer Datei untersuchen möchten und nicht notwendigerweise die Unterschiede zwischen zwei Dateien, können Sie **svn cat** verwenden:

```
$ svn cat -r 2 rules.txt
Be kind to others
Freedom = Chocolate Ice Cream
Everything in moderation
Chew with your mouth open
$
```

Sie können die Ausgabe auch direkt in eine Datei umleiten:

```
$ svn cat -r 2 rules.txt > rules.txt.v2
$
```

svn list

Der Befehl **svn list** zeigt Ihnen, welche Dateien sich in einem Projektarchiv-Verzeichnis befinden, ohne die Dateien auf Ihren lokalen Rechner herunterladen zu müssen:

```
$ svn list http://svn.example.com/repo/project
README
branches/
tags/
trunk/
```

Falls Sie eine detailliertere Auflistung wünschen, übergeben Sie die Option `--verbose` (`-v`), um eine Ausgabe ähnlich der folgenden zu bekommen:

```
$ svn list -v http://svn.example.com/repo/project
  23351 sally                Feb 05 13:26 ./
  20620 harry             1084 Jul 13  2006 README
  23339 harry                Feb 04 01:40 branches/
  23198 harry                Jan 23 17:17 tags/
  23351 sally                Feb 05 13:26 trunk/
```

Die Spalten zeigen Ihnen die Revision, in der die Datei zuletzt geändert wurde, den Benutzer, der sie änderte, die Größe, falls es sich um eine Datei handelt, sowie den Namen des Objektes.

Der Befehl **`svn list`** ohne Argumente verwendet standardmäßig den *Projektarchiv-URL* des aktuellen Arbeitsverzeichnisses und *nicht* das Verzeichnis der lokalen Arbeitskopie. Schließlich können Sie, falls Sie eine Auflistung des lokalen Verzeichnisses möchten, das einfache **ls** (oder irgendein vernünftiges nicht-unixartiges

Äquivalent) benutzen.

Bereitstellung älterer Projektarchiv-Schnappschüsse

Zusätzlich zu den obigen Befehlen können Sie die Option `--revision` (`-r`) mit **svn update** verwenden, um eine vollständige Arbeitskopie „zeitlich zurückzusetzen“:[3]

```
# Lass das aktuelle Verzeichnis aussehen wie in r1729.
$ svn update -r 1729
…
$
```

Viele Subversion-Neulinge versuchen das vorangehende **svn update**-Beispiel zu verwenden, um übergebene Änderungen „rückgängig“ zu machen, was allerdings nicht funktioniert, da Sie keine Änderungen übergeben können, die Sie durch das zeitliche Zurücksetzen einer Arbeitskopie erhalten haben, falls die geänderten Dateien neuere Revisionen haben. Siehe „Zurückholen gelöschter Objekte“ für eine Beschreibung, wie eine Übergabe „rückgängig“ gemacht wird.

Sollten Sie es bevorzugen, eine vollständige neue Arbeitskopie aus einem älteren Schnappschuss zu erzeugen, können Sie das, indem Sie den üblichen Befehl **svn checkout** modifizieren. Wie bei **svn update**, können Sie die Option `--revision` (`-r`) mitgeben. Aus Gründen, die wir in „Peg- und operative Revisionen“ erörtern werden, sollten Sie stattdessen die Zielrevision als Teil der erweiterten URL-Syntax von Subversion angeben.

```
# Den Trunk von r1729 auschecken.
$ svn checkout http://svn.example.com/svn/repo/trunk@1729 trunk-1729
…
# Den aktuellen Trunk auschecken, wie er in r1729 aussah.
$ svn checkout http://svn.example.com/svn/repo/trunk -r 1729 trunk-1729
…
$
```

Wenn Sie am Ende ein Release bauen und die Dateien aus Subversion zu einem Bündel schnüren möchten, ohne allerdings diese verdammten `.svn`-Verzeichnisse dabei zu haben, können Sie **svn export** verwenden, um eine lokale Kopie des gesamten oder teilweisen Projektarchivs ohne `.svn`-Verzeichnisse zu erhalten. Die Syntax ist grundsätzlich identisch zu der von **svn checkout**:

```
# Den Trunk aus der letzten Revision exportieren.
$ svn export http://svn.example.com/svn/repo/trunk trunk-export
…
# Den Trunk aus r1729 exportieren.
$ svn export http://svn.example.com/svn/repo/trunk@1729 trunk-1729
…
# Den aktuellen Trunk exportieren, wie er in in r1729 aussah.
$ svn export http://svn.example.com/svn/repo/trunk -r 1729 trunk-1729
…
$
```

[3] Sehen Sie? Wir haben Ihnen gesagt, dass Subversion eine Zeitmaschine sei.

Manchmal müssen Sie einfach nur aufräumen

Nachdem wir nun die täglichen Aufgaben abgehandelt haben, für die Sie regelmäßig Subversion verwenden, gehen wir nun ein paar Verwaltungsaufgaben für Ihre Arbeitskopie durch.

Entsorgen einer Arbeitskopie

Subversion merkt sich weder den Zustand noch das Vorhandensein einer Arbeitskopie auf dem Server, so dass serverseitig kein Aufwand für Arbeitskopien anfällt. Dementsprechend besteht keine Notwendigkeit, dem Server mitzuteilen, dass Sie vorhaben, eine Arbeitskopie zu löschen.

Falls die Wahrscheinlichkeit besteht, dass Sie eine Arbeitskopie wiederverwenden möchten, ist es nicht verkehrt, sie einfach auf der Platte zu lassen, bis Sie sie wieder benutzen wollen. Zu diesem Zeitpunkt reicht lediglich ein **svn update** zum Aktualisieren, und sie ist gebrauchsfertig.

Falls Sie die Arbeitskopie jedoch bestimmt nicht mehr verwenden möchten, können Sie sie ruhig löschen, indem Sie die zum Löschen von Verzeichnissen vorgesehenen Bordmittel Ihres Betriebssystems verwenden. Wir empfehlen jedoch, vorher **`svn status`** aufzurufen und alle Dateien zu untersuchen, denen ein `?` voransteht, um sicherzugehen, dass sie nicht wichtig sind.

Wiederherstellung nach einer Unterbrechung

Wenn Subversion Ihre Arbeitskopie verändert, entweder Ihre Dateien oder seinen eigenen Verwaltungsbereich, versucht es so sicher vorzugehen wie möglich. Bevor die Arbeitskopie geändert wird, schreibt Subversion seine Absichten in eine Art private „Merkliste“. Dann führt es diese Aktionen aus, um die Änderungen durchzuführen, wobei es während der Arbeit den relevanten Teil der Arbeitskopie sperrt, um andere Clients davon abzuhalten, während der Änderung auf die Arbeitskopie zuzugreifen. Zuletzt entfernt Subversion die Sperre und räumt die private Merkliste auf. Architektonisch ähnelt das einem Dateisystem mit Journal. Falls eine Operation von Subversion unterbrochen wird (z.B. wenn der Prozess abgeschossen wird oder der Rechner abstürzt), verbleibt die private Merkliste auf der Platte. Das erlaubt es Subversion später zu der Merkliste zurückzukehren, um die vorher begonnene Operationen zu vervollständigen und Ihre Arbeitskopie wieder in einen konsistenten Zustand zu bringen.

Genau das macht **svn cleanup**: Es durchsucht Ihre Arbeitskopie und arbeitet etwaige übrig gebliebene Punkte von der Merkliste ab, wobei Sperren in der Arbeitskopie entfernt werden. Falls Ihnen Subversion jemals mitteilt, dass ein Teil Ihrer Arbeitskopie „gesperrt“ ist, sollten Sie **svn cleanup** aufrufen, um das Problem zu beheben. Der Befehl **svn status** informiert Sie auch über administrative Sperren in der Arbeitskopie, indem es ein `L` neben diesen gesperrten Pfaden anzeigt:

```
$ svn status
  L     somedir
M       somedir/foo.c
$ svn cleanup
$ svn status
M       somedir/foo.c
```

Umgang mit Strukturkonflikten

Bis hierhin haben wir nur über Konflikte auf der Ebene von Dateiinhalten gesprochen. Wenn Sie und Ihre Mitarbeiter überlappende Änderungen innerhalb derselben Datei vornehmen, zwingt Sie Subversion dazu, diese Änderungen zusammenzuführen, bevor Sie sie übergeben können.[4]

Was passiert aber, wenn Ihre Mitarbeiter eine Datei verschieben oder löschen, an der Sie noch arbeiten? Vielleicht gab es ein Verständnisproblem oder die eine Person glaubt, die Datei soll gelöscht werden, während die andere Person noch Änderungen an der Datei übergeben will. Vielleicht haben Ihre Mitarbeiter ja auch etwas Refactoring betrieben und dabei Dateien umbenannt und Verzeichnisse verschoben. Falls Sie noch an diesen Dateien gearbeitet haben, müssten diese Änderungen auf die Dateien am der neuen Stelle angewendet werden. Derartige Konflikte äußern sich auf der Ebene der Verzeichnisstruktur statt des Dateiinhaltes und sind bekannt als *Baumkonflikte*.

[4]Natürlich *könnten* Sie Dateien, die Konfliktmarkierungen enthalten, als konfliktfrei erklären und übergeben, wenn Sie es wirklich wollten, doch das wird in der Praxis kaum gemacht.

Baumkonflikte vor Subversion 1.6

Vor Subversion 1.6 konnten Baumkonflikte zu ziemlich unerwarteten Ergebnissen führen. Wenn beispielsweise eine Datei lokal geändert im Projektarchiv jedoch umbenannt worden war, hätte ein **svn update** Subversion zu den folgenden Schritten veranlasst:

- Überprüfe, ob die umzubenennende Datei lokale Änderungen hat.
- Lösche die Datei an ihrer alten Stelle und bewahre eine Kopie der Datei an der alten Stelle auf, falls lokale Änderungen vorhanden waren. Diese Kopie erscheint nun als eine nicht versionierte Datei in der Arbeitskopie.
- Füge die Datei wie sie im Projektarchiv besteht an ihrer neuen Stelle hinzu.

Beim Eintreten dieser Situation besteht die Möglichkeit, dass der Anwender eine Übergabe durchführt, ohne sich bewusst zu sein, dass sich lokale Änderungen immer noch in einer nun nicht-versionierten Datei der Arbeitskopie befinden und noch nicht im Projektarchiv sind. Dies wird wahrscheinlicher (und lästiger), falls die Anzahl der von diesem Problem berührten Dateien groß ist.

Seit Subversion 1.6 werden diese und ähnliche Situationen als Konflikte in der Arbeitskopie gekennzeichnet.

Wie bei textuellen Konflikten verhindern Baumkonflikte eine Übergabe aus dem Konfliktzustand und geben dem Anwender die Gelegenheit, den Zustand der Arbeitskopie auf potenzielle Probleme, die aus dem Baumkonflikt entstehen könnten, zu überprüfen und vor der Übergabe aufzulösen.

Ein Beispiel für einen Baumkonflikt

Gegeben sei folgendes Softwareprojekt, an dem Sie gerade arbeiten:

```
$ svn list -Rv svn://svn.example.com/trunk/
      4 harry                Feb 06 14:34 ./
      4 harry              23 Feb 06 14:34 COPYING
      4 harry              41 Feb 06 14:34 Makefile
      4 harry              33 Feb 06 14:34 README
      4 harry                 Feb 06 14:34 code/
      4 harry              51 Feb 06 14:34 code/bar.c
      4 harry             124 Feb 06 14:34 code/foo.c
```

Ihr Kollege Harry hat die Datei `bar.c` in `baz.c` umbenannt. Sie arbeiten immer noch an `bar.c` in Ihrer Arbeitskopie, doch wissen noch nicht, dass die Datei im Projektarchiv umbenannt wurde.

Die Protokollmeldung für Harrys Übergabe sah so aus:

```
$ svn log -r5 svn://svn.example.com/trunk
------------------------------------------------------------------------
r5 | harry | 2009-02-06 14:42:59 +0000 (Fri, 06 Feb 2009) | 2 lines
Geänderte Pfade:
   M /trunk/Makefile
   D /trunk/code/bar.c
   A /trunk/code/baz.c (from /trunk/code/bar.c:4)

bar.c nach baz.c umbenannt und Makefile entsprechend angepasst.
```

Die von Ihnen vorgenommenen Änderungen sehen so aus:

```
$ svn diff
Index: code/foo.c
===================================================================
--- code/foo.c  (revision 4)
+++ code/foo.c  (working copy)
@@ -3,5 +3,5 @@
 int main(int argc, char *argv[])
 {
        printf("I don't like being moved around!\n%s", bar());
-       return 0;
+       return 1;
 }
Index: code/bar.c
===================================================================
--- code/bar.c  (revision 4)
+++ code/bar.c  (working copy)
@@ -1,4 +1,4 @@
 const char *bar(void)
 {
-       return "Me neither!\n";
+       return "Well, I do like being moved around!\n";
 }
```

Ihre Änderungen bauen alle auf Revision 4 auf. Sie können nicht übergeben werden, da Harry bereits Revision 5 übergeben hat:

```
$ svn commit -m "Small fixes"
Sende          code/bar.c
Sende          code/foo.c
Übertrage Daten ..
svn: Übertragen schlug fehl (Details folgen):
svn: Datei nicht gefunden: Transaktion »5-5«, Pfad »/trunk/code/bar.c«
```

An dieser Stelle müssen Sie **svn update** aufrufen. Außer Ihre Arbeitskopie zu aktualisieren, so dass Sie Harrys Änderungen sehen können, markiert es auch einen Baumkonflikt, so dass Sie die Gelegenheit bekommen, die Situation abzuschätzen und entsprechend aufzulösen.

```
$ svn update
   C code/bar.c
A    code/baz.c
U    Makefile
Aktualisiert zu Revision 5.
Konfliktübersicht:
  Baumkonflikte: 1
```

In seiner Ausgabe zeigt **svn update** Baumkonflikte mit einem großen C in der vierten Spalte an. **svn status** enthüllt weitere Details zum Konflikt:

```
$ svn status
M       code/foo.c
A  +  C code/bar.c
      >   lokal editiert, eingehend gelöscht bei Aktualisierung
M       code/baz.c
```

Beachten Sie, wie bar.c automatisch in Ihrer Arbeitskopie zum erneuten Hinzufügen vorgemerkt wird, was die Sache vereinfacht, sollten sie sich entscheiden, die Datei zu behalten.

Da eine Verschiebung in Subversion als eine Kopie mit anschließender Löschung implementiert ist, und diese beiden Operationen sich bei einer Aktualisierung sich nicht einfach in Beziehung setzen lassen, kann Sie Subversion lediglich über eine hereinkommende Löschung einer lokal modifizierten Datei warnen. Diese Löschung *kann* Teil einer Verschiebung sein oder eine tatsächliche Löschung. Um herauszufinden, was wirklich passiert ist, empfiehlt es sich, mit Ihren Kollegen Rücksprache zu halten, oder, als letzte Möglichkeit, **svn log** zu befragen.

Sowohl `foo.c` als auch `baz.c` werden in der Ausgabe von **svn status** als lokal modifiziert angezeigt. Sie selbst haben Änderungen an `foo.c` vorgenommen, das sollte Sie nicht überraschen. Aber warum wird `baz.c` als lokal modifiziert angezeigt?

Die Antwort ist, dass, trotz der Einschränkungen bei der Implementierung der Verschiebung, Subversion klug genug war, Ihre lokalen Änderungen an `bar.c` nach `baz.c` zu übertragen:

```
$ svn diff code/baz.c
Index: code/baz.c
===================================================================
--- code/baz.c  (revision 5)
+++ code/baz.c  (working copy)
@@ -1,4 +1,4 @@
 const char *bar(void)
 {
-       return "Me neither!\n";
+       return "Well, I do like being moved around!\n";
 }
```

Lokale Änderungen an der Datei `bar.c`, die während einer Aktualisierung in `baz.c` umbenannt wird, werden nur dann auf `bar.c` angewendet, falls Ihre Arbeitskopie von `bar.c` auf der Revision aufbaut, in der sie zuletzt modifiziert wurde, bevor sie im Projektarchiv verschoben wurde. Anderenfalls wird Subversion sich damit begnügen, `baz.c` aus dem Projektarchiv zu holen, und nicht versuchen, Ihre lokalen Änderungen dorthin zu übertragen. Das müssen Sie manuell machen.

svn info zeigt die URLs der am Konflikt beteiligten Objekte. Der *linke* URL zeigt die Quelle der lokalen Seite des Konfliktes, während die *rechte* URL die Quelle der hereinkommenden Seite des Konfliktes anzeigt. Diese URLs weisen darauf hin, wo Sie mit der Suche nach der mit Ihrer lokalen Änderung in Konflikt stehenden Änderung in der Vorgeschichte des Projektarchivs beginnen sollten.

```
$ svn info code/bar.c | tail -n 4
Baumkonflikt: lokal editiert, eingehend gelöscht bei Aktualisierung
  Quelle  links: (Datei) ^/trunk/code/bar.c@4
  Quelle  rechts: (nichts) ^/trunk/code/bar.c@5
```

`bar.c` heißt nun *Opfer* eines Baumkonfliktes. Sie kann nicht übergeben werden, bevor der Konflikt aufgelöst wird:

```
$ svn commit -m "Small fixes"
svn: Übertragen schlug fehl (Details folgen):
svn: Übertragung abgebrochen: »code/bar.c« bleibt im Konflikt
```

Wie kann denn dieser Konflikt nun aufgelöst werden? Sie können entweder mit Harrys Vorgehen einverstanden sein oder nicht. Falls ja, löschen Sie `bar.c` und markieren den Baumkonflikt als aufgelöst:

```
$ svn delete --force code/bar.c
D         code/bar.c
$ svn resolve --accept=working code/bar.c
Konflikt von »code/bar.c« aufgelöst
$ svn status
M       code/foo.c
M       code/baz.c
$ svn diff
Index: code/foo.c
===================================================================
--- code/foo.c  (revision 5)
+++ code/foo.c  (working copy)
@@ -3,5 +3,5 @@
 int main(int argc, char *argv[])
 {
         printf("I don't like being moved around!\n%s", bar());
-        return 0;
+        return 1;
 }
Index: code/baz.c
===================================================================
--- code/baz.c  (revision 5)
+++ code/baz.c  (working copy)
@@ -1,4 +1,4 @@
 const char *bar(void)
 {
-        return "Me neither!\n";
+        return "Well, I do like being moved around!\n";
 }
```

Falls Sie mit dem Vorgehen nicht einverstanden sind, können Sie stattdessen `baz.c` löschen, nachdem Sie sichergestellt haben, das alle nach der Umbenennung vorgenommenen Änderungen entweder bewahrt worden sind, oder verworfen werden können. Vergessen Sie nicht, die Änderungen zurückzunehmen, die Harry an `Makefile` gemacht hat. Da `bar.c` bereits zum neu Hinzufügen vorgemerkt ist, bleibt nichts mehr zu tun, und der Konflikt kann als aufgelöst markiert werden:

```
$ svn delete --force code/baz.c
D         code/baz.c
$ svn resolve --accept=working code/bar.c
Konflikt von »code/bar.c« aufgelöst
$ svn status
M       code/foo.c
A  +    code/bar.c
D       code/baz.c
M       Makefile
$ svn diff
Index: code/foo.c
===================================================================
--- code/foo.c (revision 5)
+++ code/foo.c (working copy)
@@ -3,5 +3,5 @@
 int main(int argc, char *argv[])
 {
          printf("I don't like being moved around!\n%s", bar());
- return 0;
+ return 1;
 }
Index: code/bar.c
===================================================================
--- code/bar.c (revision 5)
+++ code/bar.c (working copy)
@@ -1,4 +1,4 @@
 const char *bar(void)
 {
```

```
- return "Me neither!\n";
+ return "Well, I do like being moved around!\n";
 }
Index: code/baz.c
===================================================================
--- code/baz.c (revision 5)
+++ code/baz.c (working copy)
@@ -1,4 +0,0 @@
-const char *bar(void)
-{
- return "Me neither!\n";
-}
Index: Makefile
===================================================================
--- Makefile (revision 5)
+++ Makefile (working copy)
@@ -1,2 +1,2 @@
 foo:
- $(CC) -o $@ code/foo.c code/baz.c
+ $(CC) -o $@ code/foo.c code/bar.c
```

In jedem Fall haben Sie nun Ihren ersten Baumkonflikt aufgelöst! Sie können Ihre Änderungen übergeben und Harry in der Kaffeepause erzählen, welche Mehrarbeit er Ihnen bereitet hat.

Zusammenfassung

Nun haben wir die meisten der Subversion-Client-Befehle behandelt. Erwähnenswerte Ausnahmen sind diejenigen, die sich mit dem Branchen und Zusammenführen befassen (siehe Kapitel 4, *Verzweigen und Zusammenführen*) sowie mit Eigenschaften (siehe „Eigenschaften“). Jedoch möchten Sie vielleicht einen Augenblick damit verbringen, um durch Kapitel 9, *Die vollständige Subversion Referenz* zu blättern, um ein Gefühl für all die verschiedenen Befehle zu bekommen, über die Subversion verfügt – und wie Sie sie verwenden können, um Ihre Arbeit zu erleichtern.

Kapitel 3. Fortgeschrittene Themen

Falls Sie dieses Buch kapitelweise von vorne nach hinten lesen, sollten Sie sich bis hierhin ausreichende Kenntnisse über die Benutzung des Subversion-Clients angeeignet haben, um die gebräuchlichsten Versionskontrolltätigkeiten ausführen zu können. Sie wissen, wie eine Arbeitskopie aus einem Subversion-Projektarchiv ausgecheckt wird. Sie kommen gut damit zurecht, Änderungen mittels **svn commit** und **svn update** zu übergeben bzw. zu empfangen. Sie haben sich wahrscheinlich angewöhnt, den Befehl **svn status** quasi unbewusst aufzurufen. Sie können für alle möglichen Vorhaben und Zwecke in einer typischen Umgebung Subversion verwenden.

Subversions Funktionsumfang macht allerdings nicht bei „gewöhnlichen Versionskontrolltätigkeiten“ halt. Es bietet mehr als lediglich den Transport von Datei- und Verzeichnisänderungen in ein und aus einem zentralen Projektarchiv zu ermöglichen.

Dieses Kapitel beleuchtet einige der Funktionen von Subversion, die, obwohl wichtig, jedoch nicht Bestandteil des typischen Tagesablaufs eines Benutzers sein könnten. Es wird vorausgesetzt, dass Sie sich mit den grundlegenden Fähigkeiten von Subversion zur Datei- und Verzeichnisversionierung auskennen. Falls nicht, sollten Sie zunächst Kapitel 1, *Grundlegende Konzepte* und Kapitel 2, *Grundlegende Benutzung* lesen. Sobald Sie diese Grundlagen gemeistert und dieses Kapitel durchgearbeitet haben werden, werden Sie ein Subversion-Power-User sein.

Revisionsbezeichner

Wie bereits in „Revisionen“ beschrieben, sind Revisionsnummern in Subversion ziemlich unkompliziert – ganze Zahlen, die bei jeder Übergabe einer Änderung größer werden. Trotzdem wird es nicht lange dauern bis Sie sich nicht mehr genau erinnern können, was in welcher Revision geschah. Glücklicherweise erfordert der typische Arbeitsablauf in Subversion selten die Angabe von beliebigen Revisionen für die von Ihnen ausgeführten Funktionen. Für Funktionen, die *dennoch* einen Revisionsbezeichner erfordern, geben Sie im Allgemeinen eine Revisionsnummer an, die Sie in einer Übergabe-E-Mail, in der Ausgabe einer anderen Subversion-Funktion oder in einem anderen bedeutsamen Zusammenhang gesehen haben.

Gelegentlich müssen Sie jedoch einen Zeitpunkt genau festlegen, für den Sie sich an keine Revisionsnummer erinnern können oder für den Sie keine parat haben. Deshalb erlaubt Ihnen **svn** neben ganzzahligen Revisionsnummern weitere Formen von Revisionsbezeichnern: *Revisions-Schlüsselworte* und Revisionsdaten.

Die verschiedenen Formen von Subversion-Revisionsbezeichnern können bei der Angabe von Revisionsbereichen gemischt werden. Beispielsweise können Sie `-r` *`REV1`*`:`*`REV2`* verwenden, wobei *`REV1`* ein Revisions-Schlüsselwort und *`REV2`* eine Revisionsnummer ist oder *`REV1`* ein Datum und *`REV2`* ein Revisions-Schlüsselwort, usw. Die einzelnen Revisionsbezeichner werden voneinander unabhängig ausgewertet, so dass Sie links und rechts des Doppelpunktes angeben können, was Sie möchten.

Revisions-Schlüsselworte

Der Subversion-Client versteht eine Reihe von Revisions-Schlüsselworten. Diese Schlüsselworte können der Option `--revision` (`-r`) anstatt ganzer Zahlen als Optionsargument übergeben werden; sie werden von Subversion zu bestimmten Revisionsnummern aufgelöst:

`HEAD`

Die letzte (oder „jüngste“) Revision im Projektarchiv.

`BASE`

Die Revisionsnummer eines Objektes in der Arbeitskopie. Falls das Objekt lokal bearbeitet wurde, bezieht sie sich auf das unmodifizierte Objekt.

`COMMITTED`

Die letzte Revision kleiner oder gleich `BASE`, in der ein Objekt verändert wurde.

`PREV`

Die Revision unmittelbar *vor* der letzten Revision, in der ein Objekt verändert wurde. Technisch bedeutet das `COMMITTED`-1.

Wie aus den Beschreibungen abgeleitet werden kann, werden die Revisions-Schlüsselworte `PREV`, `BASE` und `COMMITTED` nur in Bezug auf einen Pfad der Arbeitskopie verwendet – sie sind nicht auf URLs des Projektarchivs anwendbar. `HEAD` kann hingegen in Verbindung mit beiden Pfadtypen verwendet werden.

Hier ein paar Beispiele zur Verwendung von Revisions-Schlüsselworten:

```
$ svn diff -r PREV:COMMITTED foo.c
# zeigt die letzte übergebene Änderung von foo.c an

$ svn log -r HEAD
# gibt die Protokollnachricht der letzten Übergabe an das
# Projektarchiv aus

$ svn diff -r HEAD
# vergleicht Ihre Arbeitskopie (mit allen lokalen Änderungen) mit der
# letzten Version dieses Baums im Projektarchiv

$ svn diff -r BASE:HEAD foo.c
# vergleicht die unmodifizierte Version von foo.c mit der letzten
# Version von foo.c im Projektarchiv

$ svn log -r BASE:HEAD
# gibt alle Übergabe-Protokollnachrichten des aktuellen versionierten
# Verzeichnisses seit der letzten Aktualisierung aus

$ svn update -r PREV foo.c
# macht die letzte Änderung an foo.c rückgängig, indem die
# Arbeitsrevision von foo.c vermindert wird

$ svn diff -r BASE:14 foo.c
# vergleicht die unmodifizierte Version von foo.c mit foo.c in der
# Revision 14
```

Revisionsdaten

Revisionsnummern offenbaren nichts über die Welt außerhalb des Versionskontrollsystems, doch manchmal müssen Sie einen Zeitpunkt mit einem Moment der Versionsgeschichte korrelieren. Um das zu ermöglichen, erlaubt die Option `--revision` (`-r`) auch Datumsangaben in geschweiften Klammern (`{` und `}`). Subversion akzeptiert die standardisierten ISO-8601 Datums- und Zeitformate und ein paar weitere. Hier sind einige Beispiele.

```
$ svn checkout -r {2006-02-17}
$ svn checkout -r {15:30}
$ svn checkout -r {15:30:00.200000}
$ svn checkout -r {"2006-02-17 15:30"}
$ svn checkout -r {"2006-02-17 15:30 +0230"}
$ svn checkout -r {2006-02-17T15:30}
$ svn checkout -r {2006-02-17T15:30Z}
$ svn checkout -r {2006-02-17T15:30-04:00}
$ svn checkout -r {20060217T1530}
$ svn checkout -r {20060217T1530Z}
$ svn checkout -r {20060217T1530-0500}
…
```

Beachten Sie, dass es die meisten Shells erforderlich machen, mindestens Leerzeichen in Anführungsstriche zu setzen oder anderweitig zu maskieren, wenn sie Teile von Revisionsdaten-Spezifizierungen sind. Bestimmte Shells könnten auch Probleme mit der unmaskierten Verwendung von geschweiften Klammern bekommen. Schlagen Sie in der Dokumentation Ihrer Shell nach, was in Ihrer Umgebung notwendig ist.

Falls Sie ein Datum angeben, wandelt Subversion dieses Datum in die letzte Revision zum Zeitpunkt dieses Datums um und verwendet dann die entsprechende Revisionsnummer:

```
$ svn log -r {2006-11-28}
------------------------------------------------------------------------
r12 | ira | 2006-11-27 12:31:51 -0600 (Mon, 27 Nov 2006) | 6 lines
…
```

Geht Subversion einen Tag vor?

Falls Sie ein einzelnes Datum ohne Urzeit als Revision angeben, (z.B. `2006-11-27`), könnten Sie denken, dass Subversion Ihnen die letzte Revision vom 27. November liefert. Stattdessen bekommen Sie eine Revision vom 26. oder sogar noch früher. Denken Sie daran, dass Subversion die *letzte Revision des Projektarchivs* zum angegebenen Zeitpunkt findet. Falls Sie ein Datum ohne Uhrzeit angeben, etwa `2006-11-27`, nimmt Subversion die Uhrzeit 00:00:00 an, so dass die Suche nach der letzten Revision nichts vom 27. zurückliefert.

Falls Sie den 27. für Ihre Suche berücksichtigen möchten, können Sie entweder den 27. mit Uhrzeit angeben (`{"2006-11-27 23:59"}`) oder einfach den nächsten Tag (`{2006-11-28}`).

Sie können auch einen Zeitraum angeben. Subversion findet dann alle Revisionen zwischen den und einschließlich der Daten:

```
$ svn log -r {2006-11-20}:{2006-11-29}
…
```

Da der Zeitstempel einer Revision als eine unversionierte, änderbare Eigenschaft einer Revision gespeichert ist (siehe „Eigenschaften“), können Revisions-Zeitstempel geändert werden, um die wahre Chronologie zu fälschen, oder gar vollständig entfernt werden. Die Fähigkeit von Subversion, Revisionsdaten in Revisionsnummern überführen zu können, beruht auf einer sequentiellen Ordnung der Revisions-Zeitstempel – je jünger die Revision, desto jünger der Zeitstempel. Falls diese Ordnung nicht aufrechterhalten wird, werden Sie voraussichtlich feststellen, dass der Versuch, Daten zur Angabe von Revisionsbereichen in Ihrem Projektarchiv zu verwenden, nicht immer die Daten zurückliefern wird, die Sie erwartet hätten.

Peg- und operative Revisionen

Dateien und Verzeichnisse werden auf unseren Rechnern ständig kopiert, verschoben, umbenannt und vollständig ersetzt. Ihr Versionskontrollsystem sollte nicht im Weg stehen, wenn Sie diese Dinge auch mit Dateien und Verzeichnissen unter Versionskontrolle machen. Die Dateiverwaltungsunterstützung von Subversion ist sehr befreiend, indem sie beinahe die gleiche Flexibilität bei versionierten Dateien erlaubt, die Sie bei der Handhabung unversionierter Dateien erwarten. Diese Flexibilität bedeutet aber, dass während der Lebenszeit Ihres Projektarchivs ein gegebenes versioniertes Objekt viele Pfade haben kann, und ein gegebener Pfad verschiedene vollständig unterschiedliche versionierte Objekte repräsentieren kann. Das fügt Ihrer Arbeit mit diesen Pfaden und Objekten einen gewissen Grad an Komplexität hinzu.

Subversion ist ziemlich schlau, wenn es darum geht, festzustellen, wann die Versionsgeschichte eines Objektes eine solche „Adressänderung“ beinhaltet. Wenn Sie beispielsweise das Protokoll der Versionsgeschichte einer bestimmten Datei abfragen,

die letzte Woche umbenannt wurde, wird Subversion erfreulicherweise all diese Protokolleinträge liefern – die Revision, in der die Umbenennung vorgenommen wurde und die wichtigen Einträge aus der Zeit vor und nach der Umbenennung. Meistens brauchen Sie sich also nicht um solche Dinge zu kümmern. Doch ist Subversion gelegentlich auf Ihre Hilfe angewiesen, um Unklarheiten aufzuklären.

Das einfachste Beispiel hierfür tritt auf, falls ein Verzeichnis oder eine Datei aus der Versionskontrolle gelöscht und dann ein neues Verzeichnis oder eine neue Datei gleichen Namens erzeugt und unter Versionskontrolle gestellt wird. Bei dem Ding, das Sie gelöscht haben und dem, das Sie später hinzugefügt haben handelt es sich nicht um das selbe Ding. Sie haben lediglich den gleichen Pfad gehabt, beispielsweise `/trunk/object`. Was bedeutet es dann, Subversion nach der Geschichte von `/trunk/object` zu fragen? Fragen Sie nach dem Ding, das sich momentan an diesem Ort befindet oder dem alten Ding, das Sie von dort gelöscht haben? Fragen Sie nach den Arbeiten, die an *allen* Objekten stattgefunden haben, die sich jemals unter diesem Pfad befunden haben? Subversion benötigt einen Hinweis darauf, was Sie wirklich möchten.

Durch Verschiebungen kann die Versionsgeschichte sogar weitaus komplizierter werden. Sie haben beispielsweise ein Verzeichnis namens `concept`, das ein im Werden begriffenes Software-Projekt beinhaltet, mit dem Sie herumgespielt haben. Schließlich reift das Projekt soweit heran, dass die Idee Flügel bekommen zu haben scheint, was Sie das Undenkbare machen lässt: dem Projekt einen Namen geben. [1] Nehmen wir an, Sie haben Ihre Software Frabnaggilywort genannt. Zu diesem Zeitpunkt erscheint es sinnvoll, das Verzeichnis umzubenennen, um den neuen Namen des Projektes widerzuspiegeln, also wird `concept` umbenannt in `frabnaggilywort`. Das Leben geht weiter, und von Frabnaggilywort wird eine Version 1.0 veröffentlicht, die von Massen an Menschen, die ihr Leben zu verbessern trachten, heruntergeladen und täglich benutzt werden.

Dies ist eine wirklich schöne Geschichte, die hier aber nicht endet. Als echter Unternehmer gehen Sie bereits mit der nächsten Idee schwanger. Also erstellen Sie ein neues Verzeichnis `concept`, und der Zyklus beginnt erneut. Tatsächlich startet dieser Zyklus sehr oft neu über die Jahre, jedes Mal mit dem alten Verzeichnis `concept`, das manchmal umbenannt wird, falls die Idee reift, manchmal jedoch gelöscht wird, wenn die Idee verworfen wird. Oder, um es auf die Spitze zu treiben, Sie benennen `concept` für eine Weile in etwas anderes um, aber aus irgend einem Grund taufen Sie es später wieder `concept`.

In Szenarios wie diesem, verhält sich der Versuch, Subversion aufzufordern, mit diesen wiederverwendeten Pfaden zu arbeiten, ähnlich, wie einem Autofahrer in den westlichen Vororten Chicagos zu erklären, die Roosevelt Road ostwärts zu fahren und links in die Main Street abzubiegen. In nur zwanzig Minuten können Sie die „Main Street“ in Wheaton, Glen Ellyn und Lombard kreuzen. Aber das ist keineswegs die selbe Straße. Unser Autofahrer – und Subversion – benötigen etwas mehr Details, um das Richtige machen zu können.

Glücklicherweise erlaubt es Subversion Ihnen, zu sagen, welche Main Street Sie genau meinten. Der verwendete Mechanismus wird *Peg-Revision* genannt und Sie geben ihn Subversion alleinig zu dem Zweck mit, eindeutige Linien in der Historie zu identifizieren. Da zu einer gegebenen Zeit höchstens ein versioniertes Objekt einen Pfad belegen kann – oder, genauer, in irgend einer Revision – wird zum Referenzieren einer bestimmten Linie in der Historie lediglich die Kombination aus Pfad und Peg-Revision benötigt. Peg-Revisionen werden dem Subversion-Kommandozeilen-Client in *At-Syntax* mitgegeben, die so genannt wird, da diese Syntax das Anhängen eines „At-Zeichens“ (@) und die Peg-Revision an das Ende des mit der Revision verbundenen Pfades vorsieht.

Doch was ist mit der Option `--revision` (`-r`), von der wir in diesem Buch so oft gesprochen haben? Diese Revision (oder Menge von Revisionen) wird *operative Revision* (oder *operativer Revisionsbereich*) genannt. Sobald durch den Pfad und die Peg-Revision eine bestimmte Linie in der Historie identifiziert ist, führt Subversion die verlangte Operation mit der/dem operativen Revision/Revisionsbereich aus. Auf die Analogie mit den Chicagoer Straßen angewendet, bedeutet das, wenn wir aufgefordert werden, zu 606 N. Main Street in Wheaton [2] zu gehen, können wir uns „Main Street“ als unseren Pfad vorstellen und „Wheaton“ als unsere Peg-Revision. Diese beiden Teile an Informationen identifizieren einen eindeutigen Pfad, der begangen werden kann (nördlich oder südlich auf der Main Street), und es hält uns davon ab, auf der Suche nach unserem Ziel, die falsche Main Street herauf oder herunter zu laufen. Nun fügen wir noch „606 N.“ quasi als operative Revision hinzu, und wir wissen *genau*, wo wir hin müssen.

Der Algorithmus für Peg-Revisionen

Der Kommandozeilen-Client von Subversion führt den Peg-Revisions-Algorithmus immer dann aus, wenn er mögliche Mehrdeutigkeiten in den übergebenen Pfaden und Revisionen auflösen muss. Hier ist ein Beispiel für einen solchen Aufruf:

[1] „Sie sollten es nicht mit einem Namen versehen. Sobald Sie es mit einem Namen versehen, beginnen Sie, sich mit ihm verbunden zu fühlen.“ – Mike Wazowski

[2] 606 N. Main Street, Wheaton, Illinois, ist die Heimat des Wheaton *Geschichts*-Zentrums. Es erschien angebracht….

```
$ svn command -r OPERATIVE-REV item@PEG-REV
```

Falls *OPERATIVE-REV* älter als *PEG-REV* ist, lautet der Algorithmus wie folgt:

1. Ermittle den Ort von *item* in der durch *PEG-REV* identifizierten Revision. Es kann nur ein solches Objekt geben.
2. Verfolge die Geschichte des Objektes rückwärts (durch mögliche Umbenennungen hindurch) bis seinem Vorgänger in der Revision *OPERATIVE-REV*.
3. Führe die gewünschte Aktion auf diesem Vorgänger aus, egal wo er sich befindet oder wie er heißt oder damals hieß.

Aber was ist, falls *OPERATIVE-REV jünger* ist als *PEG-REV*? Nun, das erhöht den Aufwand für das theoretische Problem der Ermittlung des Pfades in *OPERATIVE-REV* etwas komplexer, da die Geschichte des Pfades sich aufgrund von Kopiervorgängen zwischen *PEG-REV* und *OPERATIVE-REV* vervielfältigt haben könnte. Aber das ist noch nicht alles – Subversion speichert nicht genug Informationen, um die Geschichte eines Objektes auf performante Weise vorwärts zu verfolgen. Also ist der Algorithmus anders:

1. Ermittle den Ort von *item* in der durch *OPERATIVE-REV* identifizierten Revision. Es kann nur ein solches Objekt geben.
2. Verfolge die Geschichte des Objektes rückwärts (durch mögliche Umbenennungen hindurch) bis seinem Vorgänger in der Revision *PEG-REV*.
3. Stelle sicher, dass der (pfadmäßige) Ort des Objekts in *PEG-REV* der selbe ist wie in *OPERATIVE-REV*. Falls das der Fall ist, stehen mindestens zwei Orte direkt miteinander in Beziehung, so dass die gewünschte Aktion am Ort von *OPERATIVE-REV* durchgeführt werden kann. Falls nicht, konnte keine Beziehung hergestellt werden, so dass eine Fehlermeldung ausgegeben wird, das kein brauchbarer Ort ermittelt werden konnte. (Wir erwarten, dass Subversion eines Tages in der Lage sein wird, diesen Anwendungsfall flexibler und höflicher zu behandeln.)

Beachten Sie, dass Peg-Revisionen auch dann vorhanden sind, falls Sie keine Peg- oder operative Revision angeben. Der Einfachheit halber ist die standardmäßige Peg-Revision für Objekte in der Arbeitskopie `BASE` und `HEAD` für URLs im Projektarchiv. Und wird keine operative Revision angegeben, wird standardmäßig die selbe Revision wie die Peg-Revision angenommen.

Angenommen, dass unser Projektarchiv vor langer Zeit angelegt wurde, und wir in Revision 1 das erste Verzeichnis `concept` anlegten, darin eine Datei `IDEA`, die das Konzept beschreibt. Nach einigen Revisionen, in denen echter Quellcode hinzugefügt und verändert wurde, benannten wir in Revision 20 dieses Verzeichnis in `frabnaggilywort` um. Bis Revision 27 hatten wir ein neues Konzept, ein neues `concept`-Verzeichnis dafür und eine neue Datei `IDEA` zur Beschreibung. Dann zogen fünf Jahre und tausende Revisionen ins Land wie es auch in jeder guten Liebesgeschichte passiert.

Nun, Jahre später, fragen wir uns, wie die Datei `IDEA` damals in Revision 1 aussah. Subversion muss jedoch wissen, ob wir nach der *aktuellen* Datei in Revision 1 fragen oder nach dem Inhalt irgendeiner Datei, die in Revision 1 an der Stelle `concepts/IDEA` zu finden war. Sicherlich haben diese Fragen unterschiedliche Antworten, und Dank der Peg-Revisionen können Sie danach fragen. Um zu sehen, wie die aktuelle Datei `IDEA` in dieser alten Revision aussah, tippen Sie:

```
$ svn cat -r 1 concept/IDEA
svn: Kann »concept/IDEA« in Revision 1 nicht im Projektarchiv finden
```

In diesem Beispiel gab es die aktuelle Datei `IDEA` natürlich noch nicht in Revision 1, so dass Subversion einen Fehler ausgibt. Der letzte Befehl ist eine Kurzform der längeren Form, die explizit eine Peg-Revision aufführt. Die ausführliche Notation lautet:

```
$ svn cat -r 1 concept/IDEA@BASE
svn: Kann »concept/IDEA« in Revision 1 nicht im Projektarchiv finden
```

Beim Ausführen kommt es dann zum erwarteten Ergebnis.

Der scharfsinnige Leser fragt sich an dieser Stelle wahrscheinlich, ob die Syntax der Peg-Revisionen problematisch für Pfade in Arbeitskopien oder URLs sein kann, die At-Zeichen beinhalten. Woher weiß **svn** letztendlich, ob `news@11` der Name eines Verzeichnisses in meinem Baum ist oder nur die Syntax für „Revision 11 von `news`“? Während **svn** stets letzteres annimmt, gibt es glücklicherweise eine Abhilfe. Sie brauchen lediglich ein At-Zeichen am Ende des Pfades anfügen, etwa `news@11@`. **svn** schert sich nur um das letzte At-Zeichen im Argument, und es ist nicht verboten, nach dem At-Zeichen die Angabe der Peg-Revision auszulassen. Diese Abhilfe gilt sogar für Pfade, die auf ein At-Zeichen enden – Sie würden `filename@@` verwenden, um sich auf eine Datei namens `filename@` zu beziehen..

Stellen wir nun die andere Frage. Was war der Inhalt der Datei, die sich zum Zeitpunkt von Revision 1 am Ort von `concepts/IDEA` befand? Um das herauszufinden, verwenden wir eine explizite Peg-Revision.

```
$ svn cat concept/IDEA@1
The idea behind this project is to come up with a piece of software
that can frab a naggily wort.  Frabbing naggily worts is tricky
business, and doing it incorrectly can have serious ramifications, so
we need to employ over-the-top input validation and data verification
mechanisms.
```

Beachten Sie, dass wir dieses Mal keine operative Revision angegeben haben. Wenn nämlich keine operative Revision angegeben wird, nimmt Subversion standardmäßig an, dass die operative Revision die selbe wie die Peg-Revision ist.

Wie Sie sehen, scheint die Ausgabe unserer Operation korrekt zu sein. Der Text erwähnt sogar „frabbing naggily worts“, so dass es höchstwahrscheinlich die Datei ist, die die Software beschreibt, die nun Frabnaggilywort heißt. Wir können das tatsächlich überprüfen, indem wir die Kombination aus expliziter Peg-Revision und expliziter operativer Revision verwenden. Wir wissen, dass in `HEAD` das Projekt Frabnaggilywort im Verzeichnis `frabnaggilywort` liegt. Also geben wir an, dass wir sehen möchten, wie sich die Historie in Revision 1 identifizierte, die in `HEAD` als `frabnaggilywort/IDEA` bekannt ist.

```
$ svn cat -r 1 frabnaggilywort/IDEA@HEAD
The idea behind this project is to come up with a piece of software
that can frab a naggily wort.  Frabbing naggily worts is tricky
business, and doing it incorrectly can have serious ramifications, so
we need to employ over-the-top input validation and data verification
mechanisms.
```

Auch brauchen Peg und operative Revisionen nicht so trivial zu sein. Sagen wir beispielsweise, das Verzeichnis `frabnaggilywort` sei aus `HEAD` gelöscht, wir wissen aber, dass es in Revision 20 existierte, und wir wollen die Unterschiede der Datei `IDEA` zwischen den Revisionen 4 und 10 sehen. Wir können Peg-Revision 20 in Verbindung mit dem URL, verwenden, der sich auf Frabnaggilyworts Datei `IDEA` in Revision 20 bezog, und dann 4 und 10 als operativen Revisionsbereich verwenden.

```
$ svn diff -r 4:10 http://svn.red-bean.com/projects/frabnaggilywort/IDEA@20
Index: frabnaggilywort/IDEA
===================================================================
--- frabnaggilywort/IDEA (revision 4)
+++ frabnaggilywort/IDEA (revision 10)
@@ -1,5 +1,5 @@
```

```
-The idea behind this project is to come up with a piece of software
-that can frab a naggily wort.  Frabbing naggily worts is tricky
-business, and doing it incorrectly can have serious ramifications, so
-we need to employ over-the-top input validation and data verification
-mechanisms.
+The idea behind this project is to come up with a piece of
+client-server software that can remotely frab a naggily wort.
+Frabbing naggily worts is tricky business, and doing it incorrectly
+can have serious ramifications, so we need to employ over-the-top
+input validation and data verification mechanisms.
```

Glücklicherweise sind die meisten Leute nicht von solch komplizierten Situationen betroffen. Sollte das für Sie aber zutreffen, denken Sie daran, dass es sich bei Peg-Revisionen um diesen extra Hinweis handelt, den Subversion benötigt, um Mehrdeutigkeiten zu beseitigen.

Eigenschaften

Wir haben bereits detailliert besprochen, wie Subversion unterschiedliche Versionen von Dateien und Verzeichnissen im Projektarchiv ablegt und wieder herausholt. Ganze Kapitel haben sich dieser fundamentalen Funktionalität des Werkzeugs gewidmet. Falls die Versionierungsunterstützung an diesem Punkt aufhörte, wäre Subversion aus Versionskontrollperspektive immer noch vollständig.

Aber sie hört hier noch nicht auf.

Zusätzlich zur Versionierung Ihrer Verzeichnisse und Dateien liefert Subversion Schnittstellen zum Hinzufügen, Ändern und Entfernen versionierter Metadaten zu allen versionierten Dateien und Verzeichnissen. Wir bezeichnen diese Metadaten als *Eigenschaften*. Sie sind so etwas wie Tabellen mit zwei Spalten, die Namen von Eigenschaften auf beliebige Werte abbilden und an jedes Objekt Ihrer Arbeitskopie gehängt werden. Im Allgemeinen können Sie die Namen und Werte der Eigenschaften frei bestimmen, mit der Einschränkung, dass die Namen nur aus ASCII-Zeichen bestehen dürfen. Und das Beste an diesen Eigenschaften ist, dass auch sie genauso versioniert sind wie der textuelle Inhalt Ihrer Dateien. Sie können Änderungen an Eigenschaften ebenso einfach editieren, übergeben oder rückgängig machen wie Änderungen an Dateiinhalten. Das Versenden und Empfangen von Änderungen an Eigenschaften geschieht im Rahmen Ihrer typischen Übergabe- und Aktualisierungstätigkeiten – Sie müssen hierfür Ihre grundlegenden Prozesse nicht anpassen.

Subversion hat die Menge aller Eigenschaften die mit `svn:` beginnen für sich reserviert. Obwohl heute nur eine handvoll dieser Eigenschaften in Gebrauch sind, sollten Sie es vermeiden, spezielle Eigenschaften für Ihren Gebrauch zu erzeugen, die diesen Präfix besitzen. Sonst laufen Sie Gefahr, dass ein künftiger Stand von Subversion ein Verhalten oder eine Funktionalität beinhaltet, die durch eine Eigenschaft gleichen Namens beeinflusst wird, aber vielleicht mit einer völlig anderen Auslegung.

Eigenschaften tauchen auch an einer anderen Stelle von Subversion auf. So wie Dateien und Verzeichnisse mit beliebigen Eigenschafts-Namen und -Werten versehen werden können, kann auch jede Revision als Ganzes beliebige Eigenschaften bekommen. Die selben Einschränkungen gelten auch hier – menschenlesbare Namen und beliebige binäre Werte. Der Hauptunterschied ist, dass Revisions-Eigenschaften unversioniert sind. Mit anderen Worten: falls Sie den Wert einer Revisions-Eigenschaft ändern oder die Eigenschaft löschen, gibt es mit Subversion Bordmitteln keine Möglichkeit, den ursprünglichen Wert wiederherzustellen.

Subversion besitzt keine besondere Richtlinie zur Verwendung von Eigenschaften. Es verlangt nur, dass Sie keine Namen für Eigenschaften verwenden, die den Präfix `svn:` haben, da dieser Namensraum für seine eigene Verwendung reserviert ist. Und Subversion benutzt tatsächlich Eigenschaften – sowohl die versionierten als auch die unversionierten. Bestimmte versionierte Eigenschaften haben eine besondere Bedeutung oder Auswirkungen wenn sie an Dateien und Verzeichnissen hängen oder sie beinhalten eine spezielle Information über die Revision mit der sie verbunden sind. Bestimmte Revisions-Eigenschaften werden automatisch bei der Übergabe an Revisionen gehängt; sie beinhalten Informationen über die Revision. Die meisten dieser Eigenschaften werden an einer anderen Stelle in diesem Kapitel oder in anderen Kapiteln im Rahmen allgemeinerer Themen erwähnt, mit denen sie zusammenhängen. Eine erschöpfende Aufstellung der vordefinierten Eigenschaften von Subversion finden Sie in „Subversion-Eigenschaften“.

Während Subversion automatisch Eigenschaften (`svn:date`, `svn:author`, `svn:log` usw,) an Revisionen hängt, setzt es nachher die Existenz dieser Eigenschaften *nicht* voraus, und ebensowenig sollten Sie es oder die Werkzeuge, die Sie verwenden, um mit dem Projektarchiv zu interagieren. Revisionseigenschaften können programmatisch oder mit dem Client gelöscht werden (wenn die Hooks des Projektarchivs das erlauben), ohne dass die Funktionsfähigkeit von Subversion eingeschränkt würde. Wenn Sie also Scripte schreiben, die mit den Daten des Subversion-Projektarchivs arbeiten, sollten Sie nicht den Fehler machen, anzunehmen, das eine bestimmte Eigenschaft einer Revision vorhanden ist.

In diesem Abschnitt untersuchen wir den Nutzen der Unterstützung von Eigenschaften – sowohl für den Anwender von Subversion als auch für Subversion selbst. Sie werden die Unterbefehle von **svn** kennenlernen, die mit Eigenschaften zu tun haben und wie Änderungen an Eigenschaften sich auf Ihren normalen Umgang mit Subversion auswirken.

Warum Eigenschaften?

Ebenso wie Subversion Eigenschaften verwendet, um zusätzliche Informationen über die enthaltenen Dateien, Verzeichnisse und Revisionen zu speichern, könnten Eigenschaften auch für Sie ähnlich von Nutzen sein. Sie werden es vielleicht als nützlich ansehen, wenn Sie in der Nähe Ihrer versionierten Daten spezielle Metadaten dazu unterbringen können.

Nehmen wir mal an, Sie möchten eine Webpräsenz entwerfen, die viele digitale Fotos mit Bildunterschrift und Zeitstempel anzeigen soll. Da sich die Menge Ihrer Fotos ständig ändert, möchten Sie soviel wie möglich automatisieren. Die Fotos können ziemlich groß werden, so dass Sie den Besuchern Ihrer Seite Miniaturvorschaubilder anbieten möchten.

Natürlich können Sie diese Funktionalität auch mit herkömmlichen Dateien hinbekommen. Das bedeutet, Sie haben `image123.jpg` und `image123-thumbnail.jpg` gemeinsam in einem Verzeichnis. Oder Sie speichern die Vorschaubildchen in einem anderen Verzeichnis, etwa `thumbnails/image123.jpg`, falls Sie die gleichen Dateinamen beibehalten möchten. Sie können auch die Bildunterschriften und Zeitstempel auf ähnliche Weise speichern, ebenso vom Originalbild getrennt. Das Problem hierbei ist jedoch, dass sich Ihre Ansammlung an Dateien mit jedem neu hinzugefügten Bild vervielfältigt.

Betrachten Sie nun dieselbe Webpräsenz, eingerichtet unter Verwendung der Datei-Eigenschaften von Subversion. Stellen Sie sich vor, sie hätten eine einzelne Bilddatei `image123.jpg` mit Eigenschaften namens `Unterschrift`, `Zeitstempel` und sogar `Vorschaubild`. Jetzt sieht Ihr Verzeichnis viel überschaubarer aus – tatsächlich sieht es für den flüchtigen Betrachter aus, als befänden sich dort nur Bilddateien. Ihre Automatisierungs-Skripte wissen es jedoch besser. Sie wissen, dass sie **svn** verwenden können (oder noch besser, die Subversion-Sprachschnittstellen – siehe „Benutzung der APIs“), um die von Ihrer Webpräsenz zusätzlich benötigten Informationen herauszuholen, ohne eine Indexdatei lesen oder Pfad-Umbenennungs-Spielereien machen zu müssen.

Obwohl Subversion kaum Einschränkungen für die von Ihnen verwendeten Namen und Werte für Eigenschaften macht, ist es nicht entworfen worden, um optimal mit großen Eigenschafts-Werten oder umfangreichen Eigenschafts-Mengen für eine bestimmte Datei oder ein Verzeichnis umgehen zu können. Gewöhnlich behält Subversion gleichzeitig alle Eigenschafts-Namen und -Werte im Speicher, die zu einem einzelnen Objekt gehören, was bei umfangreichen Eigenschafts-Mengen zu erheblichen Leistungseinbußen oder fehlgeschlagenen Operationen führen kann.

Spezielle Revisions-Eigenschaften werden auch sehr oft genutzt. Häufig wird eine Eigenschaft verwendet, deren Wert eine Fehlernummer eines Fehlerverwaltungssystem ist, mit dem die Revision in Beziehung gebracht wird, etwa weil eine mit dieser Revision vorgenommene Änderung den entsprechenden Fehler behebt. Andere Anwendungsfälle beinhalten die Vergabe anwenderfreundlicher Namen für die Revision – es könnte schwierig sein, sich zu erinnern, dass Revision 1935 vollständig getestet war. Wenn es jedoch eine Eigenschaft `Testergebnis` mit dem Wert `alles bestanden` für diese Revision gibt, ist das eine durchaus nützliche Information. Subversion erlaubt es Ihnen auf einfache Weise mit der Option `--with-revprop` des Befehls **svn commit**:

```
$ svn commit -m "Fix up the last remaining known regression bug." \
      --with-revprop "test-results=all passing"
Sende          lib/crit_bits.c
Übertrage Daten .
Revision 912 übertragen.
```

$

Suchbarkeit (oder, warum *nicht* bei Eigenschaften)

Bei allem Nutzen haben Subversion-Eigenschaften – oder, um genauer zu sein, die für sie verfügbaren Schnittstellen – eine gravierende Schwäche: während es eine einfache Sache ist, eine spezielle Eigenschaft *anzulegen*, ist das spätere *Auffinden* dieser Eigenschaft eine ganz andere Geschichte.

Der Versuch, eine spezielles Revisions-Eigenschaft aufzufinden, bedeutet im Allgemeinen ein lineares Durchwandern aller Revisionen im Projektarchiv, wobei jede Revision gefragt wird „Besitzt Du die Eigenschaft, nach der ich suche?“ Verwenden Sie die Option `--with-all-revprops` zusammen mit dem XML-Ausgabemodus des Befehls **svn log**, um diese Suche zu ermöglichen. Beachten Sie das Vorkommen der speziellen Eigenschaft `testresults` in der folgenden Ausgabe:

```
$ svn log --with-all-revprops --xml lib/crit_bits.c
<?xml version="1.0"?>
<log>
<logentry
   revision="912">
<author>harry</author>
<date>2011-07-29T14:47:41.169894Z</date>
<msg>Fix up the last remaining known regression bug.</msg>
<revprops>
<property
   name="testresults">all passing</property>
</revprops>
</logentry>
…
$
```

Der Versuch, eine spezielle versionierte Eigenschaft zu finden ist ebenfalls mühsam und erfordert oft ein rekursives **svn propget** über eine vollständige Arbeitskopie. In Ihrem Fall mag es nicht so schlimm sein wie ein lineares Durchwandern aller Revisionen; gleichwohl lässt es viel an Effizienz und Erfolgswahrscheinlichkeit vermissen, besonders wenn der Umfang Ihrer Suche eine Arbeitskopie ab der Wurzel Ihres Projektarchivs umfassen würde.

Aus diesem Grund sollten Sie – besonders im Anwendungsfall mit der Revisions-Eigenschaft – einfach Ihre Metadaten der Protokollnachricht hinzufügen, wobei ein vorgegebenes (und vielleicht programmtechnisch erzwungenes) Format verwendet wird, das es erlaubt, die Metadaten schnell aus der Ausgabe von **svn log** herauszulesen. Das Folgende zeigt eine weit verbreitete Form von Protokollnachrichten in Subversion:

```
Issue(s): IZ2376, IZ1919
Reviewed by:  sally

This fixes a nasty segfault in the wort frabbing process
…
```

Allerdings haben wir hier auch wieder Pech. Subversion verfügt noch nicht über einen Vorlagenmechanismus für Protokollnachrichten, der Benutzern sehr behilflich beim konsistenten Formatieren Ihrer in der Protokollnachricht eingebetteten Revisions-Metadaten sein würde.

Ändern von Eigenschaften

Das Programm **svn** gestattet es, Datei und Verzeichnis-Eigenschaften auf verschiedene Weise hinzuzufügen oder zu ändern. Für Eigenschaften mit kurzen, menschenlesbaren Werten ist es vermutlich am einfachsten, eine neue Eigenschaft zu vergeben, indem deren Name und Wert auf der Kommandozeile für den Unterbefehl **svn propset** angegeben wird:

```
$ svn propset copyright '(c) 2006 Red-Bean Software' calc/button.c
Eigenschaft »copyright« für »calc/button.c« gesetzt
$
```

Jedoch haben wir die von Subversion gebotene Flexibilität bei der Behandlung Ihrer Eigenschafts-Werte angepriesen; und wenn Sie planen, einen mehrzeiligen textuellen oder sogar binären Eigenschafts-Wert zu verwenden, wollen Sie diesen wahrscheinlich nicht auf den Kommandozeile angeben. Deshalb besitzt der Unterbefehl **svn propset** eine Option `--file` (`-F`), um den Namen einer Datei angeben zu können, deren Inhalt den neuen Eigenschafts-Wert ergibt.

```
$ svn propset license -F /path/to/LICENSE calc/button.c
Eigenschaft »license« für »calc/button.c« gesetzt
$
```

Es gibt einige Einschränkungen für die Vergabe von Namen für Eigenschaften. Ein Eigenschafts-Name muss mit einem Buchstaben, einem Doppelpunkt (`:`) oder einem Unterstrich (_) beginnen; danach können Sie auch Ziffern, Bindestriche (`-`) und Punkte (`.`) verwenden.[3]

Zusätzlich zum Befehl **propset** bietet das Programm **svn** den Befehl **propedit**. Dieser Befehl verwendet den konfigurierten Editor (siehe „Config“), um Eigenschaften hinzuzufügen oder zu ändern. Wenn Sie den Befehl aufrufen, startet **svn** Ihren Editor mit einer temporären Datei, die den gegenwärtigen Wert der Eigenschaft beinhaltet (oder leer ist, falls Sie eine neue Eigenschaft hinzufügen). Dann bearbeiten Sie diesen Wert in Ihrem Editor bis er dem Wert entspricht, den Sie für die Eigenschaft verwenden möchten, speichern die Datei und beenden den Editor. Falls Subversion feststellt, dass Sie tatsächlich den bestehenden Wert der Eigenschaft geändert haben, wird das als neuer Wert angenommen. Wenn Sie Ihren Editor ohne Änderungen beenden, wird die Eigenschaft nicht verändert:

```
$ svn propedit copyright calc/button.c  ### Editor ohne Änderung verlassen
Keine Änderungen der Eigenschaft »copyright« für »calc/button.c«
$
```

Hier sollten wir anmerken, dass die **svn**-Unterbefehle, die mit Eigenschaften zu tun haben, ähnlich wie bei anderen Unterbefehlen, auch auf mehrere Pfade gleichzeitig angewendet werden können. Dadurch wird es Ihnen ermöglicht, mit einem Befehl die Eigenschaften auf einer Menge von Dateien zu bearbeiten. Wir hätten beispielsweise das Folgende machen können:

```
$ svn propset copyright '(c) 2006 Red-Bean Software' calc/*
Eigenschaft »copyright« für »calc/Makefile« gesetzt
Eigenschaft »copyright« für »calc/button.c« gesetzt
Eigenschaft »copyright« für »calc/integer.c« gesetzt
…
$
```

Das ganze Hinzufügen und Bearbeiten von Eigenschaften ist nicht gerade nützlich, falls an die gespeicherten Werte nicht einfach heranzukommen ist. Also bietet das Programm **svn** zwei Unterbefehle zur Anzeige der Namen und Werte von Eigenschaften an Dateien und Verzeichnissen. Der Befehl **svn proplist** listet alle Eigenschafts-Namen auf, die für einen Pfad vergeben sind. Sobald Sie die Eigenschafts-Namen auf einem Element kennen, können Sie die Werte einzeln mittels **svn**

[3] Wenn Sie XML kennen, so ist das etwa die ASCII-Teilmenge der Syntax für die Produktion „Name“ in XML.

propget abrufen. Wenn diesem Befehl ein Eigenschafts-Name und ein Pfad (oder eine Menge von Pfaden) mitgegeben wird, wird der Wert der Eigenschaft nach Standardausgabe geschrieben.

```
$ svn proplist calc/button.c
Eigenschaften zu »calc/button.c«:
  copyright
  license
$ svn propget copyright calc/button.c
(c) 2006 Red-Bean Software
```

Es gibt sogar eine Variante des Befehls **proplist**, die es erlaubt, sowohl die Namen als auch die Werte aller Eigenschaften auszugeben. Übergeben Sie einfach die Option `--verbose` (`-v`).

```
$ svn proplist -v calc/button.c
Eigenschaften zu »calc/button.c«:
  copyright
    (c) 2006 Red-Bean Software
  license
    ================================================================
    Copyright (c) 2006 Red-Bean Software.  All rights reserved.

    Redistribution and use in source and binary forms, with or without
    modification, are permitted provided that the following conditions
    are met:

    1. Redistributions of source code must retain the above copyright
    notice, this list of conditions, and the recipe for Fitz's famous
    red-beans-and-rice.
    …
```

Der letzte Unterbefehl, der mit Eigenschaften zu tun hat, ist **propdel**. Da Subversion Eigenschaften mit leeren Werten erlaubt, können Sie eine Eigenschaft nicht vollständig mit **svn propedit** oder mit **svn propset** entfernen. So hat beispielsweise dieser Befehl *nicht* den erwünschten Effekt:

```
$ svn propset license "" calc/button.c
Eigenschaft »license« für »calc/button.c« gesetzt
$ svn proplist -v calc/button.c
Eigenschaften zu »calc/button.c«:
  copyright
    (c) 2006 Red-Bean Software
  license

$
```

Zum vollständigen Löschen von Eigenschaften müssen Sie den Unterbefehl **propdel** verwenden. Die Syntax ist den anderen Eigenschafts-Befehlen ähnlich:

```
$ svn propdel license calc/button.c
Eigenschaft »license« wurde von »calc/button.c« gelöscht.
$ svn proplist -v calc/button.c
Eigenschaften zu »calc/button.c«:
  copyright
    (c) 2006 Red-Bean Software
$
```

Können Sie sich an diese unversionierten Revisions-Eigenschaften erinnern? Auch diese können Sie mit den eben beschriebenen Unterbefehlen von **svn** verändern. Geben Sie einfach den Kommandozeilenparameter `--revprop` an und die Revision, deren Eigenschaft Sie ändern möchten. Da Revisionen global sind, brauchen Sie keinen Zielpfad für diese Eigenschafts-Unterbefehle anzugeben sofern Sie sich in einer Arbeitskopie des Projektarchivs befinden, deren Revisions-Eigenschaft Sie ändern möchten. Anderenfalls können Sie den URL irgendeines Pfades im entsprechenden Projektarchiv angeben (inklusive des Wurzelverzeichnisses des Projektarchivs). Sie möchten beispielsweise die Protokollnachricht einer bestehenden Revision ändern.[4] Falls Ihr aktuelles Arbeitsverzeichnis Teil einer Arbeitskopie Ihres Projektarchivs ist, können Sie einfach den Befehl **svn propset** ohne Zielpfad aufrufen:

```
$ svn propset svn:log "* button.c: Fix a compiler warning." -r11 --revprop
Eigenschaft »svn:log« wurde für Revision »11« im Projektarchiv gesetzt
$
```

Selbst wenn Sie keine Arbeitskopie aus diesem Projektarchiv ausgecheckt haben, können Sie dennoch die Änderung an der Eigenschaft durchführen, indem Sie den URL der Wurzel des Projektarchivs angeben:

```
$ svn propset svn:log "* button.c: Fix a compiler warning." -r11 --revprop \
            http://svn.example.com/repos/project
Eigenschaft »svn:log« wurde für Revision »11« im Projektarchiv gesetzt.
$
```

Beachten Sie, dass die Fähigkeit, diese unversionierten Eigenschaften zu verändern, ausdrücklich vom Administrator des Projektarchivs hinzugefügt werden muss (siehe „Berichtigung des Protokolleintrags“). Das geschieht aus dem Grund, dass diese Eigenschaften nicht versioniert sind, und Sie Gefahr laufen, durch unvorsichtiges Bearbeiten Informationen zu verlieren. Der Administrator des Projektarchivs kann Schutzmaßnahmen gegen diesen Verlust ergreifen, und standardmäßig ist die Veränderung unversionierter Eigenschaften nicht freigeschaltet.

Benutzer sollten nach Möglichkeit **svn propedit** statt **svn propset** verwenden. Während das Endergebnis dieser Befehle identisch ist, wird bei ersterem der aktuelle Wert der zu ändernden Eigenschaft angezeigt, was dabei hilft, sicherzustellen, dass die Änderung auch die beabsichtigte war. Das gilt besonders für unversionierte Revisions-Eigenschaften. Darüber hinaus ist es auch bedeutend einfacher, mehrzeilige Eigenschafts-Werte in einem Texteditor statt auf der Kommandozeile zu bearbeiten.

Eigenschaften und der Arbeitsablauf von Subversion

Jetzt, da Sie mit allen **svn**-Unterbefehlen vertraut sind, die mit Eigenschaften zu tun haben, wollen wir uns ansehen, welche Auswirkungen Änderungen an Eigenschaften auf den üblichen Arbeitsablauf von Subversion haben. Wie wir bereits früher erwähnten, sind Eigenschaften von Dateien und Verzeichnissen versioniert, genau so wie der Dateiinhalt. Deshalb bietet Subversion dieselben Möglichkeiten für das Zusammenführen der Änderungen anderer mit Ihren eigenen – sauber oder konfliktbehaftet.

Wie bei Dateiinhalten handelt es sich bei Ihren Eigenschafts-Änderungen um lokale Modifikationen, die erst dann dauerhaft werden, wenn Sie sie mittels **svn commit** an das Projektarchiv übergeben. Ihre Eigenschafts-Änderungen können auch leicht rückgängig gemacht werden – der Befehl **svn revert** bringt Ihre Dateien und Verzeichnisse wieder in den unbearbeiteten Zustand – und zwar Inhalt, Eigenschaften und alles andere. Auch können Sie durch die Benutzung der Befehls **svn status** und **svn diff** interessante Informationen über den Status Ihrer Datei- und Verzeichnis-Eigenschaften erhalten.

```
$ svn status calc/button.c
 M      calc/button.c
```

[4] Das Berichtigen von Rechtschreibfehlern, grammatischen Stolpersteinen und „einfacher Unrichtigkeiten“ in Protokollnachrichten ist vielleicht der häufigste Anwendungsfall für die Option `--revprop`.

```
$ svn diff calc/button.c
Eigenschaftsänderungen: calc/button.c
___________________________________________________________________
Name: copyright
   + (c) 2006 Red-Bean Software

$
```

Beachten Sie, dass der Unterbefehl **status** das `M` in der zweiten statt in der ersten Spalte anzeigt. Das geschieht, da wir zwar die Eigenschaften von `calc/button.c` verändert haben, nicht jedoch dessen Inhalt. Hätten wir beides geändert, würden wir `M` auch in der ersten Spalte sehen. (**svn status** behandeln wir in „Verschaffen Sie sich einen Überblick über Ihre Änderungen“).

Konflikte bei Eigenschaften

Wie bei Dateiinhalten kann es zu Konflikten zwischen lokalen Änderungen an Eigenschaften und übergebenen Änderungen anderer kommen. Wenn Sie das Verzeichnis Ihrer Arbeitskopie aktualisieren und Eigenschafts-Änderungen eines versionierten Objektes erhalten, die mit Ihren eigenen kollidieren, wird Subversion melden, dass sich das Objekt in einem Konfliktzustand befindet.

```
$ svn update calc
M  calc/Makefile.in
Konflikt für Eigenschaft »linecount« für »calc/button.c« entdeckt.
Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
         (h) Hilfe für weitere Optionen: p
 C calc/button.c
Aktualisiert zu Revision 143.
Konfliktübersicht:
  Eigenschaftskonflikte: 1
$
```

Subversion erzeugt im selben Verzeichnis des konfliktbehafteten Objektes eine Datei mit der Endung `.prej`, die Einzelheiten zum Konflikt beinhaltet. Sie sollten sich den Inhalt genau ansehen, um entscheiden zu können, wie der Konflikt aufgelöst werden kann. Bis der Konflikt aufgelöst ist wird in der zweiten Spalte der Ausgabe von **svn status** für dieses Objekt ein `C` angezeigt und Versuche, Ihre lokalen Änderungen zu übergeben, werden fehlschlagen.

```
$ svn status calc
 C      calc/button.c
?       calc/button.c.prej
$ cat calc/button.c.prej
Versuch, die Eigenschaft »linecount« von »1267« in »1301« zu ändern,
aber die Eigenschaft wurde lokal von »1267« in »1256« geändert.
$
```

Um Eigenschafts-Konflikte aufzulösen, stellen Sie einfach sicher, dass die konfliktbehafteten Eigenschaften die passenden Werte enthalten und verwenden dann den Befehl **svn resolve --accept=working**, um Subversion mitzuteilen, dass Sie das Problem manuell gelöst haben.

Sie haben vielleicht auch die ungewöhnliche Art und Weise bemerkt, wie Subversion momentan Unterschiede von Eigenschaften darstellt. Sie können immer noch **svn diff** verwenden und dessen Ausgabe umleiten, um eine Patch-Datei zu erzeugen. Das Programm **patch** ignoriert Patches für Eigenschaften – es ignoriert regelmäßig alles, was es nicht versteht. Das bedeutet leider, dass für die vollständige Anwendung eines durch **svn diff** erzeugten Patches sämtliche Änderungen an Eigenschaften manuell nachgezogen werden müssen.

Automatisches Setzen von Eigenschaften

Eigenschaften sind eine mächtige Funktionalität von Subversion, die als Schlüsselkomponenten zahlreicher Subversion-Funktionen agieren, welche an anderer Stelle in diesem und in anderen Kapiteln erörtert werden – Unterstützung textueller Diffs und Zusammenführungen, Ersetzung von Schlüsselworten, Umwandlung von Zeilenenden, usw. Um jedoch den größten Nutzen aus Eigenschaften ziehen zu können, müssen sie auf den richtigen Dateien und Verzeichnissen gesetzt sein. Leider kann dieser Schritt leicht in der täglichen Routine vergessen werden, besonders deshalb, da das Versäumen des Setzens einer Eigenschaft normalerweise nicht einen offensichtlichen Fehler zur Folge hat (zumindest im Vergleich zu einer Datei, bei der versäumt wurde, sie unter Versionskontrolle zu stellen). Um Ihnen dabei zu helfen, die Eigenschaften an die Stellen zu bringen, wo sie nötig sind, bietet Subversion Ihnen ein paar einfache aber nützliche Funktionen.

Immer wenn Sie eine Datei mit **svn add** oder **svn import** unter Versionskontrolle nehmen, versucht Subversion, Sie zu unterstützen, indem es einige übliche Datei-Eigenschaften automatisch setzt. Erstens setzt Subversion auf Betriebssystemen, deren Dateisystem ein Ausführbarkeits-Erlaubnis-Bit unterstützt, automatisch die Eigenschaft `svn:executable` auf neu hinzugefügte oder importierte Dateien, bei denen das Ausführbarkeits-Bit gesetzt ist. (Siehe „Ausführbarkeit von Dateien“ weiter unten in diesem Kapitel für weitere Informationen zu dieser Eigenschaft.)

Zweitens versucht Subversion den MIME-Typen der Datei zu ermitteln. Falls Sie einen Laufzeitparameter `mime-types-files` konfiguriert haben, versucht Subversion in dieser Datei einen passenden MIME-Typen für die Endung Ihrer Datei zu finden. Wenn es fündig wird, setzt es die Eigenschaft `svn:mime-type` Ihrer Datei auf den gefundenen MIME-Typen. Falls keine Datei konfiguriert oder kein passender Typ für die Dateiendung gefunden wurde, verwendet Subversion stattdessen seine recht einfache Heuristik, um festzustellen, ob die Datei nicht-textuellen Inhalt hat. Falls das der Fall ist, wird automatisch die Eigenschaft `svn:mime-type` dieser Datei auf `application/octet-stream` gesetzt (der allgemeine „dies ist eine Ansammlung von Bytes“-MIME-Type). Falls Subversion falsch rät, oder falls Sie den Wert der Eigenschaft `svn:mime-type` präziser setzen möchten – etwa `image/png` oder `application/x-shockwave-flash` – können Sie natürlich jederzeit die Eigenschaft entfernen oder bearbeiten. (Mehr zur Verwendung von MIME-Typen durch Subversion in „Datei-Inhalts-Typ“ später in diesem Kapitel.)

UTF-16 wird häufig verwendet, um Dateien zu encoden, deren semantischer Inhalt zwar textueller Natur ist, die allerdings voller Bytes außerhalb des typischen ASCII-Zeichenraums sind. Als solche neigt Subversion dazu, diese Dateien als binär zu klassifizieren, sehr zum Leidwesen von Anwendern, die zeilenweise Unterschiedungen und Zusammenführungen, Schlüsselwortersetzungen und andere Verhaltensweisen für diese Dateien wünschen.

Darüber hinaus bietet Subversion über sein Laufzeit-Konfigurationssystem (siehe „Laufzeit-Konfigurationsbereich“) eine flexible Möglichkeit, automatisch Eigenschaften zu setzen, die es Ihnen erlaubt, Abbildungen von Dateinamensmustern auf Eigenschafts-Namen und -Werte vorzunehmen. Auch hier haben diese Abbildungen Auswirkungen auf das Hinzufügen und Importieren und können nicht nur die standardmäßigen Entscheidungen Subversions bezüglich des Mime-Typs außer Kraft setzen, sondern auch zusätzliche Subversion- oder spezielle Eigenschaften setzen. Beispielsweise könnten Sie eine Abbildung definieren, die bestimmt, dass jedes Mal wenn eine JPEG-Datei hinzugefügt wird – Dateien, deren Namen auf das Muster `*.jpg` passen – Subversion automatisch die Eigenschaft `svn:mime-type` für diese Dateien auf `image/jpeg` setzt. Oder vielleicht sollen alle Dateien mit dem Muster `*.cpp` `svn:eol-style` auf `native` und `svn:keywords` auf `Id` gesetzt bekommen. Die Unterstützung automatischer Eigenschaften ist vielleicht das praktischste Werkzeug bezüglich Eigenschaften in der Werkzeugkiste von Subversion. Siehe „Config“ für Details zur Konfiguration dieser Unterstützung.

Subversion-Administratoren fragen häufig, ob es möglich ist, serverseitig eine Menge an Eigenschaftsdefinitionen zu konfigurieren, die alle Clients automatisch beachten, wenn sie auf Arbeitskopien arbeiten, die von diesem Server ausgecheckt worden sind. Leider bietet Subversion diese Möglichkeit nicht an. Administratoren können Hook-Scripts verwenden, um sicherzustellen, dass die Dateien und Verzeichnissen hinzugefügten und geänderten Eigenschaften den von den Administratoren vorgegebenen Richtlinien entsprechen, und Übergaben ablehnen, die dagegen verstoßen. (Siehe „Erstellen von Projektarchiv-Hooks“ für mehr zu Hook-Scripten.) Es besteht aber keinerlei Möglichkeit, diese Richtlinien im Voraus für Subversion-Clients für verbindlich zu erklären.

Datei-Portabilität

Glücklicherweise verhält sich das Kommandozeilenprogramm von Subversion für routinemäßig auf verschiedenen Rechnern mit unterschiedlichen Betriebssystemen arbeitende Benutzer unter all diesen Betriebssystemen fast identisch. Wenn Sie wissen, wie **svn** auf der einen Plattform eingesetzt wird, wissen Sie auch, wie es woanders geht.

Jedoch trifft das nicht immer auf andere allgemeine Klassen von Software zu oder die eigentlichen Dateien, die Sie mit Subversion verwalten. Beispielsweise ist die Definition einer „Textdatei“ auf einer Windows-Maschine ähnlich der auf einer Linux-Kiste, jedoch mit einem wichtigen Unterschied: die Zeichenfolge zum Markieren der Zeilenenden dieser Dateien. Es gibt auch andere Unterschiede. Unix-Plattformen haben symbolische Links (die Subversion unterstützt), während es sie unter Windows nicht gibt. Unix-Plattformen verwenden Dateisystem-Berechtigungen, um die Ausführbarkeit zu ermitteln, während Windows Dateiendungen benutzt.

Da Subversion nicht in der Position ist, die gesamte Welt durch gemeinsame Definitionen und Implementierungen all dieser Dinge zu vereinen, ist das beste, was es tun kann, Ihr Leben zu vereinfachen, falls Sie mit Ihren versionierten Dateien und Verzeichnissen auf mehreren Rechnern und Betriebssystemen arbeiten müssen. Dieser Abschnitt beschreibt einige der Methoden, die Subversion hierfür verwendet.

Datei-Inhalts-Typ

Subversion reiht sich unter den zahlreichen Anwendungen ein, die die Multipurpose Internet Mail Extensions (MIME) Inhaltstypen erkennen und verwenden. Außer ein universeller Lagerort für den Inhaltstypen einer Datei zu sein, bestimmt der Wert der Datei-Eigenschaft `svn:mime-type` einige Verhaltensweisen von Subversion selbst.

Ermitteln von Dateitypen

Zahlreiche Programme unter den meisten modernen Betriebssystemen machen Annahmen über den Typen und das Format des Inhalts einer Datei basierend auf dem Dateinamen, insbesondere die Dateiendung. Beispielsweise wird von Dateien, deren Name auf `.txt` endet, generell davon ausgegangen, dass sie menschenlesbar sind, d.h., sie lassen sich durch einfaches Durchlesen verstehen, ohne komplexe Bearbeitungen zum Entziffern vornehmen zu müssen. Bei Dateien, die auf `.png` enden, wird jedoch davon ausgegangen, dass es sich um den Typ Portable Network Graphics handelt – überhaupt nicht menschenlesbar und nur sinnvoll, wenn eine Bearbeitung durch Software erfolgt, die das PNG Format versteht und diese Informationen in ein Rasterbild verwandelt.

Unglücklicherweise haben einige dieser Endungen im Lauf der Zeit ihre Bedeutungen geändert. Als PCs das erste Mal auftauchten, war eine Datei namens `README.DOC` so gut wie sicher eine einfache Textdatei, so wie heute die `.txt`-Dateien. Doch bis in die Mitte der 1990er konnten Sie beinahe wetten, dass eine Datei mit diesem Namen gar keine einfache Textdatei ist, sondern ein Microsoft-Word-Dokument in einem proprietären, nicht menschenlesbaren Format. Doch erfolgte diese Änderung nicht über Nacht – es gab sicherlich einen Zeitraum der Verwirrung für Rechnerbenutzer, wenn es darum ging, um was es sich bei einer `.DOC`-Datei eigentlich handelte.[5]

Die Beliebtheit der Vernetzung von Rechnern zog die Abbildung von Dateinamen auf Dateiinhalte noch stärker in Zweifel. Bei Informationen, die über Netze hinweg vertrieben werden und dynamisch durch Skripte auf Servern erzeugt werden, gab es oft gar keine Datei an sich, also auch keinen Dateinamen. So benötigten beispielsweise Web-Server eine andere Möglichkeit, um Browsern mitzuteilen, was sie herunterladen, damit der Browser etwas Vernünftiges mit dieser Information anfangen kann, ob die Daten mithilfe eines Programms angezeigt werden sollen, das für diesen Datentyp registriert ist, oder ob der Benutzer gefragt werden soll, wo auf seinem Rechner die heruntergeladenen Daten zu speichern sind.

Schließlich kam dabei ein Standard heraus, der, neben anderen Dingen, den Inhalt eines Datenstroms beschreibt. Im Jahr 1996 wurde RFC 2045 veröffentlicht. Es war der erste von fünf RFCs, die MIME definieren. Er erklärt das Konzept von Medientypen sowie Untertypen und empfiehlt eine Syntax zur Beschreibung für die Darstellungsweise dieser Typen. Heute werden MIME-Medientypen – oder „MIME-Typen“ – fast universell zwischen E-Mail-Anwendungen, Web-Servern und anderer Software als Standardmechanismus verwendet, um die Verwirrung um Dateiinhalte zu beseitigen.

Beispielsweise ist einer der Vorteile, die Subversion typischerweise mitbringt, die kontextabhängige, zeilenbasierte Zusammenführung der vom Server während einer Aktualisierung empfangenen Änderungen mit Ihrer Arbeitsdatei. Allerdings gibt es bei Dateien, deren Inhalt nicht aus Text besteht, oft nicht das Konzept einer „Zeile“. Damit versucht Subversion während einer Aktualisierung keine kontextabhängige Zusammenführungen für versionierte Dateien, deren Eigenschaft `svn:mime-type` auf einen nicht-textuellen MIME-Typen gesetzt ist (normalerweise etwas, das nicht mit `text/` beginnt, obwohl es Ausnahmen gibt). Stattdessen wird jedes Mal, wenn eine von Ihnen lokal veränderte binäre Datei in der Arbeitskopie aktualisiert werden soll, diese Datei nicht angerührt, sondern Subversion erzeugt zwei neue Dateien. Eine davon hat die Dateiendung `.oldrev` und beinhaltet die BASE-Revision der Datei. Die andere Datei hat eine `.newrev`-Endung und beinhaltet die aktualisierte Revision der Datei. Dieses Verhalten dient tatsächlich dem Schutz des Benutzers vor

[5] Sie glauben, das war gemein? Während der gleichen Epoche benutzte auch WordPerfect `.DOC` als bevorzugte Endung für sein proprietäres Dateiformat!

fehlgeschlagenen Versuchen, kontextabhängige Zusammenführungen mit Dateien zu machen, die einfach nicht kontextabhängig zusammengeführt werden können.

Falls die Eigenschaft `svn:mime-type` auf einen Wert gesetzt wird, der nicht auf textuellen Dateiinhalt schließen lässt, kann das einige unerwartete Auswirkungen in Verbindung mit anderen Eigenschaften haben. Da beispielsweise Zeilenenden (und somit die Umwandlung von Zeilenenden) bei nicht-textuellen Dateien keinen Sinn ergeben, verhindert Subversion, dass Sie die Eigenschaft `svn:eol-style` für diese Dateien setzen. Das ist offensichtlich, wenn es bei einer einzelnen Datei versucht wird – **svn propset** gibt einen Fehler aus und beendet sich. Allerdings könnte es nicht so klar sein, wenn Sie die Eigenschaften rekursiv setzen möchten, wobei Subversion stillschweigend diejenigen Dateien übergeht, die es für eine bestimmte Eigenschaft als untauglich erachtet.

Subversion stellt eine Anzahl von Mechanismen zur Verfügung, mit denen automatisch die Eigenschaft `svn:mime-type` an einer versionierten Datei gesetzt werden kann. Siehe „Automatisches Setzen von Eigenschaften“ für Details.

Falls die Eigenschaft `svn:mime-type` gesetzt ist, verwendet auch das Subversion-Apache-Modul dessen Wert, um den HTTP-Header `Content-type:` zu füllen, wenn es auf GET-Anfragen antwortet. Das gibt Ihrem Web-Browser einen wichtigen Hinweis darauf, wie die Datei darzustellen ist, wenn Sie sie benutzen, um den Inhalt Ihres Subversion-Projektarchivs zu durchstöbern.

Ausführbarkeit von Dateien

Unter vielen Betriebssystemen hängt die Fähigkeit, eine Datei als Befehl ausführen zu können, vom Vorhandensein eines Ausführbarkeits-Erlaubnis-Bits ab. Normalerweise ist dieses Bit standardmäßig nicht aktiviert und muss vom Benutzer für jede Datei gesetzt werden, die es benötigt. Es wäre aber ein Riesenstress, sich exakt diejenigen Dateien in einer frisch ausgecheckten Arbeitskopie merken zu müssen, die das Ausführbarkeits-Bit gesetzt haben sollen und es dann zu aktivieren. Deshalb stellt Subversion die Eigenschaft `svn:executable` zur Verfügung, um die Dateien zu markieren, die das Ausführbarkeits-Bit benötigen. Beim Auschecken berücksichtigt Subversion diesen Wunsch wenn es die Arbeitskopie mit solchen Dateien füllt.

Diese Eigenschaft hat keine Auswirkungen auf Dateisystemen, die das Konzept eines Ausführbarkeits-Bits nicht kennen, so wie FAT32 und NTFS[6]. Darüber hinaus erzwingt Subversion beim Setzen dieser Eigenschaft den Wert `*`, obwohl sie keine definierten Werte besitzt. Zum Schluss sei gesagt, dass diese Eigenschaft nur auf Dateien gültig ist, nicht auf Verzeichnissen.

Zeichenfolgen zur Zeilenende-Kennzeichnung

Falls der Inhalt einer versionierten Datei durch deren Eigenschaft `svn:mime-type` nicht anders gekennzeichnet ist, nimmt Subversion an, es handele sich um menschenlesbare Daten. Im Allgemeinen verwendet Subversion dieses Wissen lediglich, um festzustellen, ob Unterschiede kontextabhängig dargestellt werden können. Ansonsten sind für Subversion Bytes einfach Bytes.

Das bedeutet, dass Subversion von sich aus überhaupt nicht auf die von Ihren Dateien benutzte Sorte von *Zeilenende-Markierungen (EOL-Marker)* achtet. Leider haben unterschiedliche Betriebssysteme auch unterschiedliche Konventionen hinsichtlich der Zeichenfolgen, die das Ende einer Textzeile in einer Datei repräsentieren. Beispielsweise ist die gebräuchliche Zeilenende-Kennzeichnung, die von Software auf der Windows-Plattform benutzt wird, ein Paar aus ASCII-Kontrollzeichen – ein Wagenrücklauf (`CR`) gefolgt von einem Zeilenvorschub (`LF`). Unix Software verwendet jedoch nur das Zeichen `LF`, um Zeilenenden zu kennzeichnen.

Nicht alle der zahlreichen Werkzeuge unter diesen Betriebssystemen können mit Dateien umgehen, die Zeilenenden in einem Format haben, das vom *eigenen Zeilenende-Stil* des Betriebssystems abweicht, auf dem sie laufen. Unix-Programme behandeln das in Windows-Dateien vorkommende Zeichen `CR` typischerweise als ein gewöhnliches Zeichen (welches normalerweise als `^M` wiedergegeben wird), und Windows-Programme fügen alle Zeilen einer Unix-Datei zu einer großen Zeile zusammen, da keine Wagenrücklauf-Zeilenvorschub-Kombination (oder `CRLF`) zur Zeilenendemarkierung gefunden wurde.

Diese Empfindlichkeit gegenüber fremden EOL-Markern kann für Menschen frustrierend sein, die eine Datei über Betriebssystemgrenzen hinweg gemeinsam benutzen. Schauen wir uns beispielsweise eine Quelltextdatei an, die Entwickler sowohl unter Windows als auch unter Unix bearbeiten. Falls die Entwickler stets Werkzeuge verwenden, die den Zeilenende-Stil der Datei bewahren, werden keine Probleme auftreten.

[6] Die Windows-Dateisysteme benutzen Dateiendungen (wie etwa `.EXE`, `.BAT` und `.COM`), um ausführbare Dateien zu kennzeichnen.

In der Praxis jedoch scheitern viele verbreitete Werkzeuge daran, eine Datei mit fremden EOL-Markern richtig zu lesen, oder sie wandeln die Zeilenenden der Datei beim Schreiben in den eigenen Stil. Falls ersteres für einen Entwickler zutrifft, muss ein externes Umwandlungsprogramm verwendet werden (etwa **dos2unix** oder sein Gegenstück **unix2dos**), um die Datei für die Bearbeitung vorzubereiten. Im letzteren Fall ist keine spezielle Vorbereitung notwendig. In beiden Fällen jedoch ist das Ergebnis eine Datei, die sich buchstäblich in jeder Zeile vom Original unterscheidet. Vor der Übertragung seiner Änderungen hat der Benutzer zwei Möglichkeiten: Entweder kann er mit einem Umwandlungsprogramm den Zeilenende-Stil wiederherstellen, den die Datei vor den Änderungen aufwies, oder er überträgt die Datei einfach – neue EOL-Marker und alles andere inklusive.

Unter dem Strich bedeuten derartige Szenarien Zeitverschwendung und unnötige Änderungen an übertragenen Dateien. Zeitverschwendung ist schlimm genug. Falls jedoch durch die Übertragungen alle Zeilen einer Datei geändert werden, erschwert das die Aufgabe, herauszufinden, welche Zeilen sich auf eine nicht-triviale Art und Weise geändert haben. Wo wurde der Fehler tatsächlich behoben? In welcher Zeile hat sich ein Syntaxfehler eingeschlichen?

Die Lösung für dieses Problem ist die Eigenschaft `svn:eol-style`. Wird sie auf einen gültigen Wert gesetzt, benutzt Subversion sie, um festzustellen, welche besondere Behandlung für diese Datei notwendig ist, um das ständige durch unterschiedliche Betriebssysteme bedingte Hin und Her der Zeilenende-Stile bei jeder Übertragung zu vermeiden. Die gültigen Werte sind:

`native`
: Das führt dazu, dass die Datei die EOL-Marker enthält, die in dem Betriebssystem üblich sind, unter dem Subversion läuft. Mit anderen Worten: Falls ein Benutzer auf einem Windows-Rechner eine Arbeitskopie auscheckt, zu der eine Datei mit einer auf `native` gesetzten Eigenschaft `svn:eol-style` gehört, wird die Datei `CRLF`-EOL-Marker beinhalten. Ein Unix-Benutzer, der eine Arbeitskopie mit derselben Datei auscheckt, wird in seiner Kopie der Datei `LF`-EOL-Marker sehen.

  Beachten Sie, dass Subversion die Datei, unabhängig vom Betriebssystem, tatsächlich unter Verwendung normalisierter `LF`-EOL-Marker im Projektarchiv ablegt. Das geschieht jedoch grundsätzlich transparent für den Benutzer.

`CRLF`
: Das führt dazu, dass die Datei unabhängig vom Betriebssystem die Zeichenfolge `CRLF` als EOL-Marker enthält.

`LF`
: Das führt dazu, dass die Datei unabhängig vom Betriebssystem das Zeichen `LF` als EOL-Marker enthält.

`CR`
: Das führt dazu, dass die Datei unabhängig vom Betriebssystem das Zeichen `CR` als EOL-Marker enthält. Dieser Zeilenende-Stil ist nicht sehr verbreitet

Ignorieren unversionierter Objekte

Es besteht in jeder gegebenen Arbeitskopie die Wahrscheinlichkeit, dass sich neben all den versionierten Dateien und Verzeichnissen auch andere Dateien und Verzeichnisse befinden, die weder versioniert sind noch versioniert werden sollen. Texteditoren müllen Arbeitskopien mit Sicherungskopien zu, Software-Compiler erzeugen Zwischen-, oder gar Zieldateien, die Sie normalerweise nie versionieren würden. Auch Benutzer selbst legen verschiedene andere Dateien und Verzeichnisse dort ab, wo es ihnen passt, oft in versionskontrollierten Arbeitskopien.

Es ist albern, anzunehmen, dass Arbeitskopien von Subversion irgendwie immun gegen diese Art von Durcheinander und Verunreinigung seien. Tatsächlich sieht es Subversion als ein *Feature* an, dass seine Arbeitskopien lediglich gewöhnliche Verzeichnisse sind, genau wie unversionierte Dateibäume. Allerdings können diese nicht zu versionierenden Dateien und Verzeichnisse bisweilen Ärger für Subversion-Benutzer machen. Da beispielsweise die Befehle **svn add** und **svn import** standardmäßig rekursiv arbeiten und nicht wissen, welche der Dateien im Baum Sie versionieren möchten und welche nicht, ist es leicht möglich, dass etwas unbeabsichtigt unter Versionskontrolle gebracht wird. Und weil **svn status** standardmäßig jedes interessante Objekt einer Arbeitskopie aufzeigt – inklusive unversionierter Dateien und Verzeichnisse – kann dessen Ausgabe gerade dort ziemlich verrauscht sein, wo sich viele dieser Dinge befinden.

Also bietet Ihnen Subversion zwei Möglichkeiten, um ihm mitzuteilen, welche Dateien Sie lieber nicht beachten möchten. Die eine Möglichkeit verwendet das Laufzeit-Konfigurationssystem (siehe „Laufzeit-Konfigurationsbereich“) und wirkt sich deshalb auf alle Funktionen von Subversion aus, die diese Laufzeit-Einstellungen verwenden – im Allgemeinen diejenigen, die auf einem bestimmten Rechner oder durch einen bestimmten Benutzer ausgeführt werden. Die andere Möglichkeit nutzt

Subversions Unterstützung für Verzeichnis-Eigenschaften und ist enger an den versionierten Baum gebunden, so dass hiervon jeder betroffen ist, der eine Arbeitskopie dieses Baumes besitzt. Beide dieser Möglichkeiten benutzen *Dateimuster* (Ketten aus normalen Zeichen und solchen mit besonderer Bedeutung, die mit Dateinamen verglichen werden), um zu entscheiden, welche Dateien ignoriert werden können.

Das Laufzeit-Konfigurationssystem von Subversion stellt eine Option `global-ignores` zur Verfügung, dessen Wert eine Sammlung von Dateimustern ist, die durch Leerraum getrennt sind. Der Subversion-Client vergleicht diese Muster sowohl mit den Dateinamen der unter Versionskontrolle zu bringenden Kandidaten als auch mit den Namen der unversionierten Dateien, die **svn status** erkennt. Falls der Name irgendeiner Datei zu einem Muster passt, verhält sich Subversion grundsätzlich so, als würde die Datei gar nicht vorhanden sein. Das ist wirklich nützlich für die Sorte von Dateien, die Sie fast nie versionieren möchten, etwa Sicherheitskopien von Editoren wie die `*~`- und `.*~`-Dateien von Emacs.

Dateimuster in Subversion

Dateimuster (auch *Globs* oder *Shell-Jokerzeichen* genannt) sind Ketten aus Zeichen, die mit Dateinamen verglichen werden sollen, typischerweise für die schnelle Auswahl einer Teilmenge von Dateien aus einer größeren Ansammlung, ohne dabei jede einzelne Datei benennen zu müssen. Die Muster enthalten zwei Sorten von Zeichen: normale Zeichen, die wie angegeben mit möglichen Treffern verglichen werden und besondere Jokerzeichen, die für den Abgleich auf andere Weise interpretiert werden.

Es gibt verschiedene Typen von Dateimuster-Regeln, doch verwendet Subversion die auf Unix-Systemen verbreitete Variante, wie sie dort in der Systemfunktion `fnmatch` implementiert ist. Sie unterstützt die folgenden Jokerzeichen, die an dieser Stelle der Einfachheit halber beschrieben werden:

`?`
Passt auf ein beliebiges einzelnes Zeichen

`*`
Passt auf eine beliebige Zeichenkette, auch die leere

`[`
Beginn einer Zeichenklassen-Definition, die durch `]` abgeschlossen wird; passt auf eine Teilmenge von Zeichen

Das gleiche Verhalten beim Abgleich von Dateimustern können Sie bei der Eingabeaufforderung einer Unix-Shell beobachten. Es folgen einige Beispiele von Mustern für verschiedene Dinge:

```
$ ls   ### die Quelltextdateien des Buchs
appa-quickstart.xml             ch06-server-configuration.xml
appb-svn-for-cvs-users.xml      ch07-customizing-svn.xml
appc-webdav.xml                 ch08-embedding-svn.xml
book.xml                        ch09-reference.xml
ch00-preface.xml                ch10-world-peace-thru-svn.xml
ch01-fundamental-concepts.xml   copyright.xml
ch02-basic-usage.xml            foreword.xml
ch03-advanced-topics.xml        images/
ch04-branching-and-merging.xml  index.xml
ch05-repository-admin.xml       styles.css
$ ls ch*   ### die Kapitel des Buchs
ch00-preface.xml                ch06-server-configuration.xml
ch01-fundamental-concepts.xml   ch07-customizing-svn.xml
ch02-basic-usage.xml            ch08-embedding-svn.xml
ch03-advanced-topics.xml        ch09-reference.xml
ch04-branching-and-merging.xml  ch10-world-peace-thru-svn.xml
ch05-repository-admin.xml
$ ls ch?0-*   ### die Kapitel, deren Namen auf Null enden
ch00-preface.xml  ch10-world-peace-thru-svn.xml
$ ls ch0[3578]-*   ### die Kapitel, für die Mike zuständig ist
ch03-advanced-topics.xml   ch07-customizing-svn.xml
ch05-repository-admin.xml  ch08-embedding-svn.xml
$
```

Dateimuster-Abgleich ist zwar ein bisschen komplizierter als hier beschrieben, jedoch ist Beschränkung auf diese grundlegende Ebene für die Mehrheit der Subversion-Benutzer ausreichend.

Wenn die Eigenschaft `svn:ignore` an einem versionierten Verzeichnis auftritt, wird erwartet, dass der Wert eine Liste von (durch Zeilenvorschübe getrennten) Dateimustern enthält, die Subversion benutzen soll, um ignorierbare Objekte in diesem Verzeichnis zu bestimmen. Diese Dateimuster setzen nicht jene außer Kraft, die in der Laufzeit-Konfigurations-Option `global-ignores` gefunden werden, sondern werden an deren Liste angehängt. An dieser Stelle lohnt es sich, noch einmal darauf hinzuweisen, dass, anders als bei der Option `global-ignores`, die Muster der Eigenschaft `svn:ignore` nur für das Verzeichnis gelten, an dem die Eigenschaft gesetzt ist, auch nicht für irgendein Unterverzeichnis. Mit der Eigenschaft `svn:ignore` kann Subversion leicht angewiesen werden, Dateien zu ignorieren, die in diesem Verzeichnis der Arbeitskopie eines jeden Benutzers mit großer Wahrscheinlichkeit auftreten, so wie Compiler-Ausgaben oder – um ein Beispiel zu bringen, das diesem Buch angebrachter ist – die HTML-, PDF- oder PostScript-Dateien, die als Ergebnis der Umwandlung der DocBook-XML-Quelltext-Dateien in ein lesbareres Format erzeugt werden.

Die Unterstützung für ignorierbare Dateimuster in Subversion erstreckt sich lediglich auf die einmalige Handlung, unversionierte Dateien und Verzeichnisse unter Versionskontrolle zu stellen. Sobald ein Objekt unter Kontrolle von Subversion ist, haben die Ignorier-Muster keine Auswirkungen mehr auf das Objekt. Mit anderen Worten: erwarten Sie nicht, dass Subversion die Übertragung von Änderungen verhindert, die Sie an einer versionierten Datei vorgenommen haben, nur weil der Name dieser Datei auf ein Ignorier-Muster passt – Subversion beachtet *stets* alle seine versionierten Objekte.

Ignorier-Muster für CVS-Benutzer

Die Subversion-Eigenschaft `svn:ignore` gleicht in Syntax und Funktion der CVS-Datei `.cvsignore`. Wenn Sie eine CVS-Arbeitskopie nach Subversion migrieren, können Sie tatsächlich die Ignorier-Muster direkt migrieren, indem Sie die Datei `.cvsignore` als Eingabe für den Befehl **svn propset** verwenden:

```
$ svn propset svn:ignore -F .cvsignore .
Eigenschaft »svn:ignore« für ».« gesetzt
$
```

Es gibt allerdings einige Unterschiede in der Art und Weise wie CVS und Subversion Ignorier-Muster behandeln. Die beiden Systeme verwenden die Ignorier-Muster zu unterschiedlichen Zeiten, und es gibt leichte Abweichungen darin, worauf die Muster angewendet werden. Darüber hinaus versteht Subversion die Verwendung von `!` nicht als Rücksetz-Zeichen, um überhaupt keine Ignorier-Muster mehr zu haben.

Die globale Liste mit Ignorier-Mustern neigt dazu, mehr eine Sache des persönlichen Geschmacks zu sein und richtet sich eher nach der Werkzeugkette eines Benutzers als nach den Bedürfnissen einer bestimmten Arbeitskopie im einzelnen. Deshalb konzentriert sich der Rest dieses Abschnitts auf die Eigenschaft `svn:ignore` und ihre Verwendung.

Angenommen, Sie haben die folgende Ausgabe von **svn status**:

```
$ svn status calc
 M      calc/button.c
?       calc/calculator
?       calc/data.c
?       calc/debug_log
?       calc/debug_log.1
?       calc/debug_log.2.gz
?       calc/debug_log.3.gz
```

In diesem Beispiel haben Sie einige Änderungen an Eigenschaften von `button.c` vorgenommen, aber Sie haben in Ihrer Arbeitskopie auch einige unversionierte Dateien: das neueste Programm `calculator`, das Sie aus Ihrem Quelltext compiliert haben, eine Quelltextdatei namens `data.c` und eine Menge von Protokolldateien zur Fehlersuche. Sie wissen, dass das Build-System stets ein Programm `calculator` erzeugt.[7] Und Sie wissen, das die Testumgebung immer diese Protokolldateien hinterlässt. Das trifft auf alle Arbeitskopien dieses Projektes zu, nicht nur auf Ihre eigene. Und Sie wissen, dass Sie kein Interesse daran haben, diese Dinge bei jedem Aufruf von **svn status** zu sehen, Sie sind sich auch ziemlich sicher, dass sich andere auch nicht dafür interessieren. Also rufen Sie **`svn propedit svn:ignore calc`** auf, um dem Verzeichnis `calc` ein paar Ignorier-Muster hinzuzufügen.

```
$ svn propget svn:ignore calc
calculator
debug_log*
$
```

Nach dem Hinzufügen dieser Eigenschaft haben Sie nun eine Eigenschafts-Änderung für das Verzeichnis `calc`. Beachten Sie jedoch, was sonst noch anders an Ihrer **svn status**-Ausgabe ist:

```
$ svn status
 M      calc
 M      calc/button.c
?       calc/data.c
```

Nun fehlt der überflüssige Müll in der Ausgabe! Das compilierte Programm `calculator` und all diese Protokolldateien befinden sich immer noch in Ihrer Arbeitskopie; Subversion erinnert Sie nur nicht mehr ständig daran, dass sie vorhanden und unversioniert sind. Und nachdem nun der ganze Lärm aus der Anzeige verschwunden ist, verbleiben die fesselnderen Objekte – wie z.B. die Quelltextdatei `data.c`, die Sie wahrscheinlich vergessen hatten, unter Versionskontrolle zu stellen.

Natürlich ist dieser kompaktere Bericht des Zustandes Ihrer Arbeitskopie nicht der einzig verfügbare. Falls Sie wirklich die ignorierten Dateien im Bericht sehen möchten, können Sie Subversion die Option `--no-ignore` mitgeben:

```
$ svn status --no-ignore
 M      calc
 M      calc/button.c
I       calc/calculator
?       calc/data.c
I       calc/debug_log
I       calc/debug_log.1
I       calc/debug_log.2.gz
I       calc/debug_log.3.gz
```

Wie bereits früher erwähnt, wird die Liste der zu ignorierenden Dateimuster auch von **svn add** und **svn import** verwendet. Beide dieser Befehle haben zur Folge, dass Subversion damit beginnt, eine Menge von Dateien und Verzeichnissen zu verwalten. Statt den Benutzer zu zwingen, die Dateien aus einem Dateibaum auszuwählen, die unter Versionskontrolle gestellt werden sollen, verwendet Subversion die Ignorier-Muster – sowohl die globalen als auch die verzeichnisgebundenen Listen – um festzustellen, welche Dateien nicht im Rahmen einer größeren rekursiven Hinzufüge- oder Importaktion in das Versionskontrollsystem gebracht werden sollen. Auch hier können Sie wieder die Option `--no-ignore` verwenden, um Subversion mitzuteilen, die Ignorier-Listen zu ignorieren und auf allen vorhandenen Dateien und Verzeichnissen zu arbeiten.

Selbst wenn `svn:ignore` gesetzt ist, könnten Sie Probleme bekommen, falls Sie Shell-Jokerzeichen in einem Befehl verwenden. Shell-Jokerzeichen werden zu einer expliziten Liste aus Zielobjekten erweitert, bevor Subversion sie bearbeitet, so dass der Aufruf von **`svn` *`SUBCOMMAND`* `*`** genau so funktioniert wie der Aufruf

[7] Ist das nicht der eigentliche Zweck eines Build-Systems?

`svn SUBCOMMAND file1 file2 file3 …`. Im Fall des Befehls **svn add** hat das einen ähnlichen Effekt, wie die Option `--no-ignore` zu übergeben. Statt Jokerzeichen zu benutzen sollten Sie also **`svn add --force .`** verwenden, um eine größere Menge unversionierter Dinge für die Versionskontrolle vorzumerken. Das ausdrückliche Ziel stellt sicher, dass das aktuelle Verzeichnis nicht übersehen wird, weil es schon lange unter Versionskontrolle ist, und die Option `--force` veranlasst Subversion, sich durch dieses Verzeichnis zu arbeiten und unversionierte Dateien hinzuzufügen, wobei die Eigenschaft `svn:ignore` und die Laufzeit-Konfigurations-Variable `global-ignores` berücksichtigt werden. Stellen Sie sicher, dass Sie dem Befehl **svn add** auch die Option `--depth files` mitgeben, falls Sie zum Hinzufügen kein vollständig rekursives Durchwandern wünschen.

Ersetzung von Schlüsselworten

Subversion ist in der Lage, *Schlüsselworte* – nützliche dynamische Informationshäppchen zu einer versionierten Datei – im Dateiinhalt zu ersetzen. Schlüsselworte liefern im Allgemeinen Informationen zur letzten Änderung an der Datei. Da diese Information sich mit jeder Änderung der Datei auch ändert, noch wichtiger, *nachdem* sich die Datei ändert, ist es für jeden Prozess außer dem Versionskontrollsystem ein ziemlicher Aufwand, die Daten vollständig aktuell zu halten. Würde das den Autoren überlassen, veralteten die Informationen unausweichlich.

Nehmen wir beispielsweise an, dass Sie ein Dokument haben, in dem das Datum der letzten Änderung angezeigt werden soll. Sie könnten natürlich jedem Bearbeiter auferlegen, kurz vor dem Übertragen ihrer Änderungen, den Teil des Dokumentes, der das Änderungsdatum enthält, entsprechend anzupassen. Früher oder später jedoch wird jemand vergessen, das zu tun. Teilen Sie stattdessen Subversion mit, eine Schlüsselwort-Ersetzung mit dem Schlüsselwort `LastChangedDate` vorzunehmen. Sie kontrollieren, wo das Schlüsselwort in Ihrem Dokument eingefügt wird, indem Sie einen *Schlüsselwort-Anker* an die gewünschte Stelle der Datei setzen. Dieser Anker ist einfach eine Zeichenkette, die formatiert ist wie `$`*`KeywordName`*`$`.

Bei allen Schlüsselworten, die als Anker in Dateien verwendet werden, ist die Groß- und Kleinschreibung relevant. Sie müssen die korrekte Schreibung verwenden, damit das Schlüsselwort ersetzt wird. Sie sollten davon ausgehen, dass auch die Groß- und Kleinschreibung der Eigenschaftswerte von `svn:keywords` relevant ist – bestimmte Schlüsselworte werden dessen ungeachtet erkannt, jedoch wird abgeraten, von diesem Verhalten auszugehen.

Subversion definiert die Liste der Schlüsselworte, die für die Ersetzung verfügbar sind. Diese Liste enthält die folgenden Schlüsselworte, von denen einige Aliasnamen besitzen, die Sie auch verwenden können.

`Date`
: Dieses Schlüsselwort beschreibt den letzten bekannten Zeitpunkt einer Änderung dieser Datei im Projektarchiv und hat das Format `$Date: 2006-07-22 21:42:37 -0700 (Sat, 22 Jul 2006) $`. Es kann auch als `LastChangedDate` angegeben werden. Anders als das Schlüsselwort `Id`, das UTC verwendet, zeigt das Schlüsselwort `Date` Zeitpunkte in der örtlichen Zeitzone an.

`Revision`
: Dieses Schlüsselwort beschreibt die letzte bekannte Revision einer Änderung dieser Datei im Projektarchiv und sieht etwa so aus: `$Revision: 144 $`. Es kann auch als `LastChangedRevision` oder `Rev` angegeben werden.

`Author`
: Dieses Schlüsselwort beschreibt den letzten bekannten Autor einer Änderung dieser Datei im Projektarchiv und sieht etwa so aus: `$Author: harry $`. Es kann auch als `LastChangedBy` angegeben werden.

`HeadURL`
: Dieses Schlüsselwort beschreibt den vollständigen URL zur letzten Version der Datei im Projektarchiv und sieht etwa so aus: `$HeadURL: http://svn.example.com/repos/trunk/calc.c $`. Es kann zu `URL` abgekürzt werden.

`Id`
: Dieses Schlüsselwort ist eine komprimierte Kombination aus den anderen Schlüsselworten. Seine Ersetzung sieht etwa so aus: `$Id: calc.c 148 2006-07-28 21:30:43Z sally $`, und sie bedeutet, dass die Datei `calc.c` zuletzt in Revision 148 am Abend des 28. Juli 2006 von `sally` geändert wurde. Der angegebene Zeitpunkt ist in UTC, anders als beim Schlüsselwort `Date` (das die örtliche Zeitzone verwendet).

`Header`
: Dieses Schlüsselwort ist ähnlich zu `Id`, enthält aber den vollständigen URL der letzten Revision des Objektes, identisch zu `HeadURL`. Seine Ersetzung sieht etwa aus wie `$Header:`

```
http://svn.example.com/repos/trunk/calc.c 148 2006-07-28 21:30:43Z sally $.
```

Einige der vorangegangenen Beschreibungen verwenden Formulierungen wie „letzte bekannte“ oder Ähnliches. Denken Sie daran, dass die Schlüsselwort-Ersetzung vom Client vorgenommen wird und dieser nur Änderungen „kennt“, die im Projektarchiv stattgefunden haben, als Sie Ihre Arbeitskopie aktualisiert haben, um diese Änderungen zu bekommen. Falls Sie Ihre Arbeitskopie nie aktualisieren, werden Ihre Schlüsselworte niemals durch andere Werte ersetzt werden, auch wenn diese Dateien regelmäßig im Projektarchiv geändert werden.

Einfach einen Schlüsselwort-Anker in Ihre Datei einzufügen, bewirkt nichts. Subversion wird niemals versuchen, eine Textersetzung in Ihren Dateiinhalten vorzunehmen, falls Sie es nicht ausdrücklich dazu auffordern. Schließlich könnten Sie ja ein Dokument[8] über die Verwendung von Schlüsselworten schreiben und deshalb nicht wollen, dass Subversion Ihre schönen Beispiele nicht ersetzter Schlüsselwort-Anker ersetzt!

Um Subversion mitzuteilen, ob Schlüsselworte in einer bestimmten Datei ersetzt werden sollen, wenden wir uns wiederum den Unterbefehlen zu, die mit Eigenschaften zu tun haben. Die Eigenschaft `svn:keywords` an einer versionierten Datei kontrolliert, welche Schlüsselworte in dieser Datei ersetzt werden. Der Wert ist eine durch Leerzeichen getrennte Liste aus Schlüsselwort-Namen oder deren Aliasnamen.

Nehmen wir an, sie haben eine versionierte Datei namens `weather.txt`, die folgendermaßen aussieht:

```
Hier ist der neueste Bericht von der vordersten Front.
$LastChangedDate$
$Rev$
Cumulus-Wolken entstehen öfter, wenn der Sommer naht.
```

Ohne die Eigenschaft `svn:keywords` auf dieser Datei wird Subversion nichts Besonderes machen. Nun schalten wir die Ersetzung des Schlüsselwortes `LastChangedDate` ein.

```
$ svn propset svn:keywords "Date Author" weather.txt
Eigenschaft »svn:keywords« für »weather.txt« gesetzt
$
```

Nun haben Sie eine lokale Änderung an einer Eigenschaft der Datei `weather.txt` vorgenommen. Sie werden keine Änderungen am Dateiinhalt erkennen können (es sei denn, sie hätten einige vor dem Setzen der Eigenschaft gemacht). Beachten Sie, dass die Datei einen Anker für das Schlüsselwort `Rev` enthielt, wir dieses Schlüsselwort jedoch nicht in den Wert der von uns gesetzten Eigenschaft aufnahmen. Es ist Subversion ein Vergnügen, alle Aufforderungen zu ignorieren, Schlüsselworte zu ersetzen, die nicht in der Datei oder im Wert der Eigenschaft `svn:keywords` vorhanden sind.

Unmittelbar nachdem Sie diese Änderung der Eigenschaft übertragen haben, wird Subversion Ihre Arbeitsdatei mit dem neuen Ersatztext aktualisieren. Statt des Schlüsselwort-Ankers `$LastChangedDate$` werden Sie das Ergebnis der Ersetzung sehen. Das Ergebnis enthält auch das Schlüsselwort und wird weiterhin durch die Dollarzeichen ($) begrenzt. Und wie wir vorhergesehen hatten, wurde das Schlüsselwort `Rev` nicht ersetzt, da wir es nicht wollten.

Beachten Sie auch, dass wir die Eigenschaft `svn:keywords` auf `Date Author` setzten, der Schlüsselwort-Anker aber das Alias `$LastChangedDate$` verwendete und trotzdem korrekt erweitert wurde:

```
Hier ist der neueste Bericht von der vordersten Front.
$LastChangedDate: 2006-07-22 21:42:37 -0700 (Sat, 22 Jul 2006) $
$Rev$
Cumulus-Wolken entstehen öfter, wenn der Sommer naht.
```

Falls nun jemand anderes eine Änderung an `weather.txt` überträgt, wird Ihre Kopie der Datei den gleichen ersetzten Wert

[8] … oder sogar einen Buchabschnitt …

des Schlüsselwortes anzeigen wie vorher – bis Sie Ihre Arbeitskopie aktualisieren. Zu diesem Zeitpunkt werden die Schlüsselworte in Ihrer Datei `weather.txt` mit Informationen ersetzt, die der letzten bekannten Übertragung dieser Datei entsprechen.

Wo ist $GlobalRev$?

Neue Benutzer sind oft darüber verwirrt, wie das Schlüsselwort `$Rev$` funktioniert. Da das Projektarchiv eine einzelne, global größer werdende Revisionsnummer besitzt, nehmen viele Leute an, dass diese Nummer sich im Wert des Schlüsselwortes `$Rev$` widerspiegelt. Jedoch wird `$Rev$` ersetzt mit der letzten Revision, in der die Datei sich *geändert* hat, nicht die letzte Revision, auf die sie aktualisiert wurde. Dieses Verständnis beseitigt zwar die Verwirrung, jedoch bleibt oft Enttäuschung zurück – wie kommt die globale Revisionsnummer ohne ein Subversion-Schlüsselwort automatisch in Ihre Dateien?

Dies bedarf einer externen Behandlung. Subversion liefert das Werkzeug **svnversion** mit, das allein für diesen Zweck gedacht ist. Es arbeitet sich durch Ihre Arbeitskopie und gibt die Revision(en) aus, die es findet. Sie können dieses Programm zusammen mit etwas zusätzlicher Programmierung drumherum verwenden, um diese Revisions-Information in Ihre Dateien einzufügen. Für weitergehende Informationen zu **svnversion**, siehe „svnversion – Subversion Arbeitskkopie-Versions-Information“.

Sie können Subversion mitteilen, eine feste Länge (die Anzahl verwendeter Bytes) für ein ersetztes Schlüsselwort vorzuhalten. Indem ein doppelter Doppelpunkt (`::`) nach dem Namen des Schlüsselwortes geschrieben wird, dem eine Anzahl von Leerzeichen folgt, definieren Sie diese feste Breite. Wenn Subversion nun dieses Schlüsselwort durch das Schlüsselwort und seinen Wert ersetzt, werden im Wesentlichen nur die Leerzeichen ersetzt, so dass die Gesamtbreite des Schlüsselwort-Felds unverändert bleibt. Falls der ersetzte Wert kürzer als das definierte Feld ist, werden am Ende des ersetzten Feldes zusätzliche Füllzeichen (Leerzeichen) eingefügt; falls er zu lang ist, wird er mit einem speziellen Nummernzeichen (#) unmittelbar vor dem letzten Dollarzeichen abgeschnitten.

Nehmen wir zum Beispiel an, Sie hätten ein Dokument, in dem sich ein tabellarischer Abschnitt mit den Subversion-Schlüsselworten befindet. Mit der originalen Syntax für die Schlüsselwort-Ersetzung von Subversion würde die Datei etwa so aussehen:

```
$Rev$:     Revision der letzten Übertragung
$Author$:  Autor der letzten Übertragung
$Date$:    Datum der letzten Übertragung
```

Zu Beginn sieht das noch hübsch aus. Wenn Sie die Datei dann allerdings (natürlich mit aktivierter Schlüsselwort-Ersetzung) übertragen, sehen Sie:

```
$Rev: 12 $:     Revision der letzten Übertragung
$Author: harry $:  Autor der letzten Übertragung
$Date: 2006-03-15 02:33:03 -0500 (Wed, 15 Mar 2006) $:    Datum der letzten
Übertragung
```

Das Ergebnis ist weniger hübsch. Vielleicht sind Sie versucht, die Datei zu korrigieren, so dass es wieder tabellarisch aussieht. Allerdings hält das nur so lange vor, wie die Werte der Schlüsselworte die gleiche Länge haben. Falls die Revision der letzten Änderung eine weitere Stelle einnimmt (etwa von 99 auf 100) oder eine Person mit einem längeren Benutzernamen die Datei überträgt, sieht alles wieder schief aus. Wenn Sie jedoch Subversion 1.2 oder neuer verwenden, können Sie die neue Schlüsselwort-Syntax mit fester Länge verwenden und vernünftige Feldlängen definieren, etwa:

```
$Rev::              $:  Revision der letzten Übertragung
$Author::           $:  Autor der letzten Übertragung
$Date::             $:  Datum der letzten Übertragung
```

Sie übertragen diese Änderung an Ihrer Datei. Diesmal bemerkt Subversion die neue Schlüsselwort-Syntax mit fester Länge und behält die Breite der Felder bei, die Sie durch die Füllzeichen zwischen den doppelten Doppelpunkten und dem abschließenden Dollarzeichen definiert haben. Nach Ersetzung ist die Breite der Felder völlig unverändert – die kurzen Werte für `Rev` und `Author` sind mit Leerzeichen aufgefüllt und das lange Feld `Date` wird mit einem Nummernzeichen abgeschnitten:

```
$Rev:: 13            $:  Revision der letzten Übertragung
$Author:: harry      $:  Autor der letzten Übertragung
$Date:: 2006-03-15 0#$:  Datum der letzten Übertragung
```

Die Verwendung von Schlüsselworten fester Länge ist besonders praktisch, wenn Ersetzungen in komplexe Dateiformate vorgenommen werden sollen, die ihrerseits Felder fester Länge für Daten verwenden oder deren Größe für ein Datenfeld sich außerhalb der Anwendung nur sehr schwer ändern lässt. Was natürlich binäre Formate angeht, müssen Sie stets große Sorgfalt walten lassen, damit irgendeine Schlüsselwortersetzung, fester Länge oder sonstwie, die Integrität des Formates nicht verletzt. Obwohl das ziemlich einfach klingt, kann das für die meisten heutzutage verwendeten binären Dateiformate eine überraschend schwierige Aufgabe sein, und nichts, was sich Mutlose antun sollten.

Denken Sie daran, dass die Möglichkeit besteht, dass aus Multibyte-Zeichen bestehende Werte korrumpiert werden können, da die Breite eines Schlüsselwort-Feldes in Bytes gemessen wird. Ein Benutzername, der einige Multibyte-UTF-8-Zeichen enthält könnte mitten in einer Zeichenfolge abgeschnitten werden, die eines dieser Zeichen repräsentiert. Auf Byte-Ebene handelt es sich dabei bloß um eine Kürzung, ein UTF-8-Text wird jedoch wahrscheinlich als Zeichenkette mit einem falschen oder missratenen letzten Zeichen wiedergegeben. Es ist denkbar, dass bestimmte Anwendungen beim Versuch, die Datei zu laden, den fehlerhaften UTF-8-Text erkennen, die gesamte Datei als fehlerhaft einstufen und deren Bearbeitung vollständig verweigern. Falls Sie also Schlüsselworte auf eine feste Länge beschränken, sollten Sie eine Größe wählen, die diese Art der Ersetzung berücksichtigt.

Verzeichnis-Teilbäume

Standardmäßig wirken die meisten Funktionen von Subversion rekursiv. Beispielsweise erzeugt **svn checkout** eine Arbeitskopie bestehend aus allen Dateien und Verzeichnissen, die sich im angegebenen Bereich des Projektarchivs befinden, indem es rekursiv durch den Verzeichnisbaum wandert, bis die gesamte Struktur auf Ihre lokale Platte kopiert worden ist. Subversion 1.5 führt eine Funktionalität namens *Verzeichnis-Teilbäume* (oder *flache Checkouts*) ein, die es Ihnen erlaubt, einfach eine Arbeitskopie – oder einen Teil einer Arbeitskopie – flacher als vollständig rekursiv auszuchecken, wobei die Möglichkeit besteht, nachträglich anfangs ignorierte Dateien hereinzuholen.

Nehmen wir beispielsweise an, wir hätten ein Projektarchiv mit einem Baum aus Dateien und Verzeichnissen, die die Namen von Familienmitgliedern samt Haustieren hätten. (Es ist sicherlich ein seltsames Beispiel, aber bleiben Sie dran.) Ein normaler Befehl **svn checkout** würde uns eine Arbeitskopie mit dem gesamten Baum geben:

```
$ svn checkout file:///var/svn/repos mom
A    mom/son
A    mom/son/grandson
A    mom/daughter
A    mom/daughter/granddaughter1
A    mom/daughter/granddaughter1/bunny1.txt
A    mom/daughter/granddaughter1/bunny2.txt
A    mom/daughter/granddaughter2
A    mom/daughter/fishie.txt
A    mom/kitty1.txt
A    mom/doggie1.txt
Ausgecheckt, Revision 1.
$
```

Lassen Sie uns nun denselben Baum noch einmal auschecken; dieses Mal jedoch sagen wir Subversion, dass wir nur das oberste Verzeichnis ohne irgendeines seine Kinder haben möchten:

```
$ svn checkout file:///var/svn/repos mom-empty --depth empty
Ausgecheckt, Revision 1
$
```

Beachten Sie, dass wir unserer ursprünglichen Kommandozeile **svn checkout** eine neue Option `--depth` hinzugefügt haben. Diese Option gibt es für viele Subversion-Unterbefehle und ähnelt den Optionen `--non-recursive` (`-N`) und `--recursive` (`-R`). Tatsächlich ist sie eine Kombination, Verbesserung, Nachfolge und Ablösung der beiden älteren Optionen. Zunächst erweitert sie den Grad der Tiefenangabe für Anwender, indem einige vorher nicht unterstützte (oder nicht konsistent unterstützte) Tiefen hinzugefügt wurden. Hier sind die Werte, die Sie für die Tiefe bei einer gegebenen Subversion-Funktion angeben können:

`--depth empty`
: Nur das unmittelbare Argument einer Funktion verwenden, keine der darin enthaltenen Dateien und Verzeichnisse.

`--depth files`
: Das unmittelbare Argument einer Funktion mitsamt der darin unmittelbar enthaltenen Dateien verwenden.

`--depth immediates`
: Das unmittelbare Argument einer Funktion mitsamt der darin unmittelbar enthaltenen Dateien und Verzeichnisse verwenden. Der Inhalt dieser Verzeichnisse wird nicht berücksichtigt.

`--depth infinity`
: Das unmittelbare Argument einer Funktion mitsamt aller darin rekursiv enthaltenen Dateien und Verzeichnisse verwenden.

Natürlich handelt es sich bei der bloßen Zusammenlegung zweier bestehender Optionen zu einer kaum um eine neue Funktionalität, die es Wert wäre, ihr einen ganzen Abschnitt dieses Buches zu widmen. Erfreulicherweise steckt hier noch mehr drin. Der Begriff der Tiefe erstreckt sich nicht nur auf die Funktionen, die Sie mit Ihrem Client ausführen, sondern auch auf die Beschreibung der *Umgebungstiefe* eines Angehörigen einer Arbeitskopie, nämlich die durch die Arbeitskopie dauerhaft vermerkte Tiefe dieses Objekts. Ihre Hauptstärke ist eben diese Dauerhaftigkeit – die Tatsache, dass sie *anhaftet*. Die Arbeitskopie merkt sich solange die Tiefe, die Sie für jedes in ihr enthaltenes Objekt wählen, bis Sie später diese Tiefenwahl ändern; standardmäßig arbeiten Subversion-Befehle auf allen vorhandenen Angehörigen einer Arbeitskopie, egal welche Tiefe für sie gewählt wurde.

Sie können die zwischengespeicherte Umgebungstiefe einer Arbeitskopie mit dem Befehl **svn info** überprüfen. Falls die Umgebungstiefe einen anderen Wert als die unendliche Rekursion hat, zeigt **svn info** eine Zeile mit dem Wert an:

```
$ svn info mom-immediates | grep "^Depth:"
Tiefe: immediates
$
```

Unsere vorhergehenden Beispiele zeigten Checkouts unendlicher Tiefe (der Standard für **svn checkout**) sowie leerer Tiefe. Lassen Sie uns nun Beispiele für die anderen Tiefenwerte ansehen:

```
$ svn checkout file:///var/svn/repos mom-files --depth files
A    mom-files/kitty1.txt
A    mom-files/doggie1.txt
```

```
Ausgecjheckt. Revision 1.
$ svn checkout file:///var/svn/repos mom-immediates --depth immediates
A    mom-immediates/son
A    mom-immediates/daughter
A    mom-immediates/kitty1.txt
A    mom-immediates/doggie1.txt
Ausgecheckt. Revision 1.
$
```

Wie beschrieben, bedeutet jede dieser Tiefen etwas mehr als nur das Argument, aber dennoch weniger als vollständige Rekursion.

Wir haben hier zwar **svn checkout** als Beispiel genommen, jedoch finden Sie die Option `--depth` auch bei vielen anderen Subversion-Befehlen. Bei diesen anderen Befehlen stellt die Tiefenangabe eine Möglichkeit dar, den Wirkbereich einer Funktion auf eine bestimmte Tiefe zu begrenzen, etwa so, wie sich die älteren Optionen `--non-recursive` (`-N`) und `--recursive` (`-R`) verhalten. Das bedeutet, dass, wenn Sie in einer Arbeitskopie einer bestimmten Tiefe arbeiten und eine Funktion geringerer Tiefe verlangen, sich diese Funktion auf die geringere Tiefe beschränkt. Wir können eigentlich noch allgemeiner werden: Wenn eine Arbeitskopie mit einer beliebigen – sogar gemischten – Umgebungstiefe gegeben ist und ein Subversion-Befehl mit einer gewünschten Wirktiefe aufgerufen wird, so wird der Befehl die Umgebungstiefe der Arbeitskopie-Objekte berücksichtigen, wobei der Wirkbereich auf die gewünschte (oder standardmäßige) Wirktiefe beschränkt ist.

Zusätzlich zur Option `--depth` akzeptieren die Unterbefehle **svn update** und **svn switch** eine zweite Option, die mit Tiefe zu tun hat: `--set-depth`. Mit dieser Option können Sie die anhaftende Tiefe eines Objektes der Arbeitskopie ändern. Sehen Sie, was passiert, wenn wir unseren leeren Checkout nehmen und ihn mit **`svn update --set-depth`** ***`NEW-DEPTH TARGET`*** schrittweise tiefer gehen lassen:

```
$ svn update --set-depth files mom-empty
A    mom-empty/kittie1.txt
A    mom-empty/doggie1.txt
Aktualisiert zu Revision 1.
$ svn update --set-depth immediates mom-empty
A    mom-empty/son
A    mom-empty/daughter
Aktualisiert zu Revision 1.
$ svn update --set-depth infinity mom-empty
A    mom-empty/son/grandson
A    mom-empty/daughter/granddaughter1
A    mom-empty/daughter/granddaughter1/bunny1.txt
A    mom-empty/daughter/granddaughter1/bunny2.txt
A    mom-empty/daughter/granddaughter2
A    mom-empty/daughter/fishie1.txt
Aktualisiert zu Revision 1.
$
```

Während wir schrittweise eine größere Tiefe wählten, lieferte uns das Projektarchiv mehr Teile unseres Baums.

In unserem Beispiel arbeiteten wir nur auf der Wurzel unserer Arbeitskopie und änderten ihre Umgebungstiefe. Wir können aber auch die Umgebungstiefe *irgendeines* Unterverzeichnisses innerhalb der Arbeitskopie unabhängig ändern. Eine sorgfältige Verwendung dieser Fähigkeit erlaubt es uns, bestimmte Bereiche des Dateibaums der Arbeitskopie herauszustellen, während andere Teilbereiche komplett weggelassen werden. Hier ist ein Beispiel, wie wir einen Teilbereich eines Zweigs unseres Familienstammbaums aufbauen, einen anderen Zweig vollständig rekursiv darstellen und andere Teile beschnitten (nicht auf der Festplatte) lassen können.

```
$ rm -rf mom-empty
$ svn checkout file:///var/svn/repos mom-empty --depth empty
Ausgecheckt. Revision 1.
```

```
$ svn update --set-depth empty mom-empty/son
A    mom-empty/son
Aktualisiert zu Revision 1.
$ svn update --set-depth empty mom-empty/daughter
A    mom-empty/daughter
Aktualisiert zu Revision 1.
$ svn update --set-depth infinity mom-empty/daughter/granddaughter1
A    mom-empty/daughter/granddaughter1
A    mom-empty/daughter/granddaughter1/bunny1.txt
A    mom-empty/daughter/granddaughter1/bunny2.txt
Aktualisiert zu Revision 1.
$
```

Glücklicherweise wird Ihre Arbeit in der Arbeitskopie nicht komplizierter, falls Sie dort eine verzwickte Sammlung aus Umgebungstiefen haben. Sie können lokale Änderungen in Ihrer Arbeitskopie immer noch vornehmen, rückgängig machen, anzeigen und übertragen, ohne neue Optionen bei den entsprechenden Unterbefehlen angeben zu müssen (auch nicht `--depth` und `--set-depth`). Sogar **svn update** funktioniert wie dort, wo keine bestimmte Tiefe angegeben worden ist – es aktualisiert die Argumente in der Arbeitskopie, die vorhanden sind, wobei deren anhaftende Tiefen berücksichtigt werden.

An dieser Stelle könnten Sie sich vielleicht fragen: „Was soll das Ganze? Wann brauche ich das?“ Ein Szenario, bei dem diese Funktionalität sehr nützlich ist, steht in Zusammenhang mit einer bestimmten Organisation des Projektarchivs, besonders dann, wenn Sie viele in Beziehung stehende oder gemeinsam abhängige Projekte oder Software-Module als Geschwister an einem einzelnen Ort des Projektarchivs untergebracht haben (`trunk/project1`, `trunk/project2`, `trunk/project3`, usw.). In solchen Szenarios könnte es vorkommen, dass Sie persönlich nur eine handvoll dieser Projekte interessiert – vielleicht ein Hauptprojekt sowie einige andere Module, von denen es abhängt. Sie könnten zwar individuelle Arbeitskopien dieser Teile auschecken, jedoch wären diese Arbeitskopien disjunkt und es könnte umständlich sein, Funktionen gleichzeitig auf einigen oder allen anzuwenden. Die Alternative ist die Verwendung der Funktionalität von Verzeichnis-Teilbäumen, bei dem eine einzelne Arbeitskopie erstellt wird, die nur diejenigen Module enthält, die Sie interessieren. Sie würden das gemeinsame Elternverzeichnis mit einer leeren Tiefe auschecken und anschließend die von Ihnen gewünschten Objekte mit unendlicher Tiefe, wie im vorhergehenden Beispiel gezeigt. Sehen Sie es wie ein Opt-In-System für Angehörige einer Arbeitskopie.

Die ursprüngliche (Subversion 1.5) Implementierung flacher Checkouts war zwar gut, unterstützte aber nicht die Verringerung der Wirktiefe von Objekten der Arbeitskopie. Subversion 1.6 behebt dieses Problem. Wenn Sie beispielsweise in einer Arbeitskopie unendlicher Wirktiefe **`svn update --set-depth empty`** aufrufen, wird das zur Folge haben, dass alles bis auf das oberste Verzeichnis verworfen wird.[9] Subversion 1.6 führt darüber hinaus einen weiteren unterstützten Wert für die Option `--set-depth` ein: `exclude`. Die Verwendung von `--set-depth exclude` mit **svn update** führt dazu, dass das Ziel der Aktualisierung vollständig aus der Arbeitskopie entfernt wird – ein Zielverzeichnis würde nicht einmal leer hinterlassen. Das ist besonders dann hilfreich, wenn es mehr Dinge gibt, die Sie in der Arbeitskopie behalten wollen als Dinge, die Sie *nicht* mehr haben möchten.

Betrachten Sie ein Verzeichnis mit hunderten von Unterverzeichnissen, von denen Sie eins lieber nicht in Ihrer Arbeitskopie hätten. Beim „additiven“ Ansatz mit Teilbäumen könnten Sie das Verzeichnis mit einer leeren Tiefe auschecken und dann explizit (mit **`svn update --set-depth infinity`**) für jedes Unterverzeichnis die Wirktiefe hochsetzen, mit Ausnahme desjenigen, welches Sie nicht haben möchten.

```
$ svn checkout http://svn.example.com/repos/many-dirs --depth empty
…
$ svn update --set-depth infinity many-dirs/wanted-dir-1
…
$ svn update --set-depth infinity many-dirs/wanted-dir-2
…
$ svn update --set-depth infinity many-dirs/wanted-dir-3
…
### usw., usw., ...
```

[9] Natürlich auf eine sichere Art und Weise. Wie in anderen Situationen lässt Subversion Dateien, die Sie bearbeitet haben oder die nicht versioniert sind, auf der Platte.

Das könnte recht lästig werden, insbesonders, da Sie nicht einmal Ansätze dieser Verzeichnisse in Ihrer Arbeitskopie haben, mit denen Sie arbeiten könnten. Eine solche Arbeitskopie hätte auch noch eine weitere Eigenschaft, die Sie nicht erwarten oder wünschen würden: falls jemand anderes neue Unterverzeichnisse in diesem Oberverzeichnis erzeugen würde, bekämen Sie bei der Aktualisierung der Arbeitskopie davon nichts mit.

Mit Subversion 1.6 könnten Sie einen anderen Ansatz wählen. Zunächst würden Sie das Verzeichnis vollständig auschecken. Dann würden Sie **`svn update --set-depth exclude`** für das Unterverzeichnis aufrufen, das Sie nicht haben möchten.

```
$ svn checkout http://svn.example.com/repos/many-dirs
…
$ svn update --set-depth exclude many-dirs/unwanted-dir
D         many-dirs/unwanted-dir
$
```

Dieser Ansatz hinterlässt Ihre Arbeitskopiel mit demselben Inhalt wie der erste Ansatz, jedoch würden beim Aktualisieren auch alle neuen Unterverzeichnisse im obersten Verzeichnis auftauchen. Der Nachteil bei diesem Ansatz ist, dass Sie ein komplettes Unterverzeichnis auschecken müssen, das Sie gar nicht haben wollen, nur damit Sie Subversion sagen können, dass Sie es nicht benötigen. Es kann sogar unmöglich sein, falls das Unterverzeichnis zu groß für Ihre Platte ist (was vielleicht der eigentliche Grund ist, warum Sie es nicht in der Arbeitskopie haben möchten).

Obwohl die Funktion des Ausschließens eines Objektes aus der Arbeitskopie an den Befehl **svn update** gehängt wurde, haben Sie vielleicht bemerkt, dass die Ausgabe von **`svn update --set-depth exclude`** sich von der einer normalen Aktualisierung unterscheidet. Diese Ausgabe verrät die Tatsache, dass unter der Haube der Ausschluss eine vollständig clientseitige Operation ist, also ganz anders als eine typische Aktualisierung.

In einer solchen Situation könnten Sie über einen Kompromissansatz nachdenken. Checken Sie zunächst das oberste Verzeichnis mit `--depth immediates` aus. Schließen Sie dann das Verzeichnis, das Sie nicht benötigen, mit **`svn update --set-depth exclude`** aus. Erweitern Sie schließlich die Wirktiefe für alle verbleibenden Objekte auf unendliche Tiefe, was ziemlich einfach sein sollte, da Sie alle in Ihrer Shell adressierbar sind.

```
$ svn checkout http://svn.example.com/repos/many-dirs --depth immediates
…
$ svn update --set-depth exclude many-dirs/unwanted-dir
D         many-dirs/unwanted-dir
$ svn update --set-depth infinity many-dirs/*
…
$
```

Auch hier wird Ihre Arbeitskopie denselben Inhalt haben wie in den vorhergegangenen beiden Szenarien. Aber nun werden Sie jede neu hinzugefügte Datei oder jedes neu hinzugefügte Verzeichnis beim Aktualisieren – mit leerer Tiefe – mitbekommen. Dann können Sie sich entscheiden, was mit solchen neu erscheinenden Objekten geschehen soll: in unendliche Tiefe expandieren oder vollständig ausschließen.

Sperren

Das Kopieren-Ändern-Zusammenfassen-Modell von Subversion lebt und stirbt mit dessen Zusammenführungs-Algorithmen – besonders dann, wenn es um die Fähigkeit dieser Algorithmen geht, Konflikte aufzulösen, die durch die gleichzeitigen Änderungen mehrerer Benutzer an derselben Datei hervorgerufen worden sind. Subversion bringt von sich aus nur einen derartigen Algorithmus mit: ein Dreiwege-Vergleichs-Algorithmus, der über ausreichend Intelligenz verfügt, um Daten mit der Granularität einer einzelnen Textzeile zu bearbeiten. Subversion erlaubt Ihnen auch, seine Zusammenführungs-Operationen mit externen Werkzeugen zu ergänzen (wie in „Externes diff3“ und „External merge“ beschrieben), von denen manche die Arbeit besser erledigen könnten, indem sie vielleicht die Granularität eines Wortes oder eines einzelnen Buchstaben bieten. Allerdings ist all diesen Algorithmen gemein, dass sie im Allgemeinen nur auf Textdateien arbeiten. Wenn es um

nichttextuelle Dateiformate geht, sieht es ziemlich übel aus. Und falls Sie kein Werkzeug finden, das mit dieser Art von Zusammenführungen zurechtkommt, wirft das Kopieren-Ändern-Zusammenfassen-Modell Probleme für Sie auf.

Betrachten wir einmal ein Beispiel aus dem echten Leben, an dem dieses Modell scheitert. Harry und Sally sind Grafikdesigner und arbeiten am selben Projekt, ein bisschen Werbematerial für einen Automechaniker. Das Design für ein bestimmtes Plakat dreht sich um ein Bild, das ein reparaturbedürftiges Auto zeigt und in einer PNG-Datei abgelegt ist. Der Entwurf für das Plakat ist beinahe fertig, und sowohl Harry als auch Sally sind mit der Wahl des Fotos mit dem beschädigten Auto zufrieden – ein babyblauer 1967er Ford Mustang mit einer bedauerlichen Delle im Kotflügel vorne links.

Nun gibt es eine, im Grafikdesign übliche, Planänderung, was dazu führt, dass es Bedenken hinsichtlich der Farbe des Wagens gibt. Also aktualisiert Sally ihre Arbeitskopie auf `HEAD`, startet ihre Fotobearbeitungs-Software und ändert das Bild, so dass das Auto nun kirschrot ist. Zwischenzeitlich denkt sich Harry, der sich heute besonders inspiriert fühlt, dass die Wirkung des Bildes verstärkt würde, wenn der Wagen so aussähe, als habe er einen heftigeren Aufprall verkraften müssen. Auch er aktualisiert auf `HEAD` und malt ein paar Risse auf die Windschutzscheibe. Er ist vor Sally fertig und überträgt das veränderte Bild, nachdem er die Früchte seines unbestreitbaren Talents bewundert hat. Kurz danach ist Sally mit der neuen Autolackierung fertig und versucht, ihre Änderungen zu übertragen. Aber Subversion lässt, wie erwartet, die Übertragung scheitern und teilt Sally mit, dass ihre Version des Bildes nun veraltet sei.

Hier fangen die Schwierigkeiten an. Falls Harry und Sally Änderungen an einer Textdatei machten, aktualisierte Sally einfach ihre Arbeitskopie und erhielt dabei Harrys Änderungen. Schlimmstenfalls hätten beide denselben Dateiabschnitt verändert, und Sally müsste den Konflikt manuell auflösen. Aber es sind keine Textdateien – es sind binäre Bilder. Während es einfach ist, das erwartete Ergebnis der Zusammenführung der Inhalte zu beschreiben, ist die Wahrscheinlichkeit ziemlich gering, dass es eine Software gibt, die über ausreichend Intelligenz verfügt, das Bild auf dem beide Änderungen basieren, die Änderungen von Harry und die Änderungen von Sally zu untersuchen, um anschließend das Bild eines verbeulten roten Mustangs mit gesprungener Windschutzscheibe auszugeben.

Natürlich wäre es glatter gelaufen, wenn Harry und Sally ihre Änderungen an dem Bild nacheinander gemacht hätten – wenn etwa Harry gewartet hätte und seinen Sprung in der Windschutzscheibe auf Sallys nun roten Wagen gezeichnet hätte, oder wenn Sally die Farbe eines Autos mit bereits gesprungener Windschutzscheibe geändert hätte. Wie in „Die Kopieren–Ändern–Zusammenfassen-Lösung“ erörtert, verschwindeten die meisten dieser Probleme vollständig, wenn Harry und Sallys Kommunikation perfekt wäre.[10] Aus der Tatsache, dass das eigene Versionskontrollsystem eine Kommunikationsform darstellt, folgt, dass es keine schlechte Sache wäre, wenn diese Software die Serialisierung von nicht parallel durchführbaren Änderungen ermöglichte. Hier ist der Zeitpunkt für die Subversion-Implementierung des Sperren-Ändern-Freigeben-Modells gekommen. Hier reden wir nun über das *Sperren* in Subversion, das in etwa den „Reserved Checkouts“ anderer Versionskontrollsysteme entspricht.

Letztendlich existiert der Sperrmechanismus von Subversion, um die Verschwendung von Aufwand und Zeit zu minimieren. Indem einem Benutzer erlaubt wird, programmatisch das Exklusivrecht zum Ändern einer Datei im Projektarchiv in Anspruch zu nehmen, kann dieser Benutzer sich ziemlich sicher sein, dass die von ihm aufgebrachte Energie für nicht zusammenführbare Änderungen nicht verschwendet war – die Übersendung seiner Änderungen wird erfolgreich sein. Da Subversion auch den anderen Benutzern mitteilt, dass für ein bestimmtes versioniertes Objekt die Serialisierung aktiviert wurde, können diese Benutzer vernünftigerweise erwarten, dass dieses Objekt gerade von jemand anderen geändert wird. Auch sie können dann die Verschwendung ihrer Zeit und Energie für nicht zusammenführbare Änderungen vermeiden, die dann schließlich wegen mangelnder Aktualität nicht übertragen werden könnten.

Wenn sich auf den Sperrmechanismus von Subversion bezogen wird, ist eigentlich die Rede von einer ziemlich facettenreichen Ansammlung von Verhaltensweisen, die die Fähigkeit beinhaltet, eine versionierte Datei zu sperren [11] (die exklusive Berechtigung zum Ändern der Datei in Anspruch zu nehmen), die Datei freizugeben (die exklusive Änderungsberechtigung abzugeben), Berichte zu liefern, welche Dateien von wem gesperrt sind, Dateien mit Vermerken zu versehen, die das Sperren vor dem Ändern dringend empfehlen usw. In diesem Abschnitt behandeln wir all diese Facetten des umfangreicheren Sperrmechanismus.

Die drei Bedeutungen von „Sperre“

In diesem Abschnitt, und fast überall in diesem Buch, beschreiben die Wörter „Sperre“ und „sperren“ einen Mechanismus des gegenseitigen Ausschlusses zwischen Benutzern, um kollidierende Übertragungen zu vermeiden. Unglücklicherweise gibt es noch zwei weitere Arten von „Sperre“ mit dem sich manchmal Subversion, und somit auch dieses Buch, befassen muss.

Die zweite Art sind *Arbeitskopie-Sperren*, die intern von Subversion verwendet werden, um Kollisionen zwischen

[10] Kommunikation wäre insofern auch keine schlechte Medizin für die gleichnamigen Harry und Sally aus Hollywood gewesen.
[11] Momentan erlaubt Subversion nicht das Sperren von Verzeichnissen.

mehreren Subversion-Clients zu verhindern, die in derselben Arbeitskopie arbeiten. Diese Art Sperre wird durch ein `L` in der dritten Spalte der Ausgabe von **svn status** angezeigt und durch den Befehl **svn cleanup** entfernt, wie in „Manchmal müssen Sie einfach nur aufräumen“ beschrieben.

Zum Dritten gibt es *Datenbank-Sperren*, die intern vom Berkeley-DB-Backend verwendet werden, um Kollisionen zwischen mehreren Programmen zu verhindern, die auf die Datenbank zugreifen möchten. Dies ist die Sorte von Sperren, deren unerwünschte Persistenz nach einem Fehler dazu führen kann, dass das Projektarchiv sich „verklemmt“, wie in „Wiederherstellung von Berkeley DB“ beschrieben.

Im Allgemeinen können Sie diese anderen Arten von Sperren vergessen bis irgendetwas schief geht, und Sie sich wieder damit beschäftigen müssen. In diesem Buch bedeutet „Sperre“ die erste Art, solange das Gegenteil nicht aus dem Zusammenhang hervorgeht oder ausdrücklich erwähnt wird.

Anlegen von Sperren

Im Projektarchiv von Subversion ist eine *Sperre* ein Metadatum, das einem Benutzer das exklusive Recht zum Ändern einer Datei erteilt. Dieser Benutzer wird *Sperreigner* genannt. Jede Sperre hat auch eine eindeutige Identifikation, üblicherweise eine lange Zeichenkette, *Sperrmarke* genannt. Das Projektarchiv verwaltet Sperren, letztendlich übernimmt es das Anlegen, das Durchsetzten und das Entfernen derselben. Falls irgendeine Übertragungstransaktion versucht, eine gesperrte Datei zu ändern oder zu löschen (oder eins der Elternverzeichnisse der Datei zu löschen), verlangt das Projektarchiv zweierlei Informationen: dass der die Übertragung ausführende Client sich als der Eigner der Sperrmarke authentisiert, und dass die Sperrmarke im Zuge der Übertragung vorgelegt wird, um zu beweisen, dass der Client weiß, welche Sperre verwendet wird.

Um das Anlegen einer Sperre zu demonstrieren, gehen wir zurück zu unserem Beispiel mit mehreren Grafikdesignern, die an derselben binären Bilddatei arbeiten. Harry hat sich entschieden, ein JPEG-Bild zu ändern. Um andere Leute daran zu hindern, Änderungen an der Datei zu übertragen, während er sie ändert (und auch, um ihnen mitzuteilen, dass er gerade Änderungen vornimmt), sperrt er die Datei im Projektarchiv mit dem Befehl **svn lock**.

```
$ svn lock banana.jpg -m "Editing file for tomorrow's release."
»banana.jpg« gesperrt durch »harry«.
$
```

Das vorangegangene Beispiel zeigt eine Menge neuer Dinge. Beachten Sie zunächst, dass Harry die Option `--message` (`-m`) an **svn lock** übergeben hat. Ähnlich wie **svn commit** kann der Befehl **svn lock** Kommentare annehmen, entweder über `--message` (`-m`) oder `--file` (`-F`), um den Grund für die Dateisperre zu beschreiben. Im Gegensatz zu **svn commit** verlangt **svn lock** jedoch nicht nach einer Nachricht, indem es Ihren bevorzugten Texteditor aufruft. Sperrkommentare sind zwar optional, aber zur Unterstützung der Kommunikation empfohlen.

Zum Zweiten war der Sperrversuch erfolgreich. Das bedeutet, dass die Datei noch nicht gesperrt war, und Harry die letzte Version der Datei hatte. Falls Harrys Arbeitskopie der Datei nicht mehr aktuell gewesen wäre, hätte das Projektarchiv die Anfrage abgelehnt und Harry dazu gezwungen, **svn update** aufzurufen und den Sperrbefehl dann erneut zu versuchen. Der Sperrbefehl wäre ebenso fehlgeschlagen, falls die Datei bereits von jemand anderem gesperrt worden wäre.

Wie Sie sehen können, gibt der Befehl **svn lock** eine Bestätigung bei einer erfolgreichen Sperrung aus. An dieser Stelle wird die Tatsache, dass die Datei gesperrt ist, durch die Ausgabe der berichtenden Unterbefehle **svn status** und **svn info** offensichtlich.

```
$ svn status
     K banana.jpg

$ svn info banana.jpg
Pfad: banana.jpg
Name: banana.jpg
URL: http://svn.example.com/repos/project/banana.jpg
Basis des Projektarchivs: http://svn.example.com/repos/project
UUID des Projektarchivs: edb2f264-5ef2-0310-a47a-87b0ce17a8ec
Revision: 2198
```

```
Knotentyp: Datei
Plan: normal
Letzter Autor: frank
Letzte geänderte Rev: 1950
Letztes Änderungsdatum: 2006-03-15 12:43:04 -0600 (Wed, 15 Mar 2006)
Text zuletzt geändert: 2006-06-08 19:23:07 -0500 (Thu, 08 Jun 2006)
Eigenschaften zuletzt geändert: 2006-06-08 19:23:07 -0500 (Thu, 08 Jun 2006)
Prüfsumme: 3b110d3b10638f5d1f4fe0f436a5a2a5
Sperrmarke: opaquelocktoken:0c0f600b-88f9-0310-9e48-355b44d4a58e
Sperreigner: harry
Sperre erzeugt: 2006-06-14 17:20:31 -0500 (Wed, 14 Jun 2006)
Sperrkommentar (1 Zeile):
Editing file for tomorrow's release.

$
```

Aus der Tatsache, dass der Befehl **svn info**, der nicht das Projektarchiv kontaktiert, wenn er mit einem Pfad der Arbeitskopie aufgerufen wird, die Sperrmarke anzeigen kann, enthüllt eine wichtige Information über diese Marken: sie werden in der Arbeitskopie zwischengespeichert. Das Vorhandensein der Sperrmarke ist kritisch. Sie erteilt der Arbeitskopie die Berechtigung, die Sperrmarke später zu verwenden. Darüber hinaus zeigt der Befehl **svn status** ein K neben der Datei an (kurz für locKed, gesperrt), was darauf hinweist, dass die Sperrmarke vorhanden ist.

Über Sperrmarken

Eine Sperrmarke ist keine Anmeldungsmarke, eher eine *Berechtigungsmarke*. Die Marke ist kein geschütztes Geheimnis. Tatsächlich kann die für eine Sperre einzigartige Marke von jedem entdeckt werden, der **`svn info` *`URL`*** aufruft. Eine Sperrmarke wird erst dann etwas besonderes, wenn sie innerhalb einer Arbeitskopie liegt. Sie ist der Beweis dafür, dass die Sperre in dieser bestimmten Arbeitskopie angelegt wurde und nicht irgendwo anders durch irgendeinen anderen Client. Es reicht nicht, sich als Sperreigner zu authentisieren, um Missgeschicke zu verhindern.

Nehmen wir beispielsweise an, Sie sperrten eine Datei im Büro aber machten Feierabend, bevor Sie die Änderungen an dieser Datei fertiggestellt haben. Es sollte nicht möglich sein, später am Abend versehentlich Änderungen an derselben Datei von Ihrem Rechner zu Hause aus zu machen, nur weil Sie sich als Sperreigner authentisiert haben. Mit anderen Worten verhindert die Sperrmarke, dass ein Teil von Subversion-Software die Arbeit eines anderen Teils unterminiert. (In unserem Beispiel hätten Sie eine *Freigabeerzwingung* mit anschließender, erneuter Sperrung durchführen müssen, wenn Sie die Datei wirklich aus einer anderen Arbeitskopie heraus hätten ändern müssen.)

Da Harry nun `banana.jpg` gesperrt hat, kann Sally diese Datei weder ändern noch löschen:

```
$ svn delete banana.jpg
D         banana.jpg
$ svn commit -m "Delete useless file."
Lösche         banana.jpg
svn: Übertragen schlug fehl (Details folgen):
svn: Der Server hat einen unerwarteten Rückgabewert (423 Locked) in Antwort auf die "
 Anfrage DELETE für »/repos/project/!svn/wrk/64bad3a9-96f9-0310-818a-df4224ddc35d/\
banana.jpg« zurückgeliefert"
$
```

Harry aber kann seine Änderungen an der Datei übertragen, nachdem er das Gelb der Banane verbessert hat. Das funktioniert, da er sich als der Sperreigner authentisiert hat, und weil seine Arbeitskopie die korrekte Sperrmarke beinhaltet:

```
$ svn status
M    K banana.jpg
$ svn commit -m "Make banana more yellow"
```

```
Sende          banana.jpg
Übertrage Daten .
Revision 2201 übertragen.
$ svn status
$
```

Beachten Sie, dass nach Abschluss der Übertragung **svn status** anzeigt, dass die Sperrmarke nicht mehr in der Arbeitskopie vorhanden ist. Das ist das Standardverhalten von **svn commit** – es durchsucht die Arbeitskopie (oder die Liste von Zielobjekten, falls angegeben) nach lokalen Änderungen und sendet im Zuge der Übertragungstransaktion alle Sperrmarken denen es begegnet an den Server. Nach dem erfolgreichem Abschluss der Übertragung, sind alle erwähnten Sperren aufgehoben – *sogar für Dateien, die nicht übertragen worden sind*. Das soll Benutzer davon abschrecken, beim Sperren oder beim langen Halten von Sperren schludrig zu sein. Falls Harry wahllos 30 Dateien in einem Verzeichnis namens `images` sperrt, weil er sich nicht sicher ist, welche Dateien er ändern muss, dann aber nur vier dieser Dateien ändert, werden trotzdem alle 30 Sperren freigegeben, wenn er **`svn commit images`** aufruft.

Dieses Verhalten der automatischen Sperrfreigabe kann mit der Option `--no-unlock` von **svn commit** unterbunden werden. Diese Option wird am besten dann verwendet, wenn Sie Änderungen übertragen möchten, aber trotzdem weitere Änderungen planen und die Sperren deshalb beibehalten werden sollen. Sie können das auch zum Standardverhalten machen, indem Sie die Laufzeitoption `no-unlock` setzen (siehe „Laufzeit-Konfigurationsbereich“).

Natürlich wird durch das Sperren einer Datei keine Verpflichtung eingegangen, eine Änderung übertragen zu müssen. Die Sperre kann jederzeit mit einem einfachen **svn unlock** freigegeben werden:

```
$ svn unlock banana.c
»banana.c« freigegeben.
```

Entdecken von Sperren

Falls eine Übertragung aufgrund der Sperre von jemand anderen fehlschlägt, ist es ziemlich einfach, Informationen darüber zu erhalten. Die einfachste Möglichkeit ist, **`svn status --show-updates`** aufzurufen:

```
$ svn status -u
M               23   bar.c
M    O          32   raisin.jpg
       *        72   foo.h
Status bezogen auf Revision:     105
$
```

In diesem Beispiel kann Sally nicht nur sehen, dass ihre Kopie von `foo.h` nicht mehr aktuell ist, sondern auch, dass eine der zwei geänderten Dateien, die sie übertragen wollte, im Projektarchiv gesperrt ist. Das Symbol `O` steht für „Other“, was bedeutet, dass eine Sperre auf der Datei liegt, die von jemand anderen angelegt wurde. Wenn sie eine Übertragung versuchte, würde die Sperre auf `raisin.jpg` das verhindern. Sally fragt sich jetzt nur noch, wer die Sperre wann und warum angelegt hat. Auch hierzu liefert **svn info** die Antwort:

```
$ svn info ^/raisin.jpg
Pfad: raisin.jpg
Name: raisin.jpg
URL: http://svn.example.com/repos/project/raisin.jpg
Basis des Projektarchivs: http://svn.example.com/repos/project
UID des Projektarchivs: edb2f264-5ef2-0310-a47a-87b0ce17a8ec
Revision: 105
Knotentyp: Datei
```

```
Letzter Autor: sally
Letzte geänderte Rev: 32
Letztes Änderungsdatum: 2006-01-25 12:43:04 -0600 (Sun, 25 Jan 2006)
Sperrmarke: opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Sperreigner: harry
Sperre erzeugt: 2006-02-16 13:29:18 -0500 (Thu, 16 Feb 2006)
Sperrkommentar (1 Zeile):
Need to make a quick tweak to this image.
$
```

Ebenso, wie Sie **svn info** zum Untersuchen von Objekten in der Arbeitskopie verwenden können, erlaubt es Ihnen, Objekte im Projektarchiv zu untersuchen. Falls das Hauptargument von **svn info** ein Pfad der Arbeitskopie ist, wird die gesamte zwischengespeicherte Information der Arbeitskopie angezeigt; die Erwähnung irgendeiner Sperre bedeutet, dass die Arbeitskopie eine Sperrmarke hält (falls eine Datei von einem anderen Benutzer oder in einer anderen Arbeitskopie gesperrt ist, zeigt **svn info** mit einem Pfad der Arbeitskopie keinerlei Informationen über Sperren an). Falls das Hauptargument zu **svn info** ein URL ist, spiegelt die Information die letzte Version eines Objektes im Projektarchiv wider, und die Erwähnung einer Sperre beschreibt die aktuelle Sperre auf dem Objekt.

In diesem konkreten Beispiel kann Sally sehen, dass Harry die Datei am 16. Februar gesperrt hat, um „eine schnelle Optimierung“ zu machen. Da es nun Juni ist, vermutet sie, dass er wahrscheinlich die Sperre vergessen hat. Sie könnte Harry anrufen, um sich zu beschweren und ihn aufzufordern, die Datei freizugeben. Sollte er nicht erreichbar sein, könnte sie versuchen, selber die Freigabe zu erzwingen oder einen Administrator darum zu bitten.

Freigabeerzwingung und Stehlen von Sperren

Eine Sperre im Projektarchiv ist nicht heilig – in der Standardkonfiguration von Subversion können Sperren nicht nur durch die Personen freigegeben werden, die sie angelegt haben, sondern durch jeden. Falls jemand anderes als der ursprüngliche Erzeuger der Sperre diese zerstört, nennen wir das *die Freigabe der Sperre erzwingen*.

Vom Platz eines Administrators aus ist es einfach, eine Freigabe zu erzwingen. Die Programme **svnlook** und **svnadmin** können Sperren direkt aus dem Projektarchiv anzeigen sowie entfernen. (Weitere Informationen zu diesen Werkzeugen unter „Der Werkzeugkasten eines Administrators“.)

```
$ svnadmin lslocks /var/svn/repos
Pfad: /project2/images/banana.jpg
UUID Marke: opaquelocktoken:c32b4d88-e8fb-2310-abb3-153ff1236923
Eigentümer: frank
Erstellt: 2006-06-15 13:29:18 -0500 (Thu, 15 Jun 2006)
Läuft ab:
Kommentar (1 Zeile):
Still improving the yellow color.

Pfad: /project/raisin.jpg
UUID Marke: opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Eigentümer: harry
Erstellt: 2006-02-16 13:29:18 -0500 (Thu, 16 Feb 2006)
Läuft ab:
Kommentar (1 Zeile):
Need to make a quick tweak to this image.

$ svnadmin rmlocks /var/svn/repos /project/raisin.jpg
Sperre für »/project/raisin.jpg« entfernt.
$
```

Die interessantere Option ist, es den Benutzern zu erlauben, gegenseitig über das Netz die Freigabe zu erzwingen. Um das zu machen, muss Sally dem Befehl **svn unlock** einfach ein `--force` mitgeben:

```
$ svn status -u
M              23   bar.c
M    O         32   raisin.jpg
       *       72   foo.h
Status bezogen auf Revision:     105
$ svn unlock raisin.jpg
svn: »raisin.jpg« ist in dieser Arbeitskopie nicht gesperrt.
$ svn info raisin.jpg | grep URL
URL: http://svn.example.com/repos/project/raisin.jpg
$ svn unlock http://svn.example.com/repos/project/raisin.jpg
svn: Sperrfreigabe gescheitert: 403 Forbidden (http://svn.example.com)
$ svn unlock --force http://svn.example.com/repos/project/raisin.jpg
»raisin.jpg« freigegeben.
$
```

Sallys erster Versuch, die Sperre aufzuheben, schlug fehl, da sie **svn unlock** direkt in ihrer Arbeitskopie ausführte und keine Sperrmarke vorhanden war. Um die Sperre direkt aus dem Projektarchiv zu entfernen, muss sie **svn unlock** einen URL übergeben. Ihr erster Versuch, den URL zu entsperren, schlägt fehl, da sie sich nicht als der Sperreigner authentisieren kann (sie hat ja auch nicht die Sperrmarke). Wenn sie jedoch `--force` übergibt, werden die Authentisierungs- und Berechtigungsanforderungen ignoriert und die entfernte Freigabe wird erzwungen.

Es kann sein, dass das einfache Erzwingen einer Freigabe nicht ausreicht. Im aktuellen Beispiel könnte Sally nicht nur beabsichtigt haben, Harrys längst vergessene Sperre zu beseitigen, sondern die Datei für sich zu sperren. Sie kann das erreichen, indem sie **svn unlock** mit `--force` aufruft und direkt anschließend **svn lock**; hier besteht jedoch eine geringe Wahrscheinlichkeit, dass jemand anderes die Datei zwischen den beiden Befehlen sperren könnte. Einfacher ist es, die Sperre zu *stehlen*, was bedeutet, die Datei in einem atomaren Schritt freizugeben und wieder zu sperren. Um das zu tun, übergibt Sally die Option `--force` an **svn lock**:

```
$ svn lock raisin.jpg
svn: Sperranforderung gescheitert: 423 Locked (http://svn.example.com)
$ svn lock --force raisin.jpg
»raisin.jpg« gesperrt durch »sally«.
$
```

Auf jeden Fall wird Harry eine Überraschung erleben, egal, ob die Freigabe erzwungen oder die Sperre gestohlen wurde. Die Arbeitskopie von Harry beinhaltet immer noch die ursprüngliche Sperrmarke, die dazugehörige Sperre gibt es jedoch nicht mehr. Die Sperrmarke wird als *erloschen* bezeichnet. Die durch die Sperrmarke repräsentierte Sperre wurde entweder durch eine Freigabeerzwingung zerstört (sie befindet sich nicht mehr im Projektarchiv) oder gestohlen (durch eine andere Sperre ersetzt). Egal wie, Harry kann es sehen, wenn er **svn status** auffordert, Kontakt zum Projektarchiv aufzunehmen:

```
$ svn status
     K  raisin.jpg
$ svn status -u
     B          32   raisin.jpg
Status bezogen auf Revision:     105
$ svn update
  B  raisin.jpg
Aktualisiert zu Revision 105.
$ svn status
$
```

Falls die Freigabe erzwungen wurde, zeigt **`svn status --show-updates`** (`-u`) ein `B`-Symbol (Broken) neben der Datei an. Falls es eine neue Sperre an Stelle der alten gibt, wird ein `T`-Symbol (gesTohlen) angezeigt. Zu guter Letzt entdeckt **svn update** irgendwelche erloschenen Sperrmarken und entfernt sie aus der Arbeitskopie.

Richtlinien für Sperren

Verschiedene Systeme haben unterschiedliche Auffassungen, was die Strenge von Sperren angeht. Manche Leute sind der Ansicht, dass Sperren in jedem Fall streng durchgesetzt werden müssen und nur vom ursprünglichen Erzeuger oder einem Administrator freigegeben werden sollen. Sie meinen, dass sich Chaos verbreitet und der eigentliche Zweck von Sperren vereitelt wird, falls jeder eine Freigabe erzwingen kann. Die andere Seite ist der Meinung, dass Sperren in erster Linie ein Kommunikationswerkzeug sind. Wenn Benutzer sich gegenseitig die Sperren entziehen, repräsentiert das einen kulturellen Fehler im Team, so dass das Problem nicht durch Software gelöst werden kann.

Subversion geht standardmäßig den „weicheren“ Weg, erlaubt es jedoch Administratoren, mittels Hook-Skripten strengere Richtlinien durchzusetzen. Besonders die `pre-lock`- und `pre-unlock`-Aktionen gestatten Administratoren, zu entscheiden, wann das Erzeugen und Freigeben von Sperren erlaubt sein sollen. Abhängig davon, ob eine Sperre bereits besteht, können diese beiden Aktionen entscheiden, ob ein bestimmter Benutzer eine Freigabe erzwingen oder eine Sperre stehlen darf. Die `post-lock` und `post-unlock` Aktionen sind ebenfalls verfügbar und können verwendet werden, um nach Sperraktionen E-Mails zu versenden. Um mehr über Projektarchiv-Aktionen zu erfahren, siehe „Erstellen von Projektarchiv-Hooks“.

Kommunikation über Sperren

Wir haben uns angesehen, wie **svn lock** und **svn unlock** verwendet werden können, um Sperren anzulegen, freizugeben und deren Freigabe zu erzwingen. Dies erfüllt den Zweck, den Zugriff zum Übergeben von Änderungen an einer Datei zu serialisieren. Aber wie sieht es mit dem größeren Problem aus, Zeitverschwendung zu vermeiden?

Nehmen wir beispielsweise an, dass Harry eine Bilddatei sperrt und mit deren Bearbeitung beginnt. Mittlerweile, weit entfernt, möchte Sally das Gleiche machen. Sie denkt nicht daran, **`svn status --show-updates`** aufzurufen, so dass sie nicht mitbekommt, dass Harry die Datei bereits gesperrt hat. Sie verbringt Stunden mit der Bearbeitung der Datei, und beim Versuch, ihre Änderungen zu übergeben, stellt sie fest, dass die Datei entweder gesperrt oder nicht mehr aktuell ist. Wie auch immer – ihre Änderungen lassen sich nicht mit denen Harrys zusammenführen. Eine Person von beiden muss ihre Arbeit wegwerfen, und eine Menge Zeit ist verschwendet worden.

Die Lösung von Subversion für dieses Problem besteht in einem Mechanismus, der Benutzer daran erinnert, dass eine Datei *vor* dem Ändern gesperrt werden sollte. Es handelt sich um eine besondere Eigenschaft: `svn:needs-lock`. Wenn diese Eigenschaft einer Datei zugeordnet ist (egal mit welchem Wert), versucht Subversion mit Dateisystem-Zugriffsrechten, die Datei nur lesbar zu machen – natürlich nur dann, wenn der Benutzer die Datei nicht ausdrücklich gesperrt hat. Wenn eine Sperrmarke vorhanden ist (als ein Ergebnis eines Aufrufs von **svn lock**), wird die Datei schreib- und lesbar. Wird die Sperre freigegeben, wird die Datei wieder nur lesbar.

Die Theorie ist die, dass Sally sofort merkt, dass irgend etwas nicht stimmt, wenn sie die mit dieser Eigenschaft versehene Bilddatei zum Ändern öffnet: Viele Anwendungen benachrichtigen Benutzer sofort, wenn eine nur lesbare Datei zum Ändern geöffnet werden soll, und fast alle verhindern es, dass Änderungen an der Datei gespeichert werden. Das erinnert sie daran, die Datei vor dem Ändern zu sperren, wobei sie die bereits bestehende Sperre entdeckt:

```
$ /usr/local/bin/gimp raisin.jpg
gimp: error: file is read-only!
$ ls -l raisin.jpg
-r--r--r--   1 sally   sally   215589 Jun  8 19:23 raisin.jpg
$ svn lock raisin.jpg
svn: "Sperranforderung gescheitert: 423 Locked (http://svn.example.com)
$ svn info http://svn.example.com/repos/project/raisin.jpg | grep Lock
Sperrmarke: opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Sperreigner: harry
Sperre erzeugt: 2006-06-08 07:29:18 -0500 (Thu, 08 June 2006)
Sperrkommentar (1 Zeile):
Making some tweaks.  Locking for the next two hours.
$
```

Es sei Benutzern und Administratoren gleichermaßen empfohlen, die Eigenschaft `svn:needs-lock` an allen Dateien anzubringen, die nicht kontextabhängig zusammengeführt werden können. Dies ist die wichtigste Technik, um ein gutes Sperrverhalten zu bestärken und vergeudeten Aufwand zu vermeiden.

Beachten Sie, dass diese Eigenschaft ein Kommunikationswerkzeug darstellt, welches unabhängig vom Sperrsystem funktioniert. Mit anderen Worten: Jede Datei kann gesperrt werden, egal, ob diese Eigenschaft vorhanden ist oder nicht. Und andersherum bedeutet das Vorhandensein dieser Eigenschaft nicht, dass das Projektarchiv bei der Übergabe eine Sperre erforderlich macht.

Leider ist dieses System nicht unfehlbar. Es ist möglich, dass die Nur-Lesbar-Erinnerung nicht immer funktioniert, auch wenn eine Datei diese Eigenschaft besitzt. Manchmal benehmen sich Anwendungen daneben und „kapern“ die nur lesbare Datei, indem sie dem Benutzer trotzdem stillschweigend das Ändern und Sichern gestatten. In dieser Situation kann Subversion nicht viel machen – letztendlich gibt es einfach keinen Ersatz für gute zwischenmenschliche Kommunikation.[12]

Externals-Definitionen

Manchmal kann es nützlich sein, eine Arbeitskopie anzulegen, die aus einer Anzahl verschiedener Checkouts besteht. Es könnte beispielsweise sein, dass Sie verschiedene Unterverzeichnisse aus verschiedenen Bereichen des Projektarchivs haben möchten oder vielleicht sogar aus völlig verschiedenen Projektarchiven. Natürlich könnten Sie ein solches Szenario manuell erstellen – indem Sie **svn checkout** verwenden, um die verschachtelte Verzeichnisstruktur Ihrer Wahl anzulegen. Wenn diese Struktur jedoch für jeden Benutzer Ihres Projektarchivs wichtig ist, müsste jeder andere Benutzer dieselben Checkouts ausführen wie Sie.

Glücklicherweise unterstützt Subversion *Externals-Definitionen*. Eine Externals-Definition ist eine Abbildung eines lokalen Verzeichnisses auf den URL – und idealerweise eine bestimmte Revision – eines versionierten Verzeichnisses. In Subversion deklarieren Sie Externals-Definitionen in Gruppen unter Verwendung der Eigenschaft `svn:externals`. Sie können diese Eigenschaft mit **svn propset** oder **svn propedit** erstellen oder ändern (siehe „Ändern von Eigenschaften“). Sie kann jedem versionierten Verzeichnis zugeordnet werden, und ihr Wert beschreibt sowohl die Adresse des externen Projektarchivs als auch das Verzeichnis auf dem Client, in dem diese Adresse ausgecheckt werden soll.

Sobald sie mit einem versionierten Verzeichnis verknüpft ist, bietet die Eigenschaft `svn:externals` den Komfort, dass jeder, der eine Arbeitskopie mit diesem Verzeichnis auscheckt, die Vorteile der Externals-Definition mitbekommt. Mit anderen Worten: Sobald sich jemand die Mühe gemacht hat, die verschachtelte Struktur der Arbeitskopie zu definieren, braucht sich niemand mehr darum zu kümmern – Subversion checkt nach der ursprünglichen Arbeitskopie automatisch die externen Arbeitskopien aus.

Die relativen Zielverzeichnisse von Externals-Definitionen *dürfen nicht* bereits auf Ihrem oder dem Systemen anderer Benutzer vorhanden sein – Subversion erzeugt sie beim Auschecken der externen Arbeitskopie.

Mit dem Design der Externals-Definition bekommen Sie auch alle normalen Vorteile der Subversion-Eigenschaften. Die Definitionen sind versioniert. Falls Sie eine Externals-Definition ändern müssen, können Sie das mit den üblichen Unterbefehlen zum Ändern von Eigenschaften bewerkstelligen. Wenn Sie eine Änderung an der Eigenschaft `svn:externals` übergeben, wird Subversion beim nächsten **`svn update`** die ausgecheckten Elemente mit der geänderten Externals-Definition synchronisieren. Dasselbe passiert, wenn andere ihre Arbeitskopie aktualisieren und Ihre Änderungen an der Externals-Definition erhalten.

Da die Eigenschaft `svn:externals` einen mehrzeiligen Wert besitzt, empfehlen wir dringend, dass Sie **svn propedit** statt **svn propset** verwenden.

Subversion-Versionen vor 1.5 akzeptieren ein Format für Externals-Definitionen, welches eine mehrzeilige Tabelle ist, die aus Unterverzeichnissen (relativ zum versionierten Verzeichnis, mit dem die Eigenschaft verknüpft ist), optionalen Revisions-Flags und vollqualifizierten, absoluten Subversion-Projektarchiv-URLs besteht. Ein Beispiel könnte so aussehen:

[12] Vielleicht mit Ausnahme einer klassischen vulkanischen Gedankenvereinigung.

```
$ svn propget svn:externals calc
third-party/sounds             http://svn.example.com/repos/sounds
third-party/skins -r148        http://svn.example.com/skinproj
third-party/skins/toolkit -r21 http://svn.example.com/skin-maker
```

Wenn jemand eine Arbeitskopie des Verzeichnisses `calc` aus dem obigen Beispiel auscheckt, fährt Subversion damit fort, die Objekte in der Externals-Definition auszuchecken.

```
$ svn checkout http://svn.example.com/repos/calc
A    calc
A    calc/Makefile
A    calc/integer.c
A    calc/button.c
Ausgecheckt. Revision 148.

Hole externen Verweis nach calc/third-party/sounds
A    calc/third-party/sounds/ding.ogg
A    calc/third-party/sounds/dong.ogg
A    calc/third-party/sounds/clang.ogg
…
A    calc/third-party/sounds/bang.ogg
A    calc/third-party/sounds/twang.ogg
Ausgecheckt. Revision 14.

Hole externen Verweis nach calc/third-party/skins
…
```

Seit Subversion 1.5 wird jedoch ein neues Format der Eigenschaft `svn:externals` unterstützt. Die Externals-Definitionen sind immer noch mehrzeilig, jedoch hat sich die Reihenfolge und das Format der verschiedenen Informationen geändert. Die neue Syntax lehnt sich nun mehr an die Reihenfolge der Argumente an, die Sie **svn checkout** übergeben: Zunächst kommen die Revisions-Flags, dann der URL des externen Subversion-Projektarchivs und schließlich das relative lokale Unterverzeichnis. Beachten Sie jedoch, dass wir diesmal nicht von „vollqualifizierten, absoluten Subversion-Projektarchiv-URLs“ gesprochen haben. Das liegt daran, dass das neue Format relative URLs und URLs mit Peg-Revisionen unterstützt. Das vorausgegangene Beispiel einer Externals-Definition könnte in Subversion 1.5 so aussehen:

```
$ svn propget svn:externals calc
      http://svn.example.com/repos/sounds third-party/sounds
-r148 http://svn.example.com/skinproj third-party/skins
-r21  http://svn.example.com/skin-maker third-party/skins/toolkit
```

Bei Verwendung der Syntax für Peg-Revisionen (die wir detailliert in „Peg- und operative Revisionen“ behandeln), könnte es so aussehen:

```
$ svn propget svn:externals calc
http://svn.example.com/repos/sounds third-party/sounds
http://svn.example.com/skinproj@148 third-party/skins
http://svn.example.com/skin-maker@21 third-party/skins/toolkit
```

Sie sollten ernsthaft erwägen, explizite Revisionsnummern in all Ihren Externals-Definitionen zu verwenden. Wenn Sie das tun, bedeutet dies, dass Sie entscheiden, wann ein anderer Schnappschuss mit externen

> Informationen herangezogen werden soll und welcher Schnappschuss genau. Außer der Vermeidung überraschender Änderungen an Projektarchiven Dritter, auf die Sie keinen Einfluss haben, bedeuten explizite Revisionsnummern, dass beim Zurücksetzen Ihrer Arbeitskopie auf eine frühere Revision auch die Externals-Definitionen auf den entsprechenden früheren Stand zurückgesetzt werden, was wiederum bedeutet, dass die externen Arbeitskopien derart aktualisiert werden, dass *sie* so aussehen wie zum Zeitpunkt Ihres Projektarchivs der früheren Revision. Für Software-Projekte kann das den Unterschied zwischen einem erfolgreichen und einem gescheiterten Build eines älteren Schnappschusses Ihrer komplexen Codebasis ausmachen.

Bei den meisten Projektarchiven bewirken die drei Formate der Externals-Definition letztendlich dasselbe. Alle bringen die gleichen Vorteile. Leider bringen alle aber auch die gleichen Ärgernisse. Da die Definitionen absolute URLs verwenden, hat das Kopieren oder Verschieben eines damit verknüpften Verzeichnisses keine Auswirkungen auf das, was als extern ausgescheckt wird (obwohl natürlich das relative lokale Zielverzeichnis mit dem umbenannten Verzeichnis mitwandert). Das kann in bestimmten Situationen verwirrend – sogar frustrierend – sein. Nehmen wir beispielsweise an, dass Sie ganz oben ein Verzeichnis namens `my-project` haben und eine Externals-Definition auf eins seiner Unterverzeichnisse erstellt haben (`my-project/some-dir`), welches wiederum die letzte Revision eines anderen Unterverzeichnisses verfolgt (`my-project/external-dir`).

```
$ svn checkout http://svn.example.com/projects .
A    my-project
A    my-project/some-dir
A    my-project/external-dir
…
Hole externen Verweis nach »my-project/some-dir/subdir«
Externer Verweis ausgecheckt, Revision 11.

Ausgecheckt, Revision 11.
$ svn propget svn:externals my-project/some-dir
subdir http://svn.example.com/projects/my-project/external-dir

$
```

Nun benennen Sie mit **svn move** das Verzeichnis `my-project` um. Zu diesem Zeitpunkt verweist Ihre Externals-Definition noch immer auf einen Pfad unterhalb des Verzeichnisses `my-project`, obwohl das Verzeichnis nicht mehr existiert.

```
$ svn move -q my-project renamed-project
$ svn commit -m "Rename my-project to renamed-project."
Lösche         my-project
Hinzufügen     renamed-project

Revision 12 übertragen.
$ svn update

Hole externen Verweis nach »renamed-project/some-dir/subdir«
svn: Zielpfad existiert nicht
$
```

Absolute URLs können außerdem Probleme bei Projektarchiven hervorrufen, die über verschiedene URL-Schemata verfügbar sind. Falls Ihr Subversion-Server so konfiguriert sein sollte, dass jeder das Projektarchiv über `http://` oder `https://` auschecken darf, Übertragungen jedoch nur über `https://` erlaubt sind, haben Sie ein interessantes Problem. Wenn Ihre Externals-Definitionen die `http://`-Form der Projektarchiv-URLs verwenden, werden Sie nicht im Stande sein, irgend etwas aus den mit diesen Externals erzeugten Arbeitskopien zu übertragen. Wenn Sie andererseits die `https://`-Form der URLs verwenden, kann jemand, der mit `http://` auscheckt, da sein Client `https://` nicht unterstützt, die externen Verweise nicht heranziehen. Beachten Sie ferner, dass beim Umhängen Ihrer Arbeitskopie (mit **svn switch** und der Option `--relocate`) die Externals-Definitionen *nicht* automatisch umgehängt werden.

Subversion 1.5 unternimmt einen riesigen Schritt, um diese Frustrationen zu lindern. Wie bereits früher erwähnt wurde,

können die URLs im neuen Externals-Format relativ sein, und Subversion stellt eine besondere Syntax zur Verfügung, um verschiedene Arten relativer URLs darzustellen.

`../`
: Relativ zum URL des Verzeichnisses, an dem die Eigenschaft `svn:externals` gesetzt ist

`^/`
: Relativ zur Wurzel des Projektarchivs, in dem die Eigenschaft `svn:externals` versioniert ist

`//`
: Relativ zum Schema des URL des Verzeichnisses, an dem die Eigenschaft `svn:externals` gesetzt ist

`/`
: Relativ zum Wurzel-URL des Servers, auf dem die Eigenschaft `svn:externals` versioniert ist

Wenn wir uns nun ein viertes Mal das vorangegangene Beispiel mit der Externals-Definition ansehen und von der neuen absoluten URL-Syntax auf verschiedene Weise Gebrauch machen, könnten wir nun das sehen:

```
$ svn propget svn:externals calc
^/sounds third-party/sounds
/skinproj@148 third-party/skins
//svn.example.com/skin-maker@21 third-party/skins/toolkit
$
```

Subversion 1.6 bringt zwei weitere Verbesserungen für Externals-Definitionen. Zunächst erweitert es die Syntax um einen Zitier- und Maskierungsmechanismus, so dass der Pfad der externen Arbeitskopie Leerzeichen enthalten darf. Das war vorher natürlich problematisch, da Leerzeichen zum Begrenzen von Feldern einer externals Definition verwendet wurden. Nun müssen Sie eine solche Pfad-Spezifizierung lediglich in doppelte Anführungszeichen (") einpacken oder die problematischen Zeichen im Pfad mit einem rückwärtigen Schrägstrich (\) maskieren. Wenn Sie Leerzeichen im *URL*-Teil der Externals-Definition haben, sollten Sie dafür natürlich den Standard-URI-Encoding-Mechanismus verwenden.

```
$ svn propget svn:externals paint
http://svn.thirdparty.com/repos/My%20Project "My Project"
http://svn.thirdparty.com/repos/%22Quotes%20Too%22 \"Quotes\ Too\"
$
```

Subversion 1.6 führt ebenfalls die Unterstützung von Externals-Definitionen für Dateien ein. *File Externals* werden wie Externals für Verzeichnisse konfiguriert und erscheinen in der Arbeitskopie als versionierte Datei.

Nehmen wir beispielsweise an, die Datei `/trunk/bikeshed/blue.html` sei in Ihrem Projektarchiv vorhanden, und Sie möchten diese Datei gerne wie in Revision 40 in Ihrer Arbeitskopie von `/trunk/www/` unter dem Namen `green.html` haben.

Die hierfür benötigte Externals-Definition sollte nun vertraut aussehen:

```
$ svn propget svn:externals www/
^/trunk/bikeshed/blue.html@40 green.html
$ svn update
Hole externen Verweis nach »www«
E    www/green.html
Externer Verweis aktualisiert zu Revision 40.

Aktualisiert zu Revision 103.
$ svn status
```

```
    X   www/green.html
$
```

Wie Sie in der vorangegangenen Ausgabe sehen können, markiert Subversion Datei-Externals mit dem Buchstaben E wenn sie in die Arbeitskopie geholt werden und mit dem Buchstaben X wenn der Zustand der Arbeitskopie angezeigt wird.

Während Verzeichnis-Externals das externe Verzeichnis in eine beliebige Tiefe setzen können und sämtliche fehlende Zwischenverzeichnisse erstellt werden, müssen Datei-Externals in die bereits ausgecheckte Arbeitskopie gesetzt werden.

Wenn Sie das Datei-External mit **svn info** untersuchen, können Sie den URL und die Revision der Herkunft des Externals sehen.

```
$ svn info www/green.html
Path: www/green.html
Name: green.html
URL: http://svn.example.com/projects/my-project/trunk/bikeshed/blue.html
Repository Root: http://svn.example.com/projects/my-project
Repository UUID: b2a368dc-7564-11de-bb2b-113435390e17
Revision: 40
Node kind: file
Schedule: normal
Last Changed Author: harry
Last Changed Rev: 40
Last Changed Date: 2009-07-20 20:38:20 +0100 (Mon, 20 Jul 2009)
Text Last Updated: 2009-07-20 23:22:36 +0100 (Mon, 20 Jul 2009)
Checksum: 01a58b04617b92492d99662c3837b33b
$
```

Da Datei-Externals in der Arbeitskopie als versionierte Dateien erscheinen, können sie bearbeitet und sogar übergeben werden, falls sie auf eine Datei in der HEAD-Revision verweisen. Die übergebenen Änderungen erscheinen dann sowohl im External als auch in der Datei, auf die das External verweist. In unserem Beispiel allerdings verwies das External auf eine ältere Revision, so dass der Versuch scheitert, das External zu übergeben:

```
$ svn status
M   X   www/green.html
$ svn commit -m "change the color" www/green.html
Sende          www/green.html
svn: Übertragen schlug fehl (Details folgen)::
svn: Datei »/trunk/bikeshed/blue.html« ist veraltet
$
```

Denken Sie daran, wenn Sie Datei-Externals definieren. Falls Sie wünschen, dass das External auf eine bestimmte Revision einer Datei verweist, werden Sie das External nicht modifizieren können. Wenn Sie in der Lage sein wollen, das External zu bearbeiten, können Sie keine andere Revision außer HEAD angeben, die auch implizit gesetzt wird, wenn Sie keine Revision angeben.

Leider bleibt die Unterstützung für Externals-Definitionen in Subversions alles andere als ideal. Sowohl Datei- als auch Verzeichnis-Externals haben Schwächen. Für beide Arten von Externals darf der lokale Unterverzeichnis-Teil keine `..`-Verweise auf Elternverzeichnisse enthalten (etwa `../../skins/myskin`). Datei-Externals können nicht auf Dateien aus anderen Projektarchiven verweisen. Der URL eines Datei-Externals muss stets im selben Projektarchiv liegen, wie der URL, in den das Datei-External eingefügt wird. Außerdem können Datei-Externals weder verschoben noch gelöscht werden. Stattdessen muss die Eigenschaft `svn:externals` geändert werden. Allerdings können Datei-Externals kopiert werden.

Am enttäuschendsten ist vielleicht, dass die Arbeitskopieen, die über die Unterstützung von Externals-Definitionen angelegt wurden, immer noch nicht mit der primäten Arbeitskopie verbunden sind (an deren versionierten Verzeichnissen die Eigenschaft `svn:externals` tatsächlich gesetzt wurde). Und Subversion arbeitet immer noch wirklich nur auf disjunkten Arbeitskopien. Wenn Sie also beispielsweise Änderungen übergeben möchten, die Sie in einer oder mehreren dieser externen Arbeitskopien vorgenommen haben, müssen Sie explizit **svn commit** auf diesen Arbeitskopien aufrufen – die Übergabe in der primären Arbeitskopie wird sich nicht rekursiv in externe fortpflanzen.

Wir haben bereits einige der Mängel des alten Formats von `svn:externals` und die Verbesserungen durch das neue Format von Subversion 1.5 erwähnt. Seien Sie jedoch vorsichtig, dass Sie bei Verwendung des neuen Formats nicht versehentlich neue Probleme verursachen. Während beispielsweise die neuesten Clients weiterhin das ursprüngliche Format der Externals-Definitionen verstehen und unterstützen, sind vor-1.5 Clients *nicht* in der Lage, das neue Format korrekt zu verarbeiten. Falls Sie alle Ihre Externals-Definitionen in das neue Format ändern, zwingen Sie effektiv jeden, der diese Externals verwendet, Ihre Subversion-Clients auf eine Version zu bringen, die dieses Format versteht. Vermeiden Sie auch, den `-r`*`NNN`*-Teil der Definition naiverweise umzuschreiben – das ältere Format verwendet diese Revision als eine Peg-Revision, wohingegen das neuere Format sie als eine operativer Revision verwendet (mit einer Peg-Revision von `HEAD`, wenn nicht anders angegeben; siehe „Peg- und operative Revisionen“ für eine vollständige Erklärung der hiesigen Unterscheidung).

Externe Arbeitskopien sind immer noch vollständig unabhängige Arbeitskopien. Sie können direkt auf ihnen arbeiten wie in jeder anderen Arbeitskopie. Das kann sehr praktisch sein, da es Ihnen ermöglicht, eine externe Arbeitskopie zu untersuchen, unabhängig von irgendeiner primären Arbeitskopie, deren `svn:externals`-Eigenschaft ihre Instantiierung veranlasste. Seien Sie trotzdem vorsichtig, damit Sie nicht versehentlich Ihre externe Arbeitskopie auf raffinierte Art modifizieren, so dass Probleme entstehen. Wenn beispielsweise eine Externals-Definition spezifiziert, dass eine externe Arbeitskopie in einer bestimmtern Revision vorgehalten werden soll, und Sie **svn update** direkt auf der externen Arbeitskopie aufrufen, wird Subversion Ihnen gehorchen und Ihre externe Arbeitskopie ist nicht mehr synchron mit ihrer Deklaration in der primären Arbeitskopie. Die Verwendung von **svn switch**, um direkt die externe Arbeitskopie (oder Teile davon) auf einen anderen URL zu wechseln, kann ähnliche Probleme verursachen, falls die Inhalte der primären Arbeitskopie bestimmte externe Inhalte voraussetzen.

Neben den Befehlen **svn checkout**, **svn update**, **svn switch** und **svn export**, welche die *disjunkten* (oder unzusammenhängenden) Unterverzeichnisse mit den ausgecheckten Externals eigentlich verwalten, berücksichtigt auch der Befehl **svn status** Externals-Definitionen. Er zeigt für die disjunkten externen Unterverzeichnisse einen Zustandscode `X` an und durchläuft dann diese Verzeichnisse, um den Zustand der eigentlichen externen Objekte anzuzeigen. Sie können jedem dieser Unterbefehle die Option `--ignore-externals` mitgeben, um die Bearbeitung der Externals-Definitionen zu unterbinden.

Änderungslisten

Für einen Entwickler ist es üblich, zu einem gegebenen Zeitpunkt an mehreren unterschiedlichen, individuellen Änderungen an Teilen des Source-Codes zu arbeiten. Das liegt nicht notwendigerweise an schlechter Planung oder einer Art digitalen Masochismus. Ein Software-Engineer entdeckt oft Fehler am Rande während er an einem Teil des Codes in der Nähe arbeitet. Vielleicht ist er auch halbwegs mit einer großen Änderung fertig, wenn er feststellt, dass die von ihm übergebene Lösung besser als eine Sammlung kleinerer logischer Einheiten übergeben werden sollte. Oft befinden sich diese logischen Einheiten nicht in einem Modul, das sicher von anderen Änderungen getrennt ist. Die Einheiten könnten sich überlappen, mehrere unterschiedliche Dateien im gleichen Modul betreffen oder sogar verschiedene Zeilen in der selben Datei.

Entwickler können verschiedene Arbeitsweisen anwenden, um diese logischen Änderungen zu organisieren. Manche verwenden getrennte Arbeitskopien des selben Projektarchivs, um jede einzelne Änderung voranzutreiben. Andere wiederum wählen kurzlebige Arbeitszweige im Projektarchiv und verwenden eine einzelne Arbeitskopie, die ständig zwischen solchen Zweigen hin- und her geschaltet wird. Eine weitere Gruppe verwendet **diff**- und **patch**-Werkzeuge, um noch nicht übergebene Änderungen in entsprechenden Patch-Dateien zu sichern und wiederherzustellen Jede dieser Methoden hat ihre Vor- und Nachteile, und zu einem großen Teil beeinflussen die Details der vorzunehmenden Änderungen die Methodik, sie auseinander zu halten.

Subversion bietet das Leistungsmerkmal *Änderungslisten* mit, die dieser Mischung eine weitere Methode hinzufügen. Im Grunde sind Änderungslisten beliebige Label (momentan höchstens eins pro Datei), die ausschließlich zum Zweck der Zusammenfassung mehrerer Dateien auf Dateien der Arbeitskopie vergeben werden. Benutzer vieler Software-Angebote von Google kennen dieses Konzept bereits. Gmail [http://mail.google.com/] beispielsweise verfügt nicht über den traditionellen, ordnerbasierten Ansatz der Gliederung von E-Mail. In Gmail vergeben Sie beliebige Labels auf E-Mails, und mehrere E-Mails gelten zu einer Gruppe gehörend, falls sie gemeinsam ein bestimmtes Label haben. Das Ansehen nur einer Gruppe ähnlich

gelabelter E-Mails wird damit zu einem einfachen Trick der Benutzeroberfläche. Viele andere Web-2.0-Präsenzen verfügen über ähnliche Mechanismen – betrachten Sie etwa die „Tags“, die von YouTube [http://www.youtube.com/] und Flickr [http://www.flickr.com/] verwendet werden, „Kategorien“, die auf Blog Posts angewendet werden, usw. Den Menschen ist heutzutage bewusst, dass die Organisation von Daten kritisch ist, die Art und Weise der Organisation jedoch ein flexibles Konzept sein muss. Das alte Paradigma der Dateien und Ordner ist für manche Anwendung zu starr.

Subversions Unterstützung von Änderungslisten erlaubt Ihnen die Erstellung von Änderungslisten durch das Anbringen von Labels auf Dateien, die Sie mit dieser Änderungsliste in Verbindung bringen wollen, das Entfernen dieser Label und die Einschränkung des Wirkbereiches von Unterbefehlen auf die Dateien mit einem bestimmten Label. In diesem Abschnitt werden wir einen detaillierten Blick darauf werfen, wie so etwas gemacht werden kann.

Erstellen und Bearbeiten von Änderungslisten

Sie können Änderungslisten mit dem Befehl **svn changelist** erstellen, bearbeiten und löschen. Genauer gesagt verwenden Sie diesen Befehl, um die Verbindung einer Änderungsliste mit einer Datei der Arbeitskopie herzustellen oder aufzulösen. Eine Änderungsliste wird tatsächlich erstmals dann erstellt, wenn Sie eine Datei mit diesem Änderungslisten-Label versehen; sie wird gelöscht, wenn dieses Label von der letzten damit versehenen Datei entfernt wird. Sehen wir uns einmal einen Anwendungsfall an, der diese Konzepte vorstellt.

Harry beseitigt einige Fehler in der mathematischen Logik der Rechneranwendung. Seine Arbeit veranlasst ihn, einige Dateien zu ändern:

```
$ svn status
M       integer.c
M       mathops.c
$
```

Während er die Fehlerbehebung testet, bemerkt Harry, dass seine Änderungen einen tangential in Bezug stehenden Fehler der Logik der Benutzerschnittstelle in `button.c` ans Tageslicht bringen. Harry entschließt sich, auch diesen Fehler als eine von seinen Mathe-Reparaturen getrennte Übergabe zu beheben. In einer kleinen Arbeitskopie mit nur einer handvoll Dateien und wenigen logischen Änderungen kann Harry wahrscheinlich seine zwei logisch gruppierten Änderungen ohne Problem im Kopf auseinander halten. Heute jedoch wird er, um den Autoren diesen Buchs einen Gefallen zu tun, die Änderungslisten von Subversion verwenden.

Harry erstellt zunächst eine Änderungsliste und stellt sie in Beziehung zu den beiden von ihm bereits geänderten Dateien. Er macht das, indem er diesen Dateien mit dem Befehl **svn changelist** die selbe, frei wählbare Änderungsliste zuweist:

```
$ svn changelist math-fixes integer.c mathops.c
Pfad »integer.c« ist nun ein Element der Änderungsliste »math-fixes«.
Pfad »mathops.c« ist nun ein Element der Änderungsliste »math-fixes«.
$ svn status

--- Änderungsliste »math-fixes«:
M      integer.c
M      mathops.c
$
```

Wie Sie sehen können, spiegelt die Ausgabe von **svn status** diese neue Gruppierung wider.

Harry legt nun los, das sekundäre Problem der Benutzerschnittstelle zu beheben. Da er weiß, welche Datei er ändern wird, weist er auch diesen Pfad einer Änderungsliste zu. Unglücklicherweise weist Harry diese dritte Datei achtlos derselben Änderungsliste wie den beiden vorigen Dateien zu:

```
$ svn changelist math-fixes button.c
Pfad »button.c« ist nun ein Element der Änderungsliste
»math-fixes«.
```

```
$ svn status

--- Änderungsliste »math-fixes«:
        button.c
M       integer.c
M       mathops.c
$
```

Zum Glück entdeckt Harry seinen Fehler. An dieser Stelle hat er zwei Optionen. Er kann die Verbindung zur Änderungsliste von `button.c` lösen und dann einen unterschiedlichen Listennamen zuweisen:

```
$ svn changelist --remove button.c
Pfad »button.c« ist nicht länger ein Element einer Änderungsliste.
$ svn changelist ui-fix button.c
msgstr "Pfad »button.c« ist nun ein Element der Änderungsliste »ui-fix«.
$
```

Oder er kann sich das Entfernen sparen und bloß einen neuen Änderungslisten-Namen zuweisen. In diesem Fall wird Subversion Harry warnen, dass `button.c` von der ersten Änderungsliste entfernt wird:

```
$ svn changelist ui-fix button.c
svn: warnung: Entferne »button.c« aus Änderungsliste »math-fixes«.
Pfad »button.c« ist nun ein Element der Änderungsliste »ui-fix«.
$ svn status

--- Änderungsliste »ui-fix«:
        button.c

--- Änderungsliste »math-fixes«:
M       integer.c
M       mathops.c
$
```

Harry hat nun zwei unterschiedliche Änderungslisten in seiner Arbeitskopie, und **svn status** gruppiert seine Ausgaben nach den Bezeichnungen dieser Änderungslisten. Beachten Sie, dass Harry die Datei `button.c` zwar noch nicht geändert hat, sie aber trotzdem als interessant in der Ausgabe von **svn status** erscheint, da eine Verknüpfung mit einer Änderungsliste besteht. Änderungslisten können jederzeit Dateien hinzugefügt oder entzogen werden, egal, ob sie lokale Änderungen beinhalten.

Harry behebt nun das Problem der Benutzerschnittstelle in `button.c`.

```
$ svn status

--- Änderungsliste »ui-fix«:
M       button.c

--- Änderungsliste »math-fixes«:
M       integer.c
M       mathops.c
$
```

Änderungslisten als Befehlsfilter

Die visuelle Gruppierung, die Harry in der Ausgabe von **svn status** im vorangegangenenen Abschnitt sieht, ist ganz nett, jedoch nicht so richtig nützlich. Der Befehl **status** ist nur einer von mehreren, die er vielleicht in seiner Arbeitskopie ausführen möchte. Erfreulicherweise können viele andere Befehle von Subversion auf Änderungslisten arbeiten, wenn die Option `--changelist` verwendet wird.

Wenn ihnen die Option `--changelist` mitgegeben wird, beschränken Befehle von Subversion ihren Wirkbereich auf die Dateien, die mit einer bestimmten Änderungsliste verknüpft sind. Falls Harry nun die eigentlichen Änderungen sehen möchte, die er an Dateien in seiner Änderungsliste `math-fixes` vorgenommen hat, *könnte* er in der Kommandozeile des Befehls **svn diff** ausdrücklich nur die Dateien angeben, die diese Änderungsliste ausmachen.

```
$ svn diff integer.c mathops.c
Index: integer.c
===================================================================
--- integer.c (revision 1157)
+++ integer.c (working copy)
…
Index: mathops.c
===================================================================
--- mathops.c (revision 1157)
+++ mathops.c (working copy)
…
$
```

Bei ein paar Dateien funktioniert das einwandfrei, was wäre aber, wenn Harrys Änderung 20 oder 30 Dateien beträfe? Das wäre eine schrecklich lange Liste ausdrücklich aufgeführter Dateinamen. Da Harry nun jedoch Änderungslisten verwendet, kann er das explizite Aufführen der Dateimenge in seiner Änderungsliste von nun an vermeiden und stattdessen nur den Namen der Änderungsliste angeben:

```
$ svn diff --changelist math-fixes
Index: integer.c
===================================================================
--- integer.c (revision 1157)
+++ integer.c (working copy)
…
Index: mathops.c
===================================================================
--- mathops.c (revision 1157)
+++ mathops.c (working copy)
…
$
```

Und wenn es an der Zeit ist, zu übertragen, kann Harry wieder die Option `--changelist` verwenden, um die Übertragung auf die Dateien einer bestimmten Änderungsliste zu beschränken. Er könnte seine Fehlerbehebung an der Benutzerschnittstelle etwa so übertragen:

```
$ svn ci -m "Fix a UI bug found while working on math logic." \
      --changelist ui-fix
Sende          button.c
Übertrage Daten .
Revision 1158 übertragen.
$
```

Der Befehl **svn commit** verfügt über eine zweite Option in Zusammenhang mit Änderungslisten: `--keep-changelists`. Normalerweise werden Verknüpfungen zu Änderungslisten nach der Übertragung von Dateien aufgelöst. Wenn jedoch `--keep-changelists` angegeben wird, bewahrt Subversion die Verknüpfung der übertragenen (und nun unveränderten)

Dateien zur Änderungsliste. In allen Fällen berührt die Übertragung von Dateien einer Änderungsliste alle anderen Änderungslisten nicht.

```
$ svn status

--- Änderungsliste »math-fixes«:
M      integer.c
M      mathops.c
$
```

Die Option `--changelist` wirkt lediglich als ein Filter für die Objekte eines Subversion-Befehls und fügt keine neuen Objekte hinzu. Beispielsweise ist bei einem Übertragungsbefehl, der als **`svn commit /path/to/dir`** gegeben wird, das Objekt das Verzeichnis `/path/to/dir` und seine Kinder (unendlich tief). Wenn Sie diesem Befehl dann die Angabe einer Änderungsliste hinzufügen, werden nur die Dateien in und unterhalb von `/path/to/dir` als zu übertragene Objekte betrachtet, die mit dieser Änderungsliste verknüpft sind – die Übertragung wird keine Dateien von anderen Stellen (etwa `/path/to/another-dir`) umfassen, egal, ob sie mit der Änderungsliste verknüpft sind, auch wenn sie zur selben Arbeitskopie gehören wie die Objekte des Befehls.

Auch der Befehl **svn changelist** versteht die Option `--changelist`. Er ermöglicht Ihnen das schnelle Umbenennen oder Entfernen einer Änderungsliste:

```
$ svn changelist math-bugs --changelist math-fixes --depth infinity .
svn: warnung: Entferne »integer.c« aus Änderungsliste »math-fixes«.
Pfad »integer.c« ist nun ein Element der Änderungsliste »math-bugs«.
svn: warnung: Entferne »mathops.c« aus Änderungsliste »math-fixes«.
Pfad »mathops.c« ist nun ein Element der Änderungsliste »math-bugs«.
$ svn changelist --remove --changelist math-bugs --depth infinity .
Pfad »integer.c« ist nicht länger ein Element einer Änderungsliste
Pfad »mathops.c« ist nicht länger ein Element einer Änderungsliste
$
```

Schließlich können sie die Option `--changelist` mehrfach für einen Befehl auf einer einzelnen Kommandozeile angeben. Das beschränkt den auszuführenden Befehl auf Dateien, die in irgendeiner der angegebenen Änderungslisten zu finden sind.

Einschränkungen von Änderungslisten

Die Änderungslisten von Subversion sind ein praktisches Werkzeug, um Dateien in Arbeitskopien zu gruppieren; allerdings besitzen sie ein paar Einschränkungen. Änderungslisten sind Artefakte einer bestimmten Arbeitskopie, was bedeutet, dass Änderungslisten-Zuweisungen nicht an das Projektarchiv weitergegeben und auch anderweitig nicht gemeinsam von anderen Benutzern verwendet werden können. Änderungslisten lassen sich nur Dateien zuordnen – momentan unterstützt Subversion nicht die Verwendung von Änderungslisten für Verzeichnisse. Schließlich können Sie einer gegebenen Datei der Arbeitskopie höchstens eine Änderungsliste zuweisen. Hier passen die Analogien der Blog-Posts-Kategorien oder der Foto-Tags nicht mehr – sollte es notwendig sein, eine Datei mehreren Änderungslisten zuzuweisen, haben Sie Pech.

Das Netzwerkmodell

Manchmal müssen Sie verstehen, wie Ihr Subversion-Client mit seinem Server kommuniziert. Die Netzwerkschicht von Subversion ist abstrahiert, was bedeutet, dass die Clients von Subversion das gleiche allgemeine Verhalten an den Tag legen, egal mit welcher Art von Server sie zusammenarbeiten. Ob sie im HTTP-Protokoll (`http://`) mit dem Apache HTTP-Server oder im maßgeschneiderten Subversion Protokoll (`svn://`) mit **svnserve** sprechen, das grundlegende Netzwerkmodell ist das selbe. In diesem Abschnitt werden wir die Grundlagen dieses Netzwerkmodells erläutern, auch wie Subversion die Anmeldung und die Berechtigung handhabt.

Anfragen und Antworten

Den größten Teil seiner Zeit verbringt der Subversion-Client mit der Verwaltung von Arbeitskopien. Wenn er jedoch Informationen von einem entfernten Projektarchiv benötigt, stellt er eine Anfrage über das Netz, und der Server erwiedert mit einer passenden Antwort. Die Details des Netzwerkprotokolls sind dem Benutzer verborgen – der Client versucht, auf einen URL zuzugreifen, und abhänging vom URL-Schema wird ein bestimmtes Protokoll verwendet, um mit dem Server Verbindung aufzunehmen (siehe „Projektarchive adressieren“).

Rufen Sie **`svn --version`** auf, um zu sehen, welche URL-Schemas und Protokolle versteht.

Wenn der Server-Prozess eine Anfrage eines Clients erhält, verlangt er häufig, dass der Client sich identifiziert. Er sendet eine Authentisierungsaufforderung an den Client und der Client antwortet, indem er *Zugangsdaten* zurückschickt. Sobald die Anmeldung abgeschlossen ist, antwortet der Server mit den ursprünglich vom Client angefragten Informationen. Beachten Sie, dass dieses System sich von solchen wie CVS unterscheidet, bei denen der Client von sich aus dem Server Zugangsdaten anbietet („sich anmeldet“), bevor überhaupt eine Anfrage erfolgt. In Subversion werden die Zugangsdaten vom Server „eingezogen“, indem der Client zum passenden Zeitpunkt aufgefordert wird, und nicht indem der Client sie „abliefert“. Das macht gewisse Operationen eleganter. Wenn ein Server beispielsweise so konfiguriert ist, dass jedem auf der Welt erlaubt ist, ein Projektarchiv zu lesen, wird der Server niemals eine Authentisierungsaufforderung ausgeben, wenn ein Client **svn checkout** versucht.

Falls die einzelnen Netzwerkanfragen des Clients zur Erstellung einer neuen Revision im Projektarchiv führen (z.B. **svn commit**), verwendet Subversion den mit diesen Anfragen verknüpften authentifizierten Benutzernamen als Autor der Revision. Das bedeutet, dass der Name des authentifizierten Benutzers als Wert der Eigenschaft `svn:author` der neuen Revision zugewiesen wird (siehe „Subversion-Eigenschaften“). Falls der Client nicht authentifiziert wurde (d.h., falls der Server niemals eine Authentisierungsaufforderung ausgegeben hat), bleibt die Revisionseigenschaft `svn:author` leer.

Client-Zugangsdaten

Viele Subversion-Server sind so konfiguriert, dass sie eine Authentifikation benötigen. Manchmal sind anonyme Leseoperationen erlaubt, wohingegen Schreiboperationen authentifiziert sein müssen. In anderen Fällen benötigen Lese- und Schreiboperationen gleichermaßen eine Authentifikation. Die unterschiedlichen Server-Optionen von Subversion verstehen unterschiedliche Authentifikationsprotokolle, aus Anwendersicht jedoch bedeutet Authentifikation normalerweise Anwendernamen und Passwörter. Subversion-Clients bieten verschiedene Möglichkeiten, die Zugangsdaten eines Anwenders abzurufen und zu speichern, vom interaktiven Abfragen des Anwendernamens und Passworts bis zu verschlüsselten oder unverschlüsselten Zwischenspeichern auf der Platte.

Der sicherheitsbewussten Leser wird an dieser Stelle sofort einen Grund für Besorgnis erkennen. „Passwörter auf der Platte speichern? Das ist schrecklich! So etwas sollte man nie machen!“ Keine Angst! Das hört sich schlimmer an, als es ist. Die folgenden Abschnitte erörtern die verschiedenen Arten der Zugangsdatenzwischenspeicherung, die Subversion verwendet, wann es sie verwendet und wie man diese Funktionalität ganz oder teilweise abstellen kann.

Zwischenspeichern von Zugangsdaten

Subversion bietet Abhilfe gegen die lästige Notwendigkeit, dass Anwender jedes Mal Ihren Benutzernamen und das Passwort eintippen müssen. Standardmäßig werden die Zugangsdaten mit einer Kombination des Rechnerenamens des Servers, dem Port und der Zugangsregion auf Platte gespeichert, sobald der Kommandozeilen-Client erfolgreich auf eine Authentisierungsaufforderung des Servers antwortet. Auf diesen Zwischenspeicher wird dann künftig zugegriffen, was eine erneute Eingabe der Zugangsdaten durch den Anwender vermeidet. Falls scheinbar passende Zugangsdaten nicht im Zwischenspeicher verfügbar sind oder die zwischengespeicherten Daten nicht authentifiziert werden können, wird der Client den Anwender wieder standardmäßig zur Eingabe der notwendigen Informationen auffordern.

Die Entwickler von Subversion erkennen, das auf Platte zwischengespeicherte Zugangsdaten ein Sicherheitsrisiko darstellen können. Um dieses zu verringern, arbeitet Subversion mit den durch das Betriebssystem und die Umgebung bereitgestellten Mechanismen, um das Risiko der Kompromittierung dieser Daten zu minimieren.

- Unter Windows speichert der Subversion-Client Passwörter im Verzeichnis `%APPDATA%/Subversion/auth/`. Unter Windows 2000 und dessen Nachfolger verwendet der Subversion-Client die Standard-Kryptographiedienste von Windows, um das Passwort auf Platte zu verschlüsseln. Da der Schlüssel durch Windows verwaltet wird und mit den benutzereigenen Zugangsdaten verknüpft ist, kann nur der Benutzer selbst das zwischengespeicherte Passwort entschlüsseln. (Beachten Sie,

dass beim Zurücksetzen des Windows-Benutzerkonto-Passworts durch einen Administrator alle zwischengespeicherten Passwörter nicht mehr zu entschlüsseln sind. Der Subversion-Client verhält sich dann so, als wären sie nicht vorhanden und fragt die Passwörter gegebenenfalls ab.)

- Unter Mac OS X speichert der Subversion-Client auf ähnliche Weise alle Passwörter des Projektarchivs im Login-Keyring (durch den Keychain-Dienst verwaltet), der durch das Passwort des Benutzerkontos geschützt ist. Benutzerseitige Einstellungen können zusätzliche Richtlinien in Kraft setzen, etwa, dass das Benutzerkontopasswort jedesmal eingegeben werden muss, wenn das Passwort von Subversion verwendet wird.

- Für andere Unix-ähnliche Betriebssysteme existiert kein einzelner standardisierter „Schlüsselring“-Dienst. Der Subversion-Client weiß aber, wie er Passwörter sicher mit den Diensten „GNOME Keyring“ und „KDE Wallet“ speichern kann. Bevor unverschlüsselte Passwörter im Zwischenspeicherbereich `~/.subversion/auth/` gespeichert werden, fragt der Subversion-Client den Anwender, ob er das machen darf. Beachten Sie, dass der `auth/`-Bereich zur Zwischenspeicherung immer noch über Zugangsrechte geschützt ist, so dass nur der Anwender (Eigentümer) die dortigen Daten lesen kann und nicht die gesamte Welt. Die betriebssystemeigenen Dateizugriffsrechte schützen die Passwörter vor anderen nicht-administrativen Anwendern auf dem selben System, vorausgesetzt, sie haben keinen direkten physischen Zugriff zum Speichermedium des Heimverzeichnisses oder dessen Sicherungen.

Natürlich sind keine dieser Mechanismen perfekt für den echt Paranoiden. Für die Leute, die Bequemlichkeit für die größte Sicherheit opfern möchten, bietet Subversion verschiedene Möglichkeiten, das System der Zwischenspeicherung ganz abzuschalten.

Unterbinden der Zwischenspeichrung von Passwörtern

Wenn Sie eine Subversion-Operation durchführen, die von Ihnen eine Authentisierung verlangt, versucht Subversion standardmäßig, Ihre Zugangsdaten verschlüsselt auf Platte zwischenzuspeichern. Auf manchen Systemen kann es sein, dass Subversion Ihre Zugangsdaten nicht verschlüsseln kann. In diesen Situationen, fragt Subversion nach, ob Sie möchten, dass die Zugangsdaten im Klartext auf Platte gespeichert werden:

```
$ svn checkout https://host.example.com:443/svn/private-repo
-----------------------------------------------------------------------
ACHTUNG! Ihr Password für den Anmeldungstext (realm)

   <https://host.example.com:443> Subversion Repository

kann auf der Platte nur unverschlüsselt gespeichert werden! Es wird
empfohlen, falls möglich Ihr System so zu konfigurieren, dass Subversion
Passwörter verschlüsselt speichern kann. Siehe die Dokumentation für
Details.

Sie können ein weiteres Anzeigen dieser Warnung verhindern, indem Sie
den Wert der Option »store-plaintext-passwords« in
»/tmp/servers«
entweder auf »ja« oder »nein« setzen.
-----------------------------------------------------------------------
Passphrase unverschlüsselt speichern (ja/nein)?
```

Wenn Sie die Bequemlichkeit nutzen wollen, nicht ständig für künftige Operationen Ihr Passwort erneut eingeben zu müssen, können Sie mit `ja` auf diese Aufforderung antworten. Falls Sie Bedenken haben, Ihr Passwort im Klartext zwischenzuspeichern und nicht jedesmal erneut danach gefragt werden wollen, können Sie die Zwischenspeicherung von Passwörtern im Klartext entweder permanent oder abhängig vom Server abstellen.

Wenn Sie sich Gedanken dazu machen, wie Sie das Zwischenspeichern von Passwörtern unter Subversion verwenden wollen, sollten Sie die Richtlinien berücksichtigen, die für Ihren Rechner gelten – viele Firmen haben strenge Regeln für die Speicherung der Zugangsdaten von Mitarbeitern.

Um das Zwischenspeichern von Passwörtern im Klartext permanent zu unterbinden, fügen Sie die Zeile `store-plaintext-passwords = no` im Abschnitt `[global]` der Konfigurationsdatei `servers` auf dem lokalen Rechner

ein. Um die Zwischenspeicherung von Passwörtern für einen bestimmten Server zu unterbinden, verwenden Sie dieselbe Einstellung im entsprechenden Gruppenabschnitt der Konfigurationsdatei `servers`. (Für Details, siehe „Konfigurationsoptionen“ in Kapitel 7, *Subversion an Ihre Bedürfnisse anpassen*.)

Um die Zwischenspeicherung von Passwörtern vollständig für irgendeine einzelne Subversion-Kommandozeilenoperation abzuschalten, übergeben Sie dem Befehl die Option `--no-auth-cache`. Um die Zwischenspeicherung permanent vollständig zu unterbinden, fügen Sie der Konfigurationsdatei für Subversion auf dem lokalen Rechner die Zeile `store-passwords = no` hinzu.

Entfernen zwischengespeicherter Zugangsdaten

Manchmal möchten Benutzer bestimmte Zugangsdaten aus dem Zwischenspeicher entfernen. Hierzu müssen Sie in den `auth/`-Bereich gehen und die entsprechende Zwischenspeicherdatei manuell löschen. Die Zugangsdaten werden in individuellen Dateien zwischengespeichert; falls Sie in jede Datei hineinsehen, werden Sie Schlüssel und Werte entdecken. Der Schlüssel `svn:realmstring` beschreibt den bestimmten Anmeldebereich, zu dem die Datei gehört:

```
$ ls ~/.subversion/auth/svn.simple/
5671adf2865e267db74f09ba6f872c28
3893ed123b39500bca8a0b382839198e
5c3c22968347b390f349ff340196ed39

$ cat ~/.subversion/auth/svn.simple/5671adf2865e267db74f09ba6f872c28

K 8
username
V 3
joe
K 8
password
V 4
blah
K 15
svn:realmstring
V 45
<https://svn.domain.com:443> Joe's repository
END
```

Sobald Sie die passende Datei gefunden haben, können Sie sie einfach löschen.

Anmeldung über die Kommandozeile

Alle Kommandozeilenaktionen von Subversion erlauben die Optionen `--username` und `--password`, die es Ihnen ermöglichen, Ihren Anwendernamen bzw. Ihr Passwort anzugeben, damit Subversion Sie nicht nach diesen Informationen fragen muss. Das ist besonders dann praktisch, wenn Sie Subversion aus einem Script heraus aufrufen wollen, und Sie sich nicht darauf verlassen können, dass Subversion in der Lage ist, gültige zwischengespeicherte Zugangsdaten für Sie zu finden. Diese Optionen sind auch dann hilfreich, falls Subversion bereits Anmeldedaten von Ihnen zwischengespeichert hat, aber diese nicht von Ihnen verwendet werden sollen. Vielleicht teilen sich mehrere Systemanwender ein Zugangskonto, haben jedoch verschiedene Identitäten unter Subversion. Sie können die Option `--password` weglassen, falls Sie möchten, dass Subversion nur den angegebenen Anwendernamen verwenden soll, Sie aber trotzdem auffordern soll, das Passwort dieses Anwenders einzugeben.

Schlusswort zur Anmeldung

Ein letztes Wort zum Anmeldungsverhalten von **svn**, im Besonderen bezüglich der Optionen `--username` und `--password`. Viele Unterbefehle des Clients akzeptieren diese Optionen, jedoch ist es wichtig, zu verstehen, dass durch deren Verwendung *nicht* automatisch Zugangsdaten an den Server gesendet werden. Wie bereits erörtert wurde, werden die Zugangsdaten vom Server „eingezogen“, falls er es für notwendig hält; der Client kann sie nicht nach belieben „abliefern“. Wurde ein Benutzername und/oder ein Passwort als Optionen mitgegeben, werden sie dem Server nur auf Verlangen vorgelegt. Üblicherweise werden diese Optionen verwendet, um sich als ein anderer Benutzer zu authentisieren als derjenige, den Subversion standardmäßig gewählt hätte (etwa Ihr Anmeldename), oder falls Sie die interaktive Abfrage vermeiden möchten (etwa beim Aufruf von **svn** aus einem Skript).

Ein verbreiteter Fehler ist die Fehlkonfigurierung eines Servers, so dass er nie eine Aufforderung zur Authentisierung ausgibt. Falls Benutzer dann die Optionen `--username` und `--password` an den Client übergeben, wundern sie sich, dass sie nie verwendet werden, d.h., es sieht so aus, dass neue Revisionen anonym übergeben worden sind!

An dieser Stelle sei abschließend zusammengefasst, wie sich ein Subversion-Client bei Erhalt einer Aufforderung zur Authentisierung verhält.

1. Zunächst prüft der Client, ob der Benutzer irgendwelche Zugangsdaten als Kommandozeilenoptionen (`--username` und/oder `--password`) angegeben hat. Falls ja, versucht der Client, diese Zugangsdaten zur Authentisierung gegenüber dem Server zu verwenden.

2. Falls keine Zugangsdaten als Kommandozeilenoptionen angegeben worden sind oder die übergebenen ungültig waren, verwendet der Client den Rechnernamen des Servers, den Port und den Anmeldebereich, um damit im Laufzeitkonfigurationsbereich `auth/` nach passenden zwischengespeicherten Zugangsdaten zu suchen. Falls solche vorhanden sind, probiert er, sich hiermit zu authentisieren.

3. Falls letztendlich alle vorherigen Mechanismen keine erfolgreiche Authentisierung des Benutzers gegen den Server bewirken, greift der Client auf eine interaktive Abfrage der Zugangsdaten zurück (sofern ihm das nicht durch die Option `--non-interactive` oder client-spezifischer Äquivalente untersagt wurde).

Falls sich der Client durch irgendeine dieser Methoden erfolgreich authentisiert, versucht er, die Zugangsdaten auf der Platte zwischenzuspeichern (sofern der Benutzer dieses Verhalten nicht, wie oben beschrieben, unterbunden hat).

Zusammenfassung

Nachdem Sie dieses Kapitel gelesen haben, sollten Sie ein ziemlich gutes Verständnis von einigen Eigenschaften Subversions haben, die zwar nicht bei *jedem* Umgang mit Ihrem Versionskontrollsystem benutzt werden, deren Kenntnis jedoch durchaus von Vorteil sein kann. Hören Sie an dieser Stelle jedoch noch nicht auf! Lesen Sie das folgende Kapitel, in dem Sie Zweige, Tags und das Zusammenführen kennenlernen werden. Dann werden Sie den Subversion-Client fast vollständig beherrschen. Obwohl uns unsere Anwälte untersagen, Ihnen irgendetwas zu versprechen, könnte Sie dieses zusätzliche Wissen messbar cooler werden lassen.[13]

[13] Kein Kaufzwang. Es gelten die allgemeinen Geschäftsbedingungen. Keine Coolness-Garantie, ausdrücklich oder anderweitig. Abweichungen möglich.

Kapitel 4. Verzweigen und Zusammenführen

„#### (Der Edle pflegt die Wurzel)“

—Konfuzius

Verzweigen (Branching), Etikettieren (Tagging) und Zusammenführen (Merging) sind Konzepte, die fast allen Versionskontrollsystemen gemein sind. Falls Sie mit diesen Begriffen nicht vertraut sein sollten, geben wir in diesem Kapitel eine gute Einführung. Falls Sie damit vertraut sind, werden Sie es hoffentlich interessant finden, zu sehen, wie Subversion diese Konzepte implementiert.

Verzweigen ist ein grundlegender Teil der Versionskontrolle. Falls Sie Subversion erlauben wollen, Ihre Daten zu verwalten, ist dies eine Fähigkeit, von der Sie letztendlich abhängig sein werden. Dieses Kapitel geht davon aus, dass Sie mit den grundlegenden Konzepten von Subversion vertraut sind (Kapitel 1, *Grundlegende Konzepte*).

Was ist ein Zweig?

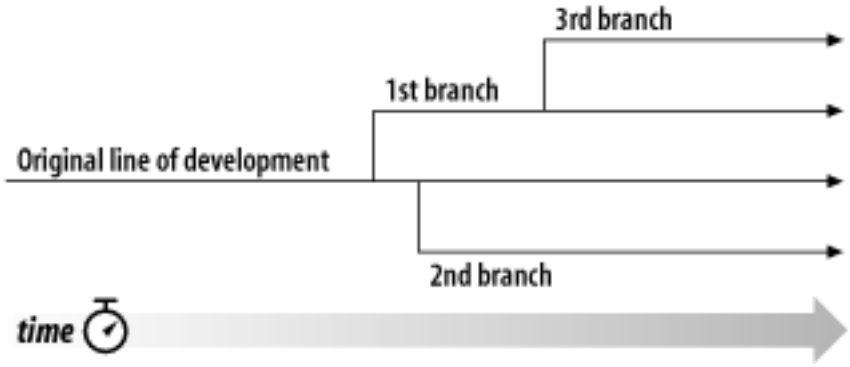
Angenommen, Ihre Aufgabe ist es, ein Dokument für eine Abteilung Ihrer Firma zu pflegen – eine Art Handbuch. Eines Tages fragt eine andere Abteilung nach dem gleichen Handbuch, jedoch an einigen Stellen für ihre Bedürfnisse „abgewandelt“, da sie auf etwas andere Weise arbeiten.

Was machen Sie in dieser Situation? Sie machen das Offensichtliche: Sie erstellen eine Kopie Ihres Dokumentes und beginnen, die beiden Kopien getrennt zu pflegen. Sobald Sie irgendeine Abteilung auffordert, kleine Änderungen vorzunehmen, pflegen Sie diese in die eine oder andere Kopie ein.

Oftmals möchten Sie die selbe Änderung in beiden Kopien machen. Wenn Sie zum Beispiel einen Schreibfehler in der ersten Kopie entdecken, ist es sehr wahrscheinlich, dass dieser Fehler auch in der zweiten Kopie vorliegt. Schließlich sind die beiden Dokumente fast gleich; sie unterscheiden sich nur in kleinen Dingen.

Das ist das Grundkonzept eines *Zweigs* (Branch) – nämlich eine Entwicklungslinie, die unabhängig von einer anderen existiert, jedoch über eine gemeinsame Geschichte verfügt, wenn lang genug in der Zeit zurück gegangen wird. Ein Zweig beginnt sein Leben stets als eine Kopie von etwas und läuft von da an weiter, wobei er seine eigene Geschichte erzeugt (siehe Abbildung 4.1, „Entwicklungszweige“).

Abbildung 4.1. Entwicklungszweige



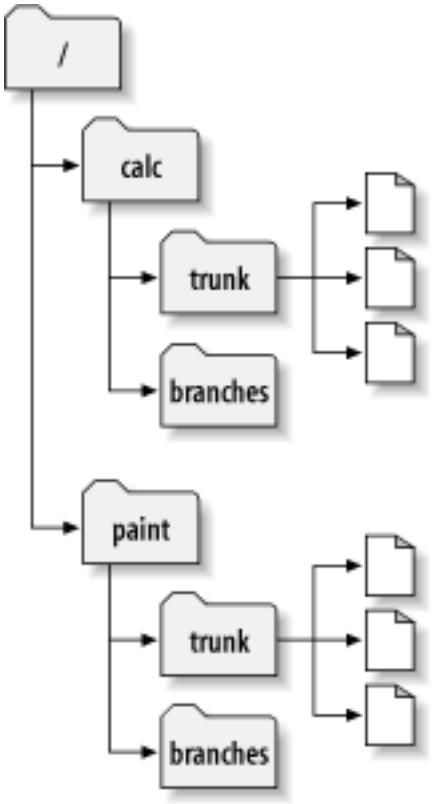
Subversion verfügt über Befehle, die Ihnen helfen, parallele Zweige Ihrer Dateien und Verzeichnisse zu verwalten. Es erlaubt Ihnen, durch das Kopieren Ihrer Daten, Zweige zu erstellen und merkt sich, dass die Zweige untereinander in Beziehung stehen. Es hilft Ihnen auch, Änderungen von einem Zweig auf den anderen zu duplizieren. Schließlich ermöglicht es, dass Teile Ihrer Arbeitskopie verschiedene Zweige repräsentieren können, was Ihnen während Ihrer täglichen Arbeit erlaubt, verschiedene Entwicklungslinien zu „mischen und gegenüberzustellen“.

Verwenden von Zweigen

An dieser Stelle sollten Sie verstehen, wie jede Übergabe an das Projektarchiv dort einen neuen Zustand des Dateibaums (genannt „Revision“) erzeugt. Wenn nicht, blättern Sie zurück und lesen Sie in „Revisionen“ über Revisionen nach.

Für dieses Kapitel verwenden wir das Beispiel aus Kapitel 1, *Grundlegende Konzepte*. Erinnern Sie sich, dass Sie und Ihre Mitarbeiterin Sally sich ein Projektarchiv teilen, das zwei Projekte beinhaltet: `paint` und `calc`. Beachten Sie, dass in Abbildung 4.2, „Projektarchiv-Struktur zu Beginn“ dieses Mal jedoch jedes Projektverzeichnis Unterverzeichnisse namens `trunk` und `branches` beinhaltet. Der Grund hierfür wird bald klar sein.

Abbildung 4.2. Projektarchiv-Struktur zu Beginn



Wie vorher sei hier angenommen, dass sowohl Sally als auch Sie Arbeitskopien des „calc“ Projektes besitzen. Ausdrücklich hat jeder von Ihnen eine Arbeitskopie von `/calc/trunk`. Alle Dateien des Projektes befinden sich in diesem Unterverzeichnis statt in `/calc` selber, da Ihr Team entschieden hat, dass in `/calc/trunk` die „Hauptlinie“ der Entwicklung stattfindet.

Sagen wir mal, dass Sie die Aufgabe bekommen haben, ein großes Stück Software umzusetzen. Die Erstellung benötigt eine lange Zeit und berührt alle Dateien im Projekt. Das Problem, dass sofort auftaucht ist, dass Sie Sally nicht in die Quere kommen möchten, die gerade hier und da kleinere Fehler beseitigt. Sie ist davon abhängig, dass die letzte Version des Projektes (in `/calc/trunk`) stets benutzbar ist. Wenn Sie nun damit beginnen, Stück für Stück Ihre Änderungen zu übergeben, werden Sie die Dinge für Sally (und auch für andere Teammitglieder) bestimmt in Unordnung bringen.

Eine Strategie ist, sich in ein Loch zu verkriechen: Sie und Sally können für eine Woche oder zwei den Informationsaustausch einstellen. Das heißt, Sie fangen damit an, die Dateien Ihrer Arbeitskopie auszuräumen und umzuorganisieren, ohne Änderungen zu übergeben oder die Arbeitskopie zu aktualisieren, bevor Sie mit Ihrer Arbeit vollständig fertig sind. Das wirft allerdings einige Probleme auf. Erstens ist das nicht sehr sicher. Viele Leute möchten Ihre Arbeit regelmäßig ins Projektarchiv sichern, für den Fall, dass etwas Schlimmes mit der Arbeitskopie passieren könnte. Zweitens ist das nicht sehr flexibel. Falls Sie Ihre Arbeit an mehreren Rechnern verrichten (vielleicht haben Sie eine Arbeitskopie von `/calc/trunk` auf zwei unterschiedlichen Maschinen), müssten Sie entweder alle Änderungen manuell hin und her kopieren oder die gesamte Arbeit an nur einem Rechner erledigen. Ebenso schwierig wäre es, Ihre Änderungen mit anderen zu teilen. Eine weit verbreitete „beste Vorgehensweise“ ist es, Ihren Mitarbeitern zu erlauben, während Sie mit Ihrer Arbeit fortfahren, Ihre bisherigen

Ergebnisse zu überprüfen. Wenn niemand Ihre unmittelbaren Änderungen sieht, haben Sie keine möglichen Rückmeldungen und es könnte sein, dass Sie für Wochen einen falschen Weg einschlagen, bevor es jemand aus Ihrem Team bemerkt. Schließlich könnte es am Ende, wenn Sie mit Ihren Änderungen fertig sind, sehr schwierig sein, Ihr Arbeitsergebnis wieder mit dem Hauptteil der Quelltexte Ihrer Firma zusammenzuführen. Sally (und andere) hätten viele andere Änderungen ins Projektarchiv übergeben haben können, die sich schwer in Ihre Arbeitskopie einarbeiten ließen – besonders, falls Sie **svn update** nach Wochen der Isolierung ausführen.

Die bessere Lösung ist, Ihren eigenen Zweig oder Ihre eigene Entwicklungslinie im Projektarchiv zu erzeugen. Dies erlaubt Ihnen, Ihre halbfertigen Arbeitsergebnisse regelmäßig zu sichern, ohne andere zu stören; dennoch können Sie selektiv Informationen mit Ihren Kollegen teilen. Im Weiteren werden Sie sehen, wie das funktioniert.

Erzeugen eines Zweiges

Es ist sehr einfach, einen Zweig zu erzeugen – Sie erstellen mit dem Befehl **svn copy** eine Kopie des Projektes im Projektarchiv. Subversion kann nicht nur Dateien, sondern auch komplette Verzeichnisse kopieren. In diesem Fall möchten Sie eine Kopie des Verzeichnisses `/calc/trunk` machen. Wo soll die neue Kopie angelegt werden? Wo Sie wünschen – es ist eine Frage der Projektkonventionen. Sagen wir mal, dass Ihr Team die Konvention vereinbart hat, Zweige im Bereich `/calc/branches` des Projektarchivs anzulegen, und Sie Ihren Zweig `my-calc-branch` nennen möchten. Sie werden ein neues Verzeichnis `/calc/branches/my-calc-branch` anlegen, das als Kopie von `/calc/trunk` beginnt.

Sie haben vielleicht schon gesehen, wie mit **svn copy** in einer Arbeitskopie eine Datei auf eine andere kopiert wird. Es kann allerdings auch verwendet werden, um eine „entfernte“ Kopie innerhalb des Projektarchivs durchzuführen. Kopieren Sie einfach einen URL auf einen anderen:

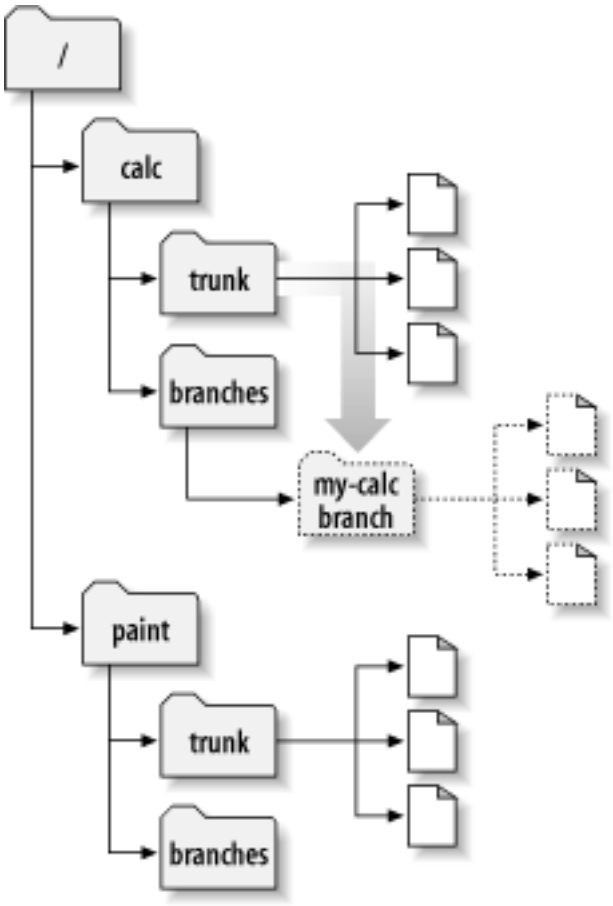
```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/my-calc-branch \
      -m "Privaten Zweig von /calc/trunk angelegt."

Revision 341 übertragen.
```

Dieser Befehl bewirkt eine fast sofortige Übergabe im Projektarchiv, wobei in Revision 341 ein neues Verzeichnis erzeugt wird. Das neue Verzeichnis ist eine Kopie von `/calc/trunk`. Dies wird in Abbildung 4.3, „Projektarchiv mit neuer Kopie“ gezeigt. [1] Obwohl es auch möglich ist, einen Zweig zu erzeugen, indem **svn copy** verwendet wird, um ein Verzeichnis innerhalb der Arbeitskopie zu duplizieren, wird dieses Vorgehen nicht empfohlen. Es kann in der Tat sehr langsam sein! Das client-seitige Kopieren eines Verzeichnisses besitzt einen linearen Zeitaufwand, da wirklich jede Datei und jedes Verzeichnis innerhalb dieser Arbeitskopie auf der lokalen Platte dupliziert werden muss. Das Kopieren eines Verzeichnisses auf dem Server jedoch besitzt einen konstanten Zeitaufwand und ist die Art und Weise, auf die die meisten Leute Zweige erstellen.

Abbildung 4.3. Projektarchiv mit neuer Kopie

[1] Subversion unterstützt nicht das Kopieren zwischen verschiedenen Projektarchiven. Wenn Sie mit **svn copy** oder **svn move** URLs verwenden, können Sie nur Objekte innerhalb desselben Projektarchivs kopieren oder verschieben.



Billige Kopien

Das Projektarchiv von Subversion ist auf eine besondere Weise konstruiert. Wenn Sie ein Verzeichnis kopieren, brauchen Sie sich keine Gedanken darüber zu machen, dass das Projektarchiv riesengroß wird – Subversion dupliziert tatsächlich überhaupt keine Daten. Stattdessen erzeugt es einen neuen Verzeichniseintrag, der auf einen *bestehenden* Baum verweist. Falls Sie ein erfahrener Unix-Benutzer sind, werden Sie erkennen, dass es sich um dasselbe Konzept handelt wie bei einem Hardlink. Während weitere Änderungen an den Dateien und Verzeichnissen unterhalb des kopierten Verzeichnisses gemacht werden, fährt Subversion fort, dieses Konzept anzuwenden wo es geht. Es dupliziert Daten nur dann, wenn es notwendig wird, verschiedene Versionen von Objekten auseinanderzuhalten.

Deshalb hören Sie Subversion-Benutzer oft von „billigen Kopien“ sprechen. Es spielt keine Rolle, wie umfangreich das Verzeichnis ist – es bedarf lediglich eines kleinen, konstanten Zeitaufwands und Speicherplatzes, um eine Kopie davon zu erstellen. Diese Fähigkeit ist tatsächlich die Grundlage für die Umsetzung von Übergaben in Subversion: Jede Revision ist eine „billige Kopie“ der vorhergehenden Revision mit ein paar Dingen, die sich im Innern geändert haben. (Um mehr hierüber zu lesen, gehen Sie auf die Website von Subversion und lesen Sie in den Subversion-Design-Dokumenten über die „bubble-up“-Methode.)

Natürlich sind diese internen Mechanismen des Kopierens und Teilens vor dem Benutzer verborgen, der lediglich Kopien von Bäumen sieht. Die Hauptsache hierbei ist, das Kopien billig sind, sowohl was die Zeit als auch den Speicherplatz angeht. Wenn Sie einen Zweig komplett im Projektarchiv anlegen (durch den Aufruf von **`svn copy`** ***`URL1 URL2`***), handelt es sich um eine schnelle Operation mit konstanter Zeitdauer. Erstellen Sie Zweige so oft Sie wollen.

Arbeiten mit Ihrem Zweig

Da Sie nun einen Zweig des Projektes erzeugt haben, können Sie eine neue Arbeitskopie auschecken, um ihn zu benutzen:

```
$ svn checkout http://svn.example.com/repos/calc/branches/my-calc-branch
A  my-calc-branch/Makefile
A  my-calc-branch/integer.c
A  my-calc-branch/button.c
Ausgecheckt, Revision 341.
$
```

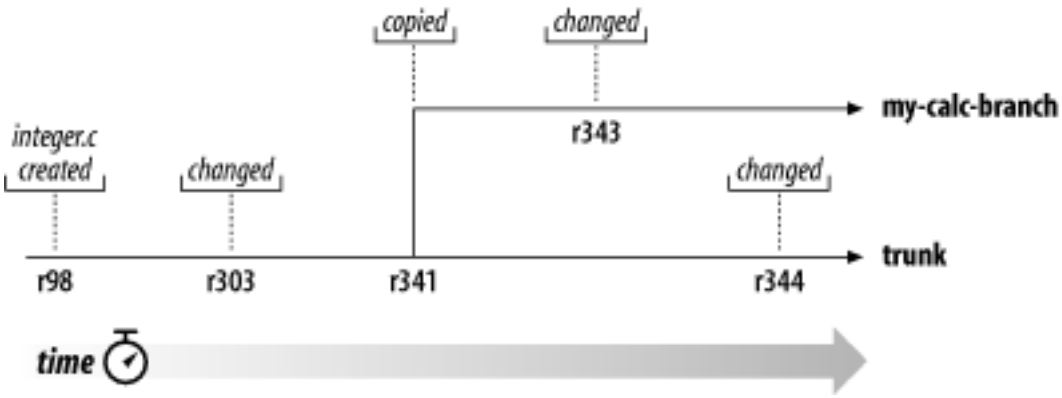
An dieser Arbeitskopie ist nichts besonders; sie spiegelt bloß ein anderes Verzeichnis im Projektarchiv wieder. Wenn Sie Änderungen übergeben, wird sie Sally jedoch nicht sehen, wenn sie aktualisiert, da sie eine Arbeitskopie von `/calc/trunk` hat. (Stellen Sie sicher, dass Sie „Zweige durchlaufen“ weiter unten in diesem Kapitel lesen: Der Befehl **svn switch** ist eine Alternative für die Bereitstellung einer Arbeitskopie eines Zweiges.)

Tun wir mal so, als ob eine Woche ins Land geht und die folgenden Übergaben stattfinden:

- Sie machen eine Änderung an `/calc/branches/my-calc-branch/button.c`, die die Revision 342 erzeugt.
- Sie machen eine Änderung an `/calc/branches/my-calc-branch/integer.c`, die die Revision 343 erzeugt.
- Sally macht eine Änderung an `/calc/trunk/integer.c`, die die Revision 344 erzeugt.

Nun finden zwei unabhängige Entwicklungslinien (siehe Abbildung 4.4, „Die Verzweigung der Geschichte einer Datei“) auf `integer.c` statt.

Abbildung 4.4. Die Verzweigung der Geschichte einer Datei



Es wird interessant, wenn Sie die Geschichte der Änderungen an Ihrer Kopie von `integer.c` betrachten:

```
$ pwd
/home/user/my-calc-branch

$ svn log -v integer.c
------------------------------------------------------------------------
r343 | user | 2002-11-07 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Geänderte Pfade:
   M /calc/branches/my-calc-branch/integer.c

* integer.c:  Wazjub gefrozzelt.

------------------------------------------------------------------------
r341 | user | 2002-11-03 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Geänderte Pfade:
```

```
   A /calc/branches/my-calc-branch (from /calc/trunk:340)

Privaten Zweig von /calc/trunk angelegt.

------------------------------------------------------------------------
r303 | sally | 2002-10-29 21:14:35 -0600 (Tue, 29 Oct 2002) | 2 lines
Geänderte Pfade:
   M /calc/trunk/integer.c

* integer.c:  Einen Docstring geändert.

------------------------------------------------------------------------
r98 | sally | 2002-02-22 15:35:29 -0600 (Fri, 22 Feb 2002) | 2 lines
Geänderte Pfade:
   A /calc/trunk/integer.c

* integer.c:  Diese Datei dem Projekt hinzugefügt.

------------------------------------------------------------------------
```

Beachten Sie, dass Subversion die Geschichte von `integer.c` auf Ihrem Zweig über die gesamte Zeit zurück verfolgt, und dabei sogar über den Punkt hinweg geht, an dem es kopiert wurde. Es zeigt die Erzeugung des Zweigs als ein Ereignis in der Geschichte, da `integer.c` implizit kopiert wurde, als alles andere in `/calc/trunk/` kopiert wurde. Sehen Sie nun, was passiert, wenn Sally den gleichen Befehl auf Ihre Arbeitskopie der Datei anwendet:

```
$ pwd
/home/sally/calc

$ svn log -v integer.c
------------------------------------------------------------------------
r344 | sally | 2002-11-07 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  Ein paar Rechtschreibfehler behoben.

------------------------------------------------------------------------
r303 | sally | 2002-10-29 21:14:35 -0600 (Tue, 29 Oct 2002) | 2 lines
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  Einen Docstring geändert.

------------------------------------------------------------------------
r98 | sally | 2002-02-22 15:35:29 -0600 (Fri, 22 Feb 2002) | 2 lines
Changed paths:
   A /calc/trunk/integer.c

* integer.c:  Diese Datei dem Projekt hinzugefügt.

------------------------------------------------------------------------
```

Sally sieht ihre eigene Änderung in Revision 344, aber nicht die Änderung, die Sie in Revision 343 gemacht haben. Was Subversion angeht, hatten diese beiden Übergaben Auswirkungen auf unterschiedliche Dateien an unterschiedlichen Stellen im Projektarchiv. *Glwichwohl* zeigt Subversion, dass die beiden Dateien einen Teil der Geschichte gemeinsam haben. Bevor die Kopie des Zweiges in Revision 341 gemacht wurde, waren die Dateien dieselbe Datei. Deshalb sehen sowohl Sie als auch Sally die Änderungen, die in den Revisionen 303 und 98 gemacht wurden.

Die Schlüsselkonzepte des Verzweigens

Sie sollten sich zwei Lektionen aus diesem Abschnitt merken. Erstens besitzt Subversion kein internes Konzept für einen Zweig – es weiß lediglich, wie Kopien angelegt werden. Wenn Sie ein Verzeichnis kopieren, ist das entstehende Verzeichnis bloß ein „Zweig“, weil *Sie* ihm diese Bedeutung geben. Sie mögen über das Verzeichnis anders denken oder es anders behandeln, doch für Subversion ist es einfach ein gewöhnliches Verzeichnis, das nebenbei mit einigen zusätzlichen historischen Informationen ausgestattet ist.

Zweitens bestehen die Zweige von Subversion, bedingt durch den Kopiermechanismus, als *normale Dateisystemverzeichnisse* im Projektarchiv. Das ist ein Unterschied zu anderen Versionskontrollsystemen, bei denen Zweige typischerweise definiert werden, indem auf einer eigenen Ebene den Dateisammlungen „Etiketten“ hinzugefügt werden. Der Ort Ihres Zweig-Verzeichnisses spielt für Subversion keine Rolle. Die meisten Teams folgen der Konvention, alle Zweige in einem Verzeichnis namens `/branches` abzulegen, jedoch steht es Ihnen frei, eine Vorgehensweise nach Ihren Wünschen zu erfinden.

Grundlegendes Zusammenführen

Nun arbeiten Sie und Sally auf parallelen Zweigen des Projektes: Sie arbeiten auf einem privaten Zweig, und Sally arbeitet auf dem *Stamm* oder dem Hauptzweig der Entwicklung.

Bei Projekten mit einer großen Zahl von Mitarbeitern besitzen die meisten gewöhnlich Arbeitskopien vom Stamm. Sobald jemand eine langwierige Änderung machen muss, die wahrscheinlich den Stamm stören würde, ist es Standard, einen Zweig zu erzeugen und die Änderungen bis zum Abschluss der Arbeiten nach dorthin zu übergeben.

Die gute Nachricht ist also, dass Sie und Sally sich nicht in die Quere kommen. Die schlechte Nachricht ist, dass es sehr leicht ist, *zu* weit auseinander zu treiben. Erinnern Sie sich, dass eins der Probleme bei der Strategie „sich in ein Loch verkriechen“ darin bestand, dass es zu dem Zeitpunkt, an dem Sie mit dem Zweig fertig sind, fast unmöglich sein kann, Ihre Änderungen ohne eine riesige Zahl an Konflikten auf den Stamm zurückzuführen.

Stattdessen könnten Sie und Sally fortfahren, während der Arbeit Änderungen gemeinsam zu verwenden. Es liegt an Ihnen, zu entscheiden, welche Änderungen teilenswert sind; Subversion bietet Ihnen die Fähigkeit, Änderungen selektiv zwischen Zweigen zu „kopieren“. Und wenn Sie mit Ihrem Zweig vollständig fertig sind, kann die gesamte Menge Ihrer Änderungen vom Zweig auf den Stamm zurück kopiert werden. In der Terminologie von Subversion heißt der allgemeine Vorgang, Änderungen von einem Zweig auf einen anderen zu übertragen *Zusammenführen* (Merging) und wird durch verschiedene Aufrufe des Befehls **svn merge** durchgeführt.

In den folgenden Beispielen gehen wir davon aus, dass sowohl auf Ihrem Subversion-Client als auch auf dem Server Subversion 1.5 (oder neuer) läuft. Falls einer von beiden älter als Version 1.5 ist, wird es komplizierter: Das System wird Änderungen nicht automatisch mitverfolgen, so dass Sie schmerzhafte manuelle Methoden anwenden müssen, um ähnliche Resultate zu erzielen. Dass heißt, dass Sie stets die detaillierte Syntax beim Zusammenführen verwenden müssen, um bestimmte Revisionsintervalle zu übertragen (siehe „Merge-Syntax: Die vollständige Enthüllung“ weiter unten in diesem Kapitel), und besonders sorgfältig verfolgen müssen, was bereits zusammengeführt ist und was nicht. Aus diesem Grund empfehlen wir Ihnen *dringend*, sicherzustellen, dass Ihr Client und Server mindestens die Version 1.5 haben.

Änderungsmengen

Bevor wir weitermachen, sollten wir Sie warnen, dass Sie auf den kommenden Seiten viele Erörterungen zum Thema „Änderungen“ erwarten. Viele mit Versionskontrollsystemen erfahrene Leute benutzen die Begriffe „Änderung“ und „Änderungsmenge“ (Changeset) austauschbar, so dass wir klären sollten, was Subversion unter einer *Änderungsmenge* versteht.

Jeder scheint eine etwas unterschiedliche Definition für den Begriff Änderungsmenge zu haben oder zumindest eine unterschiedliche Erwartung darüber, was es für ein Versionskontrollsystem bedeutet, so etwas zu besitzen. Für unsere Zwecke reicht es aus, zu sagen, dass eine Änderungsmenge lediglich eine Sammlung von Änderungen mit einem eindeutigen Namen ist. Die Änderungen können aus der Bearbeitung an Textdateien, Modifizierungen an der Baumstruktur oder Justierungen an Metadaten bestehen. In einfachen Worten ist eine Änderungsmenge einfach ein Patch mit einem Namen, auf den Sie sich beziehen können.

In Subversion bezeichnet eine globale Revisionsnummer *N* einen Baum im Projektarchiv: Sie beschreibt das Aussehen des Projektarchivs nach der *N*-ten Übergabe. Sie ist auch der Name einer impliziten Änderungsmenge: Wenn Sie den Baum *N* mit dem Baum *N*-1 vergleichen, können Sie genau den Patch ableiten, der übergeben wurde. Daher ist es einfach, sich Revision *N* nicht nur als Baum sondern auch als Änderungsmenge vorzustellen. Falls Sie ein Fehlerverwaltungssystem verwenden, können Sie die Revisionsnummern benutzen, um auf bestimmte Patches zu verweisen, die Fehler beheben – zum Beispiel: „Dieser Fehler wurde durch r9238 behoben“. Dann kann jemand **`svn log -r 9238`** aufrufen, um den Protokolleintrag zu genau der

Änderungsmenge zu lesen, die den Fehler behoben hat, und sich mit **`svn diff -c 9238`** den eigentlichen Patch ansehen. Und auch (wie Sie bald sehen werden) der Subversion Befehl **svn merge** kann Revisionsnummern verwenden. Sie können bestimmte Änderungsmengen von einem Zweig mit einem anderen zusammenführen, indem sie in den Argumenten zum entsprechenden Kommando benannt werden: Die Übergabe von **`-c 9238`** an **svn merge** würde das Änderungsmenge r9238 mit Ihrer Arbeitskopie zusammenführen.

Einen Zweig synchron halten

Machen wir mit unserem Beispiel weiter und nehmen an, dass eine Woche vergangen ist seitdem Sie begonnen haben, auf Ihrem privaten Zweig zu arbeiten. Ihre Arbeit ist noch nicht beendet, jedoch wissen Sie, dass gleichzeitig andere Leute in Ihrem Team weiterhin wichtige Änderungen im `/trunk` des Projektes gemacht haben. Es ist in Ihrem Interesse, diese Änderungen in Ihren Zweig zu übernehmen, um sicherzustellen, dass sie sich gut mit Ihren Änderungen vertragen.

Ihren Zweig regelmäßig mit der Hauptentwicklungslinie zu synchronisieren hilft, „überraschende“ Konflikte zu vermeiden, wenn es an der Zeit ist, Ihre Änderungen zurück auf den Stamm zu bringen.

Subversion kennt die Geschichte Ihres Zweigs und weiß, wann Sie ihn vom Stamm abgezweigt haben. Um die letzten, aktuellsten Änderungen vom Stamm auf Ihren Zweig zu bringen, sollten Sie zunächst sicherstellen, dass die Arbeitskopie des Zweigs „sauber“ ist – dass sie keine lokalen Änderungen hat, die durch **svn status** angezeigt werden. Dann rufen Sie einfach die folgenden Befehle auf:

```
$ pwd
/home/user/my-calc-branch

$ svn merge ^/calc/trunk
--- Zusammenführen von r345 bis r356 in ».«:
U    button.c
U    integer.c
$
```

Diese einfache Syntax – **`svn merge`** ***`URL`*** – fordert Subversion auf, alle neuen Änderungen von dem URL mit dem aktuellen Arbeitsverzeichnis (welches typischerweise das Wurzelverzeichnis Ihrer Arbeitskopie ist) zusammenzuführen. Beachten Sie auch, dass wir die Syntax mit dem Zirkumflex (^) verwenden[2], um nicht den vollständigen `/trunk`-URL tippen zu müssen.

Nach dem Ausführen des vorangegangenen Beispiels enthält Ihre Arbeitskopie nun neue lokale Änderungen, die Nachbildungen all der Änderungen auf dem Stamm seit der Erstellung Ihres Zweiges sind:

```
$ svn status
 M      .
M       button.c
M       integer.c
$
```

Zu diesem Zeitpunkt ist es weise, sich die Änderungen mithilfe von **svn diff** sorgfältig anzusehen, und anschließend die Software von Ihrem Zweig zu bauen und zu testen. Beachten Sie, dass auch das aktuelle Arbeitsverzeichnis („`.`“) verändert wurde; **svn diff** zeigt an, dass seine Eigenschaft `svn:mergeinfo` entweder angelegt oder modifiziert wurde. Das ist ein wichtiges Metadatum in Zusammenhang mit Zusammenführungen, das Sie *nicht* anfassen sollten, da es von künftigen **svn merge**-Befehlen benötigt wird. (Wir werden später in diesem Kapitel mehr über diese Metadaten erfahren.)

Nach der Übernahme kann es möglich sein, dass Sie noch einige Konflikte auflösen müssen (wie bei **svn update**) oder möglicherweise noch einige kleinere Bearbeitungen durchzuführen haben, damit alles wieder funktioniert. (Denken Sie daran, dass die Abwesenheit *syntaktischer* Konflikte nicht bedeutet, dass keine *semantischen* Konflikte vorhanden sind!) Falls ernsthafte Probleme auftauchen, können Sie jederzeit die lokalen Änderungen mit **`svn revert . -R`** wieder rückgängig

[2]Diese wurde in svn 1.6 eingeführt,

machen und eine lange „was geht hier eigentlich vor“-Unterredung mit Ihren Mitarbeitern führen. Falls jedoch alles gut aussieht, können Sie die Änderungen an das Projektarchiv übergeben:

```
$ svn commit -m "Die letzten Änderungen von trunk mit my-calc-branch
zusammengeführt."
Sende          .
Sende          button.c
Sende          integer.c
Übertrage Daten ..
Revision 357 übertragen.
```

An dieser Stelle ist Ihr Zweig „synchron“ mit dem Stamm, und Sie können sich ruhig zurücklehnen in der Gewissheit, dass Sie sich nicht zu weit von der Arbeit aller anderen entfernen, während Sie isoliert weiterarbeiten.

Warum stattdessen keine Patches verwenden?

Eine Frage könnte Ihnen durch den Kopf gehen, besonders, falls Sie ein Unix-Benutzer sind: Warum soll ich überhaupt **svn merge** verwenden? Warum kann ich dieselbe Aufgabe nicht mit dem Betriebssystembefehl **patch** lösen? Zum Beispiel:

```
$ cd my-calc-branch
$ svn diff -r 341:HEAD ^/calc/trunk > patchfile
$ patch -p0  < patchfile
Patching file integer.c using Plan A...
Hunk #1 succeeded at 147.
Hunk #2 succeeded at 164.
Hunk #3 succeeded at 241.
Hunk #4 succeeded at 249.
done
$
```

Bei diesem speziellen Beispiel gibt es wahrhaftig keinen großen Unterschied. Allerdings hat **svn merge** besondere Fähigkeiten, die über die des Programms **patch** hinaus gehen. Das von **patch** verwendete Dateiformat ist sehr eingeschränkt; es kann lediglich Dateiinhalte verändern. Es besteht keine Möglichkeit, Änderungen an *Bäumen*, etwa das Hinzufügen, Entfernen oder Umbenennen von Dateien und Verzeichnissen abzubilden. Desweiteren bemerkt das Programm **patch** keine Änderungen an Eigenschaften. Falls Sallys Änderung etwa ein neues Verzeichnis hinzugefügt hätte, wäre es in der Ausgabe von **svn diff** überhaupt nicht erwähnt worden. **svn diff** gibt nur das eingeschränkte patch-Format aus, so dass es einige der Konzepte gar nicht wiedergeben kann.

Der Befehl **svn merge** jedoch kann Änderungen an der Baumstruktur und an Eigenschaften erfassen, indem sie direkt auf Ihre Arbeitskopie angewendet werden. Noch wichtiger ist, dass dieser Befehl alle Änderungen festhält, die auf Ihren Zweig angewendet wurden, so dass Subversion genau Bescheid weiß, welche Änderungen an welcher Stelle vorhanden sind (siehe „Zusammenführungsinformation und Vorschauen“). Dies ist eine kritische Fähigkeit, die die Verwaltung von Zweigen brauchbar macht; ohne sie müssten Benutzer sich manuelle Aufzeichnungen darüber machen, welche Änderungsmengen zusammengeführt worden sind und welche noch nicht.

Nehmen wir an, noch eine Woche sei ins Land gegangen. Sie haben weitere Änderungen an Ihren Zweig übergeben, und Ihre Kollegen haben damit weitergemacht, den Stamm zu verbessern. Nun möchten Sie mal wieder die letzten Änderungen vom Stamm mit Ihrem Zweig abgleichen, damit Sie wieder synchron sind. Starten Sie einfach noch einmal den **svn merge**-Befehl!

```
$ svn merge ^/calc/trunk
--- Zusammenführen von r357 bis r380 in ».«:
U    integer.c
U    Makefile
A    README
```

```
$
```

Subversion weiß, welche Änderungen Sie bereits mit Ihrem Zweig abgeglichen haben, so dass es sorgfältig nur die Änderungen berücksichtigt, die Sie noch nicht haben. Einmal mehr müssen Sie bauen, testen und die lokalen Änderungen an Ihren Zweig mit **svn commit** übergeben.

Reintegration eines Zweigs

Was passiert jedoch, wenn Sie schließlich Ihre Arbeit abgeschlossen haben? Ihre neue Funktion ist fertig, und Sie sind bereit, die Änderungen von Ihrem Zweig zurück auf den Stamm zu überführen (so dass Ihr Team die Früchte Ihrer Arbeit genießen kann). Die Vorgehensweise ist einfach. Zunächst synchronisieren Sie Ihren Zweig noch einmal mit dem Stamm, wie Sie es bisher gemacht haben:

```
$ svn merge ^/calc/trunk
--- Zusammenführen von r381 bis r385 in ».«:
U    button.c
U    README

$ # bauen, testen, ...

$ svn commit -m "Letzte Zusammenführung der Änderungen von trunk changes in
my-calc-branch."
Sende          .
Sende          button.c
Sende          README
Übertrage Daten ..
Revision 390 übertragen.
```

Nun verwenden Sie **svn merge** mit der Option `--reintegrate` , um Ihre Änderungen vom Zweig zurück auf den Stamm zu überführen. Sie benötigen eine Arbeitskopie von `/trunk`. Sie bekommen sie entweder durch **svn checkout**, indem Sie von irgendwo auf Ihrer Platte eine alte Arbeitskopie vom Stamm hervorholen, oder den Befehl **svn switch** (siehe „Zweige durchlaufen“) verwenden. Ihre Arbeitskopie darf keine lokalen Änderungen beinhalten oder aus gemischten Revisionen bestehen (siehe „Arbeitskopien mit gemischten Revisionen“). Obwohl es sich dabei normalerweise um die bewährten Vorgehensweisen beim Zusammenführen handelt, sind sie bei der Verwendung der Option `--reintegrate` zwingend notwendig.

Sobald Sie eine saubere Arbeitskopie des Stamms haben, sind Sie bereit, Ihren Zweig damit zusammenzuführen:

```
$ pwd
/home/user/calc-trunk

$ svn update  # (stellen Sie sicher, dass die Arbeitskopie aktuell ist)
Revision 390.

$ svn merge --reintegrate ^/calc/branches/my-calc-branch
-- Zusammenführen der Unterschiede zwischen Projektarchiv-URLs in ».«:
U    button.c
U    integer.c
U    Makefile
 U   .

$ # bauen, testen, überprüfen, ...

$ svn commit -m "my-calc-branch mit Stamm zusammenführen!"
Sende          .
Sende          button.c
Sende          integer.c
```

```
Sende          Makefile
Übertrage Daten ..
Revision 391 übertragen.
```

Gratulation! Ihr Zweig ist nun zurück in die Hauptentwicklungslinie überführt worden. Beachten Sie, dass dieses Mal die Option `--reintegrate` verwendet wurde. Diese Option ist kritisch, wenn Änderungen von einem Zweig in die ursprüngliche Entwicklungslinie reintegriert werden – vergessen Sie sie nicht! Sie wird benötigt, da diese Art der „Rücküberführung“ etwas anderes ist, als was Sie bisher gemacht haben. Vorher haben wir **svn merge** aufgefordert, die „nächste Änderungsmenge“ von einer Entwicklungslinie (dem Stamm) zu holen und sie mit einer anderen (Ihrem Zweig) abzugleichen. Das ist recht überschaubar, und Subversion weiß jedesmal, wo es wieder ansetzen soll. Bei unseren vorangehenden Beispielen können Sie sehen, dass es erst die Intervalle 345:356 vom Stamm auf den Zweig überführte; später fuhr es mit dem nächsten verfügbaren aufeinanderfolgenden Intervall 356:380 fort. Wenn Sie die letzte Synchronisierung machen, wird es das Intervall 380:385 zusammenführen.

Wenn Sie jedoch den Zweig auf den Stamm zurückführen, sehen die dem zugrundeliegenden Berechnungen ganz anders aus. Ihr Zweig ist nun ein Mischmasch aus abgeglichenen Änderungen vom Stamm und privaten Änderungen auf dem Zweig, so dass es kein einfaches, aufeinanderfolgendes Intervall mit Revisionen zum Herüberkopieren gibt. Indem Sie die Option `--reintegrate` angeben, fordern Sie Subversion auf, sorgfältig *nur* die Änderungen von Ihrem Zweig zu replizieren. (Und tatsächlich macht es das so, dass es die letzte Version auf dem Stamm mit der letzten Version auf dem Zweig vergleicht: Der Unterschied macht genau die Änderung auf dem Zweig aus!)

Beachten Sie, dass die Option `--reintegrate` im Gegensatz zur allgemeineren Natur der meisten Optionen von Subversion-Unterbefehlen doch recht spezialisiert ist. Sie unterstützt den oben beschriebenen Anwendungsfall, hat daneben jedoch wenig Verwendungsmöglichkeiten. Wegen dieses engen Einsatzgebietes sowie der Tatsache, das eine aktualisierte Arbeitskopie ohne gemischte Revisionen erforderlich ist, wird diese Option nicht mit den meisten anderen Optionen von **svn merge** funktionieren. options. Sie werden einen Fehler bekommen, falls Sie andere nicht-globale Optionen verwenden, als die folgenden: `--accept`, `--dry-run`, `--diff3-cmd`, `--extensions` oder `--quiet`.

Nachdem nun Ihr privater Zweig mit dem Stamm zusammengeführt wurde, können Sie ihn aus dem Projektarchiv löschen:

```
$ svn delete ^/calc/branches/my-calc-branch \
      -m "my-calc-branch entfernt, auf Stamm zurüchgeführt in r391."
Revision 392 übertragen.
```

Aber halt! Ist die Geschichte des Zweigs nicht wertvoll? Was, wenn jemand sich eines Tages die Evolution Ihrer Funktion ansehen möchte und hierfür auf die Änderungen des Zweiges schauen möchte? Keine Sorge! Denken Sie daran, dass, obwohl Ihr Zweig nicht mehr im Verzeichnis `/branches` sichtbar ist, seine Existenz gleichwohl ein unveränderbarer Teil der Geschichte des Projektarchivs ist. Ein einfacher Befehl **svn log** auf dem `/branches` URL wird die gesamte Geschichte des Zweiges anzeigen. Ihr Zweig kann eines Tages sogar wiederbelebt werden, sollten Sie dieses wünschen (siehe „Zurückholen gelöschter Objekte“).

Sobald eine Zusammenführung mit `--reintegrate` vom Zweig auf den Stamm durchgeführt wurde, kann der Zweig nicht mehr für weitere Arbeiten verwendet werden. Er kann weder Änderungen vom Stamm korrekt absorbieren, noch kann er ordentlich auf den Stamm zurückintegriert werden. Aus diesem Grund sollten Sie ihn zerstören und erneut aus dem Stamm erzeugen, wenn Sie weiter auf dem Zweig arbeiten wollen:

```
$ svn delete http://svn.example.com/repos/calc/branches/my-calc-branch \
      -m "my-calc-branch löschen, zurückgeführt auf Stamm."
Revision 392 übertragen.

$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/my-calc-branch
      -m "my-calc-branch erneut von trunk@HEAD abgezweigt."
Revision 393 übertragen.
```

Es gibt eine weitere Möglichkeit, wie der Zweig nach der Zurückführung wiederverwendbar gemacht werden kann, ohne ihn zu löschen. Siehe „Einen reintegrierten Zweig am Leben erhalten“.

Zusammenführungsinformation und Vorschauen

Der grundsätzliche Mechanismus, den Subversion verwendet, um Änderungsmengen zu verfolgen – d.h. welche Änderungen auf welchen Zweig übertragen worden sind – besteht aus der Datenspeicherung in versionierten Eigenschaften. Namentlilch werden Daten die das Zusammenführen betreffen in der Eigenschaft `svn:mergeinfo` festgehalten, die mit Dateien und Verzeichnissen verknüpft ist. (Falls Sie mit Subversion-Eigenschaften nicht vertraut sind, siehe „Eigenschaften“.)

Sie können sich die Eigenschaft ansehen, wie jede andere auch:

```
$ cd my-calc-branch
$ svn propget svn:mergeinfo .
/trunk:341-390
$
```

Obwohl es möglich ist, `svn:mergeinfo` zu ändern wie jede andere versionierte Eigenschaft, raten wir dringend davor ab, es sei denn, Sie wissen *wirklich*, was Sie da machen.

Die Eigenschaft `svn:mergeinfo` wird automatisch von Subversion gepflegt, sobald Sie den Befehl **svn merge** ausführen. Ihr Wert gibt Aufschluss darüber, welche Änderungen an einem gegebenen Pfad mit dem in Frage kommenden Verzeichnis abgeglichen wurden. Im vorangegangenen Beispiel ist `/trunk` der Ursprungspfad der zusammengeführten Änderungen und `/branches/my-calc-branch` das Zielverzeichnis.

Subversion stellt auch den Unterbefehl, **svn mergeinfo** zur Verfügung, der dabei hilfreich sein kann, nicht nur die in ein Verzeichnis eingeflossenen Änderungsmengen anzuzeigen, sondern auch, welche Änderungsmengen noch für einen Abgleich bereit stehen. Das ergibt eine Art Vorschau der nächsten Änderungsmengen, die **svn merge** auf Ihren Zweig abgleichen wird.

```
$ cd my-calc-branch

# Welche Änderungen wurden bereits vom Stamm auf den Zweig abgeglichen?
$ svn mergeinfo ^/calc/trunk
r341
r342
r343
…
r388
r389
r390

# Welche Änderungen kommen für einen Abgleich vom Stamm auf den Zweig noch in Frage?
$ svn mergeinfo ^/calc/trunk --show-revs eligible
r391
r392
r393
r394
r395
$
```

Der Befehl **svn mergeinfo** erwartet einen „Quell“-URL (woher die Änderungen kommen würden) und einen optionalen „Ziel“-URL (wohin die Änderungen abgeglichen würden). Falls kein Ziel-URL angegeben ist, wird das aktuelle Arbeitsverzeichnis als Ziel angenommen. Da wir im vorangegangenen Beispiel unser dem Zweig entsprechenden Arbeitsverzeichnis abfragen, geht der Befehl davon aus, dass wir daran interessiert sind, Änderungen vom angegebenen Stamm-URL für `/branches/mybranch` zu erhalten.

Eine andere Methode, eine genauere Vorschau auf einen Abgleich zu bekommen, ist die Verwendung der Option `--dry-run`:

```
$ svn merge ^/calc/trunk --dry-run
U    integer.c

$ svn status
#  es wird nichts ausgegeben, die Arbeitskopie ist unverändert
```

Die Option `--dry-run` macht tatsächlich überhaupt keine lokalen Änderungen an der Arbeitskopie. Sie zeigt nur Status-Codes, die ausgegeben *würden*, wenn ein echter Abgleich stattfände. Sie ist nützlich, um eine Vorschau für einen möglichen Abgleich auf „hoher Ebene“ zu erhalten, falls **svn diff** zu detailliert wäre.

Nach dem Durchführen eines Abgleichs, aber vor der Übergabe des Ergebnisses, können Sie **`svn diff --depth=empty`** ***`/pfad/zum/abgleichs/ziel`*** verwenden, um nur die Änderungen am unmittelbaren Ziel des Abgleichs zu sehen. Falls das Ziel ein Verzeichnis war, werden nur Unterschiede von Eigenschaften angezeigt. Das ist eine praktische Methode, um sich die Änderungen an der Eigenschaft `svn:mergeinfo` anzusehen, die dort durch den Abgleich vermerkt wurden, und die Sie daran erinnern, was Sie eben abgeglichen haben.

Natürlich ist die beste Methode, eine Vorschau eines Abgleichs zu erhalten, ihn zu machen! Denken Sie daran, dass der Aufruf von **svn merge** an sich nichts Riskantes ist (es sei denn, sie haben lokale Änderungen an Ihrer Arbeitskopie gemacht – aber wir haben bereits betont, dass Sie in eine derartige Umgebung nicht abgleichen sollten). Falls Ihnen das Ergebnis des Abgleichs nicht gefallen sollte, rufen Sie einfach **`svn revert . -R`** auf, um die Änderungen an Ihrer Arbeitskopie rückgängig zu machen, und versuchen Sie den Befehl erneut mit unterschiedlichen Optionen. Der Abgleich ist solange nicht endgültig, bis Sie mit **svn commit** das Ergebnis übergeben.

Während es vollkommen in Ordnung ist, durch wiederholte Aufrufe von **svn merge** und **svn revert** mit Abgleichen zu experimentieren, könnte es allerdings sein, dass Sie über einige lästige (aber leicht zu umgehende) Fallstricke stolpern. Wenn zum Beispiel durch den Abgleich eine neue Datei hinzugefügt wird (d.h., sie wird zum Hinzufügen markiert), so wird **svn revert** sie nicht wirklich entfernen; es entfernt lediglich die Markierung zum Hinzufügen. Was übrig bleibt, ist eine unversionierte Datei. Wenn Sie dann den Abgleich erneut versuchen, könnten Sie einen Konflikt bekommen, weil die unversionierte Datei „im Weg steht“. Die Lösung? Nach dem Rückgängigmachen sollten Sie die Arbeitskopie aufräumen und unversionierte Dateien und Verzeichnisse entfernen. Die Ausgabe von **svn status** sollte so sauber wie möglich sein und idealerweise gar nichts anzeigen.

Änderungen rückgängig machen

Sehr häufig wird **svn merge** verwendet, um eine Änderung rückgängig zu machen, die bereits an das Projektarchiv übergeben worden war. Nehmen wir einmal an, Sie arbeiten fröhlich in einer Arbeitskopie von `/calc/trunk` und entdecken, dass die damalige Änderung an `integer.c` in Revision 303 völlig falsch war. Sie hätte nie übergeben werden sollen. Sie können **svn merge** verwenden, um die Änderung in Ihrer Arbeitskopie „zurückzunehmen“, und dann die lokale Änderung an das Projektarchiv übergeben. Alles, was Sie hierfür tun müssen, ist, eine *umgekehrte* Differenz anzugeben. (Sie machen das durch die Angabe von `--revision 303:302` oder durch das äquivalente `--change -303`.)

```
$ svn merge -c -303 ^/calc/trunk
--- Reverse-merging r303 into 'integer.c':
-- Rückwärtiges Zusammenführen von r303 in »integer.c«:
U    integer.c

$ svn status
 M      .
M       integer.c
```

```
$ svn diff
…
# überprüfen, ob die Änderung entfernt wurde
…

$ svn commit -m "Änderung aus in r303 rückgängig machen."
Sende          integer.c
Übertrage Daten .
Revision 350 übertragen.
```

Wie wir früher bereits erwähnten, kann man eine Projektarchiv-Version als eine bestimmte Änderungsmenge betrachten. Bei Verwendung der Option `-r` wird **svn merge** aufgefordert, eine Änderungsmenge oder ein ganzes Intervall von Änderungsmengen auf Ihre Arbeitskopie anzuwenden. In unserem Fall, bei dem wir eine Änderung zurücknehmen, fordern wir **svn merge** auf, die Änderungsmenge #303 *rückwärts* auf unsere Arbeitskopie anzuwenden.

Merken Sie sich, dass ein solches Rückgängigmachen wie jeder andere **svn merge**-Vorgang ist, so dass Sie **svn status** und **svn diff** benutzen sollten, um sicherzustellen, dass Ihre Arbeit in dem Zustand ist, den Sie haben möchten, und verwenden Sie anschließend **svn commit**, um die endgültige Version in das Projektarchiv zu bringen. Nach der Übergabe wird sich diese bestimmte Änderungsmenge nicht mehr in der `HEAD`-Revision wiederfinden.

Nun denken Sie vielleicht: Gut, aber das hat doch nicht wirklich die Übergabe rückgängig gemacht, oder? Die Änderung besteht immer noch in Revision 303. Falls jemand eine Version des Projektes `calc` zwischen den Revisionen 303 und 349 auscheckt, wird doch trotzdem die fehlerhafte Änderung sichtbar, oder nicht?

Ja, das stimmt. Wenn wir davon sprechen, eine Änderung zu „entfernen“, sprechen wir eigentlich darüber, sie aus der `HEAD`-Revision zu entfernen. Die ursprüngliche Änderung besteht immer noch in der Geschichte des Projektarchivs. Für die meisten Situationen ist das ausreichend. Die meisten Leute sind sowieso nur am `HEAD` eines Projektes interessiert. Es gibt jedoch Spezialfälle, in denen Sie wirklich alle Beweise der Übergabe vernichten möchten. (Vielleicht hat jemand ein vertrauliches Dokument in das Projektarchiv übergeben.) Das ist leider nicht so einfach, da Subversion absichtlich so konstruiert wurde, dass es niemals Informationen verliert. Revisionen sind unveränderliche Bäume, die aufeinander aufbauen. Die Beseitigung einer Revision aus der Geschichte würde einen Dominoeffekt auslösen, Chaos in allen nachfolgenden Revisionen anrichten und möglicherweise alle Arbeitskopien ungültig machen.[3]

Zurückholen gelöschter Objekte

Das Tolle an Versionskontrollsystemen ist, dass Informationen nie verlorengehen. Selbst wenn Sie eine Datei oder ein Verzeichnis löschen, ist es zwar nicht mehr in der `HEAD`-Revision vorhanden, jedoch noch in früheren Revisionen. Eine der häufigsten Fragen neuer Benutzer ist: „Wie bekomme ich meine alte Datei oder mein altes Verzeichnis zurück?“

Der erste Schritt ist es, genau zu definieren *welches* Objekt Sie zurückholen möchten. Hier ist eine nützliche Metapher: Sie können sich vorstellen, dass jedes Objekt im Projektarchiv in einem zweidimensionalen Koordinatensystem befindet. Die erste Koordinate ist ein bestimmter Revisionsbaum und die zweite Koordinate ist ein Pfad innerhalb dieses Baumes. So kann jede Version Ihrer Datei oder Ihres Verzeichnisses durch ein bestimmtes Koordinatenpaar definiert werden. (Erinnern Sie sich an die Syntax einer „Peg-Revision“ – foo.c@224 – die in „Peg- und operative Revisionen“ erwähnt wurde.)

Zunächst sollten Sie **svn log** benutzen, um das exakte Koordinatenpaar zu ermitteln, das Sie zurückholen wollen. Eine gute Strategie ist es, **`svn log --verbose`** in einem Verzeichnis aufzurufen, in dem das gelöschte Objekt einmal enthalten war. Die Option `--verbose` (`-v`) gibt eine Liste aller geänderten Objekte in jeder Revision aus; Sie müssen nur noch die Revision finden, in der Sie die Datei oder das Verzeichnis gelöscht haben. Sie können das visuell tun oder ein Werkzeug zur Untersuchung der Protokollausgaben einsetzen (mit **grep** oder vielleicht durch eine inkrementelle Suche in einem Editor).

```
$ cd parent-dir
$ svn log -v
…
------------------------------------------------------------------------
r808 | joe | 2003-12-26 14:29:40 -0600 (Fri, 26 Dec 2003) | 3 lines
```

[3] Allerdings gibt es im Subversion-Projekt Pläne, eines Tages einen Befehl zu implementieren, der die Aufgabe erledigen würde, Informationen dauerhaft zu löschen. Bis dahin, siehe „svndumpfilter“ für einen möglichen Notbehelf.

```
Geänderte Pfade:
   D /calc/trunk/real.c
   M /calc/trunk/integer.c

Schnelle Funktionen zur Fourier-Transformation zu integer.c hinzugefügt.
real.c gelöscht, da Code jetzt in double.c.
…
```

In diesem Beispiel nehmen wir an, dass Sie nach der gelöschten Datei `real.c` suchen. Beim Durchsehen der Protokolle des Elternverzeichnisses haben Sie entdeckt, dass diese Datei in Revision 808 gelöscht wurde. Daher war die letzte Revision in der die Datei noch vorhanden war die unmittelbare Vorgänger-Revision. Die Schlussfolgerung: Sie möchten den Pfad `/calc/trunk/real.c` aus Revision 807 zurückholen.

Das war der schwierige Teil – die Nachforschung. Nun, da Sie wissen, was Sie wiederherstellen wollen, haben Sie die Wahl zwischen zwei verschiedenen Methoden.

Die eine Option ist, **svn merge** zu verwenden, um Revision 808 „rückwärts“ anzuwenden. (Wir haben bereits in „Änderungen rückgängig machen“ besprochen, wie Änderungen rückgängig gemacht werden.) Das hätte den Effekt, `real.c` als lokale Änderung erneut hinzuzufügen. Die Datei würde zum Hinzufügen ins Projektarchiv markiert, und nach der Übergabe wäre die Datei wieder in `HEAD` vorhanden.

In diesem besonderen Beispiel ist das aber wahrscheinlich nicht die beste Strategie. Die Rückwärts-Anwendung von Revision 808 würde nicht nur `real.c` zum Hinzufügen markieren, sondern, wie aus den Protokollmeldungen hervorgeht, dass ebenso bestimmte Änderungen an `integer.c` zurücknehmen, was Sie aber nicht wollen. Sie können sicherlich Revision 808 rückwärts anwenden und dann mit **svn revert** die lokalen Änderungen an `integer.c` zurücknehmen; allerdings ist diese Technik nicht sehr effektiv. Was wäre, wenn 90 Dateien in Revision 808 geändert worden wären?

Eine zweite, zielorientiertere, Strategie ist es, den Befehl **svn merge** überhaupt nicht zu verwenden, sondern stattdessen **svn copy**. Kopieren Sie einfach das exakte „Koordinatenpaar“ aus Revision und Pfad vom Projektarchiv in Ihre Arbeitskopie:

```
$ svn copy http://svn.example.com/repos/calc/trunk/real.c@807 ./real.c

$ svn status
A  +    real.c

$ svn commit -m "real.c aus revision 807 wiederhergestellt, /calc/trunk/real.c."
Hinzufügen     real.c
Übertrage Daten .
Revision 1390 übertragen.
```

Das Plus-Zeichen in der Statusausgabe zeigt an, dass das Objekt nicht bloß zu Hinzufügen vorgemerkt ist, sondern zum Hinzufügen „mit Geschichte“. Subversion merkt sich, woher es kopiert wurde. Künftig wird beim Anwenden von **svn log** auf diese Datei die gesamte Geschichte, über das Zurückholen hinweg, inklusive der Geschichte vor Revision 807 durchlaufen. In anderen Worten, dieses neue `real.c` ist nicht wirklich neu; es ist ein direkter Nachfahre der ursprünglichen, gelöschten Datei. Dies ist normalerweise eine gute und nützliche Sache. Falls Sie jedoch die Datei *ohne* geschichtliche Verbindung zur alten Datei zurückholen wollen, funktioniert diese Technik ebensogut:

```
$ svn cat ^/calc/trunk/real.c@807 > ./real.c

$ svn add real.c
A         real.c

$ svn commit -m "real.c aus Revision 807 wiederhergestellt."
Hinzufügen     real.c
Übertrage Daten .
Revision 1390 übertragen.
```

Obwohl unser Beispiel zeigt, wie eine Datei zurückgeholt wird, sollten sie beachten, dass dieselben Techniken auch beim Wiederherstellen von gelöschten Verzeichnissen funktionieren. Beachten Sie auch, dass die Wiederherstellung nicht unbedingt in Ihrer Arbeitskopie passieren muss – sie kann auch vollständig im Projektarchiv ausgeführt werden:

```
$ svn copy ^/calc/trunk/real.c@807 ^/calc/trunk/ \
      -m "real.c aus Revision 807 wiederhergestellt."
Revision 1390 übertragen.

$ svn update
A    real.c
Aktualisiert zu Revision 1390.
```

Fortgeschrittenes Zusammenführen

Hier endet die automatische Magie. Früher oder später, sobald Sie den Dreh beim Verzweigen und Zusammenführen heraus haben, werden Sie Subversion fragen müssen, *bestimmte* Änderungen von einem Ort zum anderen zusammenzuführen. Um dies tun zu können, werden Sie damit beginnen müssen, kompliziertere Argumente an **svn merge** zu übergeben. Der nächste Abschnitt beschreibt die vollständig erweiterte Syntax des Befehls und behandelt eine Anzahl verbreiteter Szenarien, die diese benötigen.

Die Rosinen herauspicken

Genauso oft wie der Begriff „Änderungsmenge“ wird die Wendung *die Rosinen herauspicken* in Versionskontrollsystemen verwendet. Das bezieht sich darauf, *eine* bestimmte Änderungsmenge von einem Zweig auszuwählen und sie auf einen anderen anzuwenden. Die Rosinen herauszupicken kann sich auch darauf beziehen, eine bestimmte Menge von (nicht notwendigerweise angrenzenden) Änderungsmengen von einem auf einen anderen Zweig zu duplizieren. Dies steht im Gegensatz zu den üblicheren Zusammenführungs-Szenarien, bei denen der „nächste“ zusammenhängende Bereich von Revisionen automatisch dupliziert wird.

Warum sollte jemand nur eine einzelne Änderung wollen? Das kommt häufiger vor, als Sie denken. Gehen wir beispielsweise einmal zurück in die Vergangenheit und stellen uns vor, dass Sie Ihren Zweig noch nicht wieder mit dem Stamm zusammengeführt hätten. In der Kaffeeküche bekommen Sie mit, dass Sally eine interessante Änderung an `integer.c` auf dem Stamm gemacht hat. Als Sie sich die Geschichte der Übergaben auf dem Stamm ansehen, entdecken Sie, dass sie in Revision 355 einen kritischen Fehler beseitigt hat, der direkte Auswirkungen auf die Funktion hat, an der Sie gerade arbeiten. Es kann sein, dass Sie noch nicht bereit sind, alle Änderungen vom Stamm zu übernehmen, jedoch benötigen Sie diese bestimmte Fehlerbehebung, um mit Ihrer Arbeit weitermachen zu können.

```
$ svn diff -c 355 ^/calc/trunk

Index: integer.c
===================================================================
--- integer.c (revision 354)
+++ integer.c (revision 355)
@@ -147,7 +147,7 @@
     case 6:  sprintf(info->operating_system, "HPFS (OS/2 or NT)"); break;
     case 7:  sprintf(info->operating_system, "Macintosh"); break;
     case 8:  sprintf(info->operating_system, "Z-System"); break;
-    case 9:  sprintf(info->operating_system, "CP/MM");
+    case 9:  sprintf(info->operating_system, "CP/M"); break;
     case 10:  sprintf(info->operating_system, "TOPS-20"); break;
     case 11:  sprintf(info->operating_system, "NTFS (Windows NT)"); break;
     case 12:  sprintf(info->operating_system, "QDOS"); break;
```

Ebenso wie Sie **svn diff** im vorigen Beispiel benutzt haben, um sich Revision 355 anzusehen, können Sie die gleiche Option an **svn merge** übergeben:

```
$ svn merge -c 355 ^/calc/trunk
--- Zusammenführen von r355 in ».«:
U    integer.c

$ svn status
M       integer.c
```

Sie können nun Ihre üblichen Tests durchführen, bevor Sie diese Änderung an den Zweig übergeben. Nach der Übergabe merkt sich Subversion, dass r355 mit dem Zweig zusammengeführt wurde, so dass künftige „magische“ Zusammenführungen, die Ihren Zweig mit dem Stamm synchronisieren, r355 überspringen. (Das Zusammenführen derselben Änderung auf denselben Zweig führt fast immer zu einem Konflikt!)

```
$ cd my-calc-branch

$ svn propget svn:mergeinfo .
/trunk:341-349,355

# Beachten Sie, dass r355 nicht als Zusammenführungs-Kandidat aufgeführt wird
# da es bereits zusammengeführt wurde.
$ svn mergeinfo ^/calc/trunk --show-revs eligible
r350
r351
r352
r353
r354
r356
r357
r358
r359
r360

$ svn merge ^/calc/trunk
--- Zusammenführen von r350 bis r354 in ».«:
 U   .
U    integer.c
U    Makefile
--- Zusammenführen von r356 bis r360 in ».«:
 U   .
U    integer.c
U    button.c
```

Dieser Anwendungsfall des Abgleichens (oder *Nachziehens*) von Fehlerbehebungen von einem Zweig zu einem anderen ist vielleicht der gängigste Grund für Änderungen, die Rosinen herauszupicken; es kommt ständig vor, beispielsweise, wenn ein Team einen „Software-Release-Zweig“ verwendet. (Wir erörtern dieses Muster in „Release-Zweige“.)

Haben Sie bemerkt, wie im letzten Beispiel der Aufruf von **svn merge** dazu geführt hat, zwei unterschiedliche Abgleichsintervalle anzuwenden? Der Befehl führte zwei unabhängige Patches auf Ihrer Arbeitskopie aus, um die Änderungsmenge 355 zu überspringen, die Ihr Zweig bereits beinhaltete. An und für sich ist daran nichts falsch, bis auf die Tatsache, dass die Möglichkeit besteht, eine Konfliktauflösung komplizierter zu machen. Falls das erste Änderungsintervall Konflikte erzeugt, *müssen* Sie diese interaktiv auflösen, um die Zusammenführung fortzusetzen und das zweite Änderungsintervall anzuwenden. Wenn Sie die Konfliktauflösung der ersten Phase aufschieben, wird der komplette Zusammenführungsbefehl mit einer Fehlermeldung abbrechen.[4]

Ein Wort zur Warnung: Während **svn diff** und **svn merge** vom Konzept her sehr ähnlich sind, haben sie in vielen Fällen eine unterschiedliche Syntax. Gehen Sie sicher, dass Sie Details hierzu in Kapitel 9, *Die vollständige Subversion Referenz* nachlesen oder **svn help** fragen. Zum Beispiel benötigt **svn merge** einen Pfad in der Arbeitskopie als Ziel, d.h., einen Ort, an dem es den erzeugten Patch anwenden kann. Falls das Ziel nicht angegeben wird, nimmt es an, dass Sie eine der folgenden

[4] Zumindest trifft das zur Zeit für Subversion 1.6 zu. Dieses Verhalten könnte sich in künftigen Versionen von Subversion verbessern.

häufigen Operationen durchführen möchten:

- Sie möchten Verzeichnisänderungen auf Ihr aktuelles Arbeitsverzeichnis abgleichen.
- Sie möchten die Änderungen in einer bestimmten Datei mit einer Datei gleichen Namens in Ihrem aktuellen Arbeitsverzeichnis zusammenführen.

Falls Sie ein Verzeichnis zusammenführen und keinen Zielpfad angegeben haben, nimmt **svn merge** den ersten Fall an und versucht, die Änderungen auf Ihr aktuelles Arbeitsverzeichnis anzuwenden. Falls Sie eine Datei zusammenführen und diese Datei (oder eine gleichnamige Datei) in Ihrem aktuellen Arbeitsverzeichnis existiert, nimmt **svn merge** den zweiten Fall an und wendet die Änderungen auf eine lokale Datei gleichen Namens an.

Merge-Syntax: Die vollständige Enthüllung

Sie haben nun einige Beispiele zum Befehl **svn merge** gesehen und werden bald einige mehr sehen. Falls Sie verwirrt darüber sind, wie das Zusammenführen genau funktioniert, sind Sie nicht alleine. Viele Anwender (besonders diejenigen, für die Versionskontrolle etwas Neues ist) sind anfangs verwirrt darüber, wie die korrekte Syntax des Befehls lautet und wann das Feature verwendet werden soll. Aber, keine Angst, dieser Befehl ist tatsächlich viel einfacher als Sie denken! Es gibt eine einfache Technik, die verstehen hilft, wie sich **svn merge** genau verhält.

Die Hauptquelle der Verwirrung ist der *Name* des Befehls. Der Begriff „merge“ (Zusammenführung, Mischung) deutet irgendwie an, dass Zweige miteinander verschmolzen werden, oder dass irgendeine geheimnisvolle Mischung der Daten erfolgt. Das ist nicht der Fall. Ein besserer Name für den Befehl wäre vielleicht **svn ermittele-die-Unterschiede-und-wende-sie-an** gewesen, da das alles ist, was passiert: Die Bäume im Projektarchiv werden verglichen und die Unterschiede in eine Arbeitskopie eingearbeitet.

Falls Sie **svn merge** benutzen, um einfache Kopien von Änderungen zwischen Zweigen vorzunehmen, wird es üblicherweise automatisch das Richtige machen. Beispielsweise wird ein Befehl wie der folgende:

```
$ svn merge ^/calc/branches/some-branch
```

versuchen, alle Änderungen, die auf `some-branch` gemacht worden sind, in Ihr aktuelles Arbeitsverzeichnis zu kopieren, welches vermutlich eine Arbeitskopie ist, die mit dem Zweig irgendeine historische Verbindung teilt. Der Befehl ist klug genug, nur die Änderungen zu kopieren, die Ihre Arbeitskopie noch nicht hat. Wenn Sie diesen Befehl einmal die Woche wiederholen, wird er nur die „neuesten“ Änderungen vom Zweig kopieren, die seit Ihrem letzten Zusammenführen stattfanden.

Wenn Sie den Befehl **svn merge** in seiner ganzen Pracht wählen, indem Sie ihm bestimmte Revisionsintervalle zum kopieren übergeben, benötigt der Befehl drei Hauptargumente:

1. Einen Anfangsbaum im Projektarchiv (häufig *linke Seite* des Vergleichs genannt)
2. Einen Endbaum im Projektarchiv (häufig *rechte Seite* des Vergleichs genannt)
3. Eine Arbeitskopie, die die Unterschiede als lokale Änderungen aufnimmt (häufig *Ziel* der Zusammenführung genannt)

Sobald diese drei Argumente angegeben sind, werden die zwei Bäume miteinander verglichen und die Unterschiede als lokale Änderungen auf die Ziel-Arbeitskopie angewendet. Wenn der Befehl fertig ist, sieht das Ergebnis so aus, als hätten Sie die Dateien manuell editiert oder verschiedene **svn add**- oder **svn delete**-Befehle ausgeführt. Wenn Ihnen das Ergebnis gefällt, können Sie es übergeben. Falls nicht, können Sie einfach mit **svn revert** alle Änderungen rückgängig machen.

Die Syntax von **svn merge** erlaubt Ihnen, die drei notwendigen Argumente auf eine recht flexible Weise anzugeben. Hier sind einige Beispiele:

```
$ svn merge http://svn.example.com/repos/branch1@150 \
            http://svn.example.com/repos/branch2@212 \
```

```
            my-working-copy
$ svn merge -r 100:200 http://svn.example.com/repos/trunk my-working-copy
$ svn merge -r 100:200 http://svn.example.com/repos/trunk
```

Die erste Syntax führt alle drei Argumente explizit auf, indem jeder Baum mit dem Format *URL@REV* bezeichnet und die Ziel-Arbeitskopie angegeben wird. Die zweite Syntax kann als Kurzform verwendet werden, wenn Sie zwei unterschiedliche Revisionen desselben URL vergleichen. Die letzte Syntax zeigt, dass das Arbeitskopie-Argument optional ist; entfällt es, wird das aktuelle Verzeichnis genommen.

Obwohl das erste Beispiel die „vollständige“ Syntax von **svn merge** zeigt, muss sie sehr sorgfältig verwendet werden; es können hierbei Zusammenführungen entstehen, bei denen keinerlei `svn:mergeinfo` Metadaten aufgezeichnet werden. Der nächste Abschnitt geht näher darauf ein.

Zusammenführen ohne Zusammenführungsinformationen

Subversion versucht immer wenn es kann, Metadaten über das Zusammenführen zu erzeugen, um spätere Aufrufe von **svn merge** schlauer zu machen. Trotzdem gibt es Situationen, in denen `svn:mergeinfo`-Daten nicht erzeugt oder geändert werden. Denken Sie daran, vor diesen Szenarien auf der Hut zu sein:

Zusammenführen von Quellen ohne Beziehung
: Falls Sie **svn merge** dazu auffordern, zwei URLs zu vergleichen, die nicht miteinander in Beziehung stehen, wird trotzdem ein Patch erzeugt und auf die Arbeitskopie angewendet, allerdings werden keine Metadaten erzeugt. Es gibt keine gemeinsame Geschichte der zwei Quellen, und spätere „schlaue“ Zusammenführungen hängen von dieser gemeinsamen Geschichte ab.

Zusammenführen aus fremden Projektarchiven
: Obwohl es möglich ist, einen Befehl wie **`svn merge -r 100:200 http://svn.foreignproject.com/repos/trunk`** auszuführen, wird auch dieser resultierende Patch keine historischen Metadaten über die Zusammenführung haben. Zum gegenwärtigen Zeitpunkt hat Subversion keine Möglichkeit, unterschiedliche Projektarchiv-URLs innerhalb der Eigenschaft `svn:mergeinfo` zu repräsentieren.

Verwendung von `--ignore-ancestry`
: Wenn diese Option an **svn merge** übergeben wird, veranlasst das die Zusammenführungs-Logik, ohne nachzudenken Unterschiede auf dieselbe Art zu erzeugen, wie es **svn diff** macht, und ignoriert dabei irgendwelche historischen Verbindungen. Wir werden das später in diesem Kapitel in „Die Abstammung berücksichtigen oder ignorieren“ erörtern.

Zusammenführen rückgängig machen
: Weiter oben in diesem Kapitel („Änderungen rückgängig machen“) haben wir darüber gesprochen, wie man mit **svn merge** einen „Rückwärts-Patch“ verwendet, um Änderungen rückgängig zu machen. Wenn diese Technik dazu verwendet wird, um eine Änderung in der Geschichte eines Objektes zurückzunehmen (z.B. r5 an den Stamm übergeben, und dann sofort r5 mit **`svn merge . -c -5`** rückgängig machen), hat dies keine Auswirkungen auf die aufgezeichneten Zusammenführungsinformationen.[5]

Natürliche Historie und implizite Zusammenführungsinformation

Wenn bei einem Pfad die Eigenschaft `svn:mergeinfo` gwsetzt ist, nennen wir das „explizite“ Zusammenführungsinformation. Ja, das bedeutet, dass ein Pfad auch „implizite“ Zusammenführungsinformation haben kann! Implizite Zusammenführungsinformation, oder *natürliche Historie*, ist einfach die dem Pfad eigene Historie (siehe „Geschichtsforschung“) die als Information zur Verfolgung von Zusammenführungen interpretiert wird. Obwohl es sich bei impliziter Zusammenführungsinformation größtenteils um Details der Implementierung handelt, kann es als nützliche Abstraktion für das Verstehen des Verhaltens der Zusammenführungsverfolgung dienen.

[5] Interessanterweise werden wir nach dem Zurücknehmen einer Revision auf diese Art nicht in der Lage sein, diese Revision erneut mit **`svn merge . -c 5`** anzuwenden, da aus den Metadaten hervorgeht, dass r5 bereits angewendet wurde. Wir müssten die Option `--ignore-ancestry` verwenden, damit der Befehl die bestehenden Metadaten ignoriert.

Nehmen wir mal an, Sie hätten in Revision 100 `^/trunk` erzeugt und dann in Revision 201 `^/branches/feature-branch` als Kopie von `^/trunk@200` erzeugt. Die natürliche Historie von `^/branches/feature-branch` enthält all diejenigen Pfade im Projektarchiv und Revisionsintervalle, die die Historie des neuen Zweigs je berührt hat.

/trunk:100-200
/branches/feature-branch:201

Mit jeder dem Projektarchiv hinzugefügten neuen Revision wird die nat#rliche Historie – und somit die implizite Zusammenführungsinformation – des Zweigs durch diese Revisionen stetig erweitert, bis zu dem Tag, an dem der Zweig gelöscht wird. So würde die implizite Zusammenführungsinformation unseres Zweiges aussehen, wenn die `HEAD`-Revision des Projektarchivs bis auf 234 angewachsen wäre:

/trunk:100-200
/branches/feature-branch:201-234

Implizite Zusammenführungsinformation taucht nicht wirklich in der Eigenschaft `svn:mergeinfo` auf, doch Subversion handelt so, als tauchte sie dort auf. Das ist der Grund dafür, dass nichts passiert, wenn Sie `^/branches/feature-branch` auschecken und dann **`svn merge ^/trunk -c 58`** in der so erstellten Arbeitskopie aufrufen. Subversion weiß, dass die in Revision 53 an `^/trunk` übergebenen Änderunge bereits in der natürlichen Historie des Ziels vohanden sind, so dass sie nicht erneut zusammengefuhrt werden mussen. Letzten Endes *ist* es das primäre Ziel der Zusammenführungsverfolgungs/Funktionalität von Subversion, unnötige Zusammenführungen zu vermeiden!

Mehr über Konflikte beim Zusammenführen

Wie der Befehl **svn update** wendet auch **svn merge** Änderungen auf Ihre Arbeitskopie an. Und deshalb kann er auch Konflikte erzeugen. Die von **svn merge** hervorgerufenen Konflikte sind jedoch manchmal anders geartet, und dieser Abschnitt erklärt diese Unterschiede.

Zunächst gehen wir davon aus, dass Ihre Arbeitskopie keine lokalen Änderungen enthält. Wenn Sie mit **svn update** auf eine bestimmte Revision aktualisieren, werden die vom Server gesendeten Änderungen immer „sauber“ auf Ihre Arbeitskopie angewendet. Der Server erzeugt das Delta, indem er zwei Bäume vergleicht: eine virtuelle Momentaufnahme Ihrer Arbeitskopie und der Revisionsbaum, an dem Sie interessiert sind. Da die linke Seite des Vergleichs völlig gleich zu dem ist, was Sie bereits haben, wird das Delta garantiert Ihre Arbeitskopie korrekt in den rechten Baum überführen.

svn merge jedoch kann das nicht gewährleisten und kann viel chaotischer sein: Der fortgeschrittene Benutzer kann den Server auffordern, *irgendwelche* zwei Bäume miteinander zu vergleichen, sogar solche, die nicht mit der Arbeitskopie in Beziehung stehen! Das bedeutet, dass ein hohes Potenzial für menschliche Fehler besteht. Benutzer werden manchmal die falschen zwei Bäume miteinander vergleichen, so dass ein Delta erzeugt wird, das sich nicht sauber anwenden lässt. **svn merge** wird sein Bestes geben, um soviel wie möglich vom Delta anzuwenden, doch bei einigen Teilen kann das unmöglich sein. So wie der Unix-Befehl **patch** sich manchmal über „failed hunks“ beschwert, wird sich **svn merge** ähnlich über „skipped targets“ beschweren:

```
$ svn merge -r 1288:1351 http://svn.example.com/myrepos/branch
U    foo.c
U    bar.c
Fehlendes Ziel: »baz.c« übersprungen.
U    glub.c
U    sputter.h
```

```
Konflikt in »glorb.h« entdeckt.
Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
         (mc) eigene konfliktbehaftete Datei, (tc) fremde konfliktbehaftete Datei
         (s) alle Optionen anzeigen:
```

Im vorangegangenen Beispiel kann es der Fall gewesen sein, dass `baz.c` in beiden Momentaufnahmen des Zweiges vorkommt, die verglichen werden, und das resultierende Delta den Inhalt der Datei verändern will, die in der Arbeitskopie aber nicht vorhanden ist. Wie auch immer, die „skipped“-Nachricht bedeutet, dass der Benutzer höchstwahrscheinlich die falschen Bäume miteinander vergleicht; es ist das klassische Zeichen für einen Anwenderfehler. Falls dies passiert, ist es einfach, alle durch das Zusammenführen hervorgerufenen Änderungen rekursiv rückgängig zu machen (**`svn revert . --recursive`**), alle unversionierten Dateien oder Verzeichnisse zu löschen, die nach dem Rückgängigmachen zurückgeblieben sind, und **svn merge** noch einmal mit unterschiedlichen Argumenten aufzurufen.

Beachten Sie auch, dass das vorangegangene Beispiel einen Konflikt in `glorb.h` anzeigt. Wir bemerkten bereits, dass die Arbeitskopie keine lokalen Änderungen besitzt: Wie kann da ein Konflikt entstehen? Noch einmal: Weil der Benutzer **svn merge** dazu verwenden kann, ein altes Delta zu definieren und auf die Arbeitskopie anzuwenden, kann es sein, dass dieses alte Delta textuelle Änderungen enthält, die nicht sauber in eine Arbeitsdatei eingearbeitet werden können, selbst dann nicht, wenn die Datei keine lokalen Änderungen vorzuweisen hat.

Ein weiterer kleiner Unterschied zwischen **svn update** und **svn merge** sind die Namen der erzeugten Textdateien, falls ein Konflikt entsteht. In „Lösen Sie etwaige Konflikte auf“ sahen wir, dass bei einer Aktualisierung die Dateien namens `filename.mine`, `filename.rOLDREV` und `filename.rNEWREV` erzeugt werden. Falls **svn merge** einen Konflikt hervorruft, erstellt es jedoch drei Dateien namens `filename.working`, `filename.left` und `filename.right`. In diesem Fall beschreiben die Begriffe „left“ (links) und „right“ (rechts) von welcher Seite des Vergleichs zwischen den beiden Bäumen die Datei hergeleitet wurde. Auf alle Fälle werden Ihnen diese unterschiedlichen Namen dabei helfen, zwischen Konflikten zu unterscheiden, die durch eine Aktualisierung entstanden, und solchen die durch eine Zusammenführung hervorgerufen wurden .

Änderungen blockieren

Manchmal gibt es eine bestimmte Änderungsmenge, die Sie nicht automatisch zusammengeführt haben wollen. Beispielsweise ist vielleicht die Vorgehensweise Ihres Teams dergestalt, dass Neuentwicklungen auf `/trunk` gemacht werden, aber konservativer, wenn es darum geht, Änderungen auf einen stabilen Zweig zurückzuportieren, den sie zur Veröffentlichung benutzen. Auf der einen Seite können Sie die Rosinen in Form von einzelnen Änderungsmengen manuell aus dem Stamm herauspicken und in den Zweig einpflegen – nur die Änderungen, die stabil genug sind, um die Qualitätsprüfung zu bestehen. Vielleicht ist es ja auch nicht ganz so streng, und Sie möchten normalerweise, dass **svn merge** die meisten Änderungen vom Stamm automatisch mit dem Zweig zusammenführt. In diesem Fall könnten Sie ein Verfahren gebrauchen, dass es Ihnen erlaubt, einige bestimmte Änderungen auszulassen, d.h. zu vermeiden, dass sie automatisch in den Zweig eingebracht werden.

Die einzige Möglichkeit, mit Subversion 1.6 eine Änderungsmenge zu blockieren, besteht darin, dem System vorzugaukeln, dass die Änderung *bereits* eingearbeitet wurde. Dazu rufen Sie den Befehl mit der Option `--record-only` auf:

```
$ cd my-calc-branch

$ svn propget svn:mergeinfo .
/trunk:1680-3305

# In den Metadaten r3328 als bereits zusammengeführt vermerken.
$ svn merge -c 3328 --record-only http://svn.example.com/repos/calc/trunk

$ svn status
M       .

$ svn propget svn:mergeinfo .
/trunk:1680-3305,3328

$ svn commit -m "Das Zusammenführen von r3328 mit dem Zweig verhindern."
…
```

Diese Technik funktioniert zwar, sie ist allerdings auch ein wenig gefährlich. Das Hauptproblem ist, dass wir nicht klar unterscheiden zwischen „ich habe diese Änderung bereits“ und „ich habe diese Änderung nicht, aber ich will sie jetzt nicht“. Wir belügen das System gewissermaßen, indem wir es glauben lassen, dass die Änderung schon eingearbeitet sei. Das schiebt die Verantwortung, sich daran zu erinnern, dass die Änderung tatsächlich gar nicht übernommen wurde sondern nicht gewünscht war, auf Sie – den Benutzer. Es gibt keine Möglichkeit, Subversion nach einer Liste „blockierter Änderungen“ zu fragen. Wenn Sie sie verfolgen möchten (so dass Sie eines Tages die Blockierung aufheben können) müssen Sie sie irgendwo in eine Textdatei schreiben oder in einer erfundenen Eigenschaft festhalten.

Einen reintegrierten Zweig am Leben erhalten

Es gibt einen anderen Weg, als einen Zweig nach der Reintegration zu zerstören und erneut zu erzeugen. Zum Verständnis, warum das funktioniert, müssen Sie verstehen, warum der Zweig unmittelbar nach dem Reintegrieren zunächst nicht weiterverwendbar ist.

Nehmen wir an, Sie haben den Zweig in Revision *A* angelegt. Bei der Arbeit auf diesem Zweig haben Sie eine oder mehrere Revisionen erzeugt, die Änderungen an dem Zweig beinhalten. Vor der Reintegration des Zweigs auf die Hauptentwicklungslinie haben Sie eine abschließende Zusammenführung von dort auf Ihren Zweig vollzogen und das Ergebnis dieser Zusammenführung als Revision *B* übertragen.

Bei der Reintegration Ihres Zweigs auf die Hauptentwicklungslinie erzeugen Sie eine neue Revision *X* die jene ändert. Die Änderungen an der Hauptentwicklungslinie in dieser Revision *X* sind semantisch äquivalent zu den Änderungen, die Sie zwischen Revision *A* und *B* auf Ihrem Zweig vorgenommen haben.

Falls Sie jetzt versuchen, ausstehende Änderungen von der Hauptentwicklungslinie mit Ihrem Zweig zusammenzuführen, wird Subversion die in Revision *X* vorgenommenen Änderungen als Kandidaten für die Zusammenführung betrachten. Da Ihr Zweig jedoch bereits alle in Revision *X* vorgenommenen Änderungen enthält, kann das Zusammenführen dieser Änderungen fälschlicherweise zu Konflikten führen! Bei diesen Konflikten handelt es sich oft um Baumkonflikte, besonders dann, wenn während der Entwicklung auf dem Zweig dort oder auf der Hauptentwicklungslinie Umbenennungen gemacht wurden.

Wie soll also damit umgegangen werden? Wir müssen sicherstellen, dass Subversion nicht versucht, die Revision *X* mit dem Zweig zusammenzuführen. Das kann mit der Zusammenführungs-Option `--record-only` erreicht werden, die in „Änderungen blockieren“ vorgestellt wurde.

Um die nur vermerkte Zusammenführung auszuführen, erstellen Sie sich eine Arbeitskopie des Zweigs, der in Revision *X* frisch reintegriert wurde, und führen Sie lediglich die Revision *X* von der Hauptentwicklungslinie mit Ihrem Zweig zusammen, indem Sie sicherstellen, dass Sie die Option `--record-only` verwenden.

Diese Zusammenführung verwendet die Syntax zum Herauspicken der Rosinen, wie sie in „Die Rosinen herauspicken“ vorgestellt wurde. Um mit dem aktuellen Beispiel aus „Reintegration eines Zweigs“ fortzufahren, in dem Revision *X* die Revision 391 war:

```
$ cd my-calc-branch
$ svn update
Aktualisiert zu Revision 393.
$ svn merge --record-only -c 391 ^/calc/trunk
$ svn commit -m "Block revision 391 from being merged into my-calc-branch."
Sende        .

Revision 394 übertragen.
```

Nun ist Ihr Zweig wieder bereit, Änderungen von der Hauptentwicklungslinie aufzunehmen. Nach einer weiteren Synchronisierung Ihres Zweigs auf die Hauptentwicklungslinie können Sie sogar ein zweites Mal Ihren Zweig reintegrieren. Falls notwendig, können Sie eine weitere, nur vermerkte Zusammenführung machen, um den Zweig am Leben zu erhalten. Spülen und wiederholen.

Nun sollte es auch offensichtlich sein, warum das Löschen und Wiederherstellen des Zweigs den selben Effekt hat wie die obige nur vermerkte Zusammenführung. Da die Revision *X* Teil der natürlichen Historie des frisch erzeugten Zweigs ist, wird Subversion niemals versuchen, die Revision *X* mit dem Zweig zusammenzuführen, und vermeidet dadurch fälschliche Konflikte.

Protokolle und Anmerkungen, die Zusammenführungen anzeigen

Ein Hauptmerkmal jedes Versionskontrollsystems ist es, darüber Buch zu führen, wer was wann geändert hat. Die Befehle **svn log** und **svn blame** sind die geeigneten Werkzeuge hierfür: Wenn sie auf individuelle Dateien angewendet werden, zeigen sie nicht nur die Geschichte der Änderungsmengen, die in diese Datei hineinflossen, sondern auch, welcher Benutzer wann welche Zeile im Quelltext geschrieben hat.

Wenn jedoch Änderungen über Zweige hinweg dupliziert werden, wird es schnell kompliziert. Wenn Sie z.B. **svn log** nach der Geschichte Ihres Zweigs fragen, wird es Ihnen exakt jede Revision anzeigen, die je in den Zweig hineingeflossen ist:

```
$ cd my-calc-branch
$ svn log -q
------------------------------------------------------------------------
r390 | user | 2002-11-22 11:01:57 -0600 (Fr, 22. Nov 2002) | 1 line
------------------------------------------------------------------------
r388 | user | 2002-11-21 05:20:00 -0600 (Do, 21. Nov 2002) | 2 lines
------------------------------------------------------------------------
r381 | user | 2002-11-20 15:07:06 -0600 (Mi, 20. Nov 2002) | 2 lines
------------------------------------------------------------------------
r359 | user | 2002-11-19 19:19:20 -0600 (Di, 19. Nov 2002) | 2 lines
------------------------------------------------------------------------
r357 | user | 2002-11-15 14:29:52 -0600 (Fr, 15. Nov 2002) | 2 lines
------------------------------------------------------------------------
r343 | user | 2002-11-07 13:50:10 -0600 (Do, 07. Nov 2002) | 2 lines
------------------------------------------------------------------------
r341 | user | 2002-11-03 07:17:16 -0600 (So, 03. Nov 2002) | 2 lines
------------------------------------------------------------------------
r303 | sally | 2002-10-29 21:14:35 -0600 (Di, 29. Oct 2002) | 2 lines
------------------------------------------------------------------------
r98 | sally | 2002-02-22 15:35:29 -0600 (Fr, 22. Feb 2002) | 2 lines
------------------------------------------------------------------------
```

Aber ist das wirklich eine genaue Wiedergabe aller Änderungen, die auf dem Zweig stattgefunden haben? Was hier ausgelassen wird, ist, dass die Revisionen 390, 381 und 357 tatsächlich Ergebnisse des Zusammenführens von Änderungen aus dem Stamm waren. Wenn Sie sich eins dieser Protokolle im Detail anschauen, können Sie die verschiedenen Änderungsmengen vom Stamm, die die Änderungen auf dem Zweig ausmachen, nirgendwo sehen:

```
$ svn log -v -r 390
------------------------------------------------------------------------
r390 | user | 2002-11-22 11:01:57 -0600 (Fri, 22 Nov 2002) | 1 line
Geänderte Pfade:
   M /branches/my-calc-branch/button.c
   M /branches/my-calc-branch/README

Letzte Zusammenführung der Änderungen von trunk changes in my-calc-branch.
```

Wir wissen, dass diese Zusammenführung in den Zweig nichts anderes war als eine Zusammenführung von Änderungen vom Stamm. Wie können wir zusätzlich diese Änderungen sehen? Die Antwort lautet, die Option `--use-merge-history` (`-g`) zu verwenden. Diese Option expandiert diejenigen „Teil“-Änderungen, aus denen die Zusammenführung bestand.

```
$ svn log -v -r 390 -g
------------------------------------------------------------------------
r390 | user | 2002-11-22 11:01:57 -0600 (Fri, 22 Nov 2002) | 1 line
Geänderte Pfade:
   M /branches/my-calc-branch/button.c
   M /branches/my-calc-branch/README
```

```
Letzte Zusammenführung der Änderungen von trunk changes in my-calc-branch.
------------------------------------------------------------------------
r383 | sally | 2002-11-21 03:19:00 -0600 (Thu, 21 Nov 2002) | 2 lines
Geänderte Pfade:
   M /branches/my-calc-branch/button.c
Zusammengeführt mittels: r390

Inverse Grafik auf Knopf behoben.
------------------------------------------------------------------------
r382 | sally | 2002-11-20 16:57:06 -0600 (Wed, 20 Nov 2002) | 2 lines
Geänderte Pfade:
   M /branches/my-calc-branch/README
Zusammengeführt mittels: r390

Meine letzte Änderung in README dokumentiert.
```

Dadurch, dass wir die Protokoll-Operation aufgefordert haben, die Geschichte der Zusammenführungen zu verwenden, sehen wir nicht nur die Revision, die wir abgefragt haben (r390), sondern auch die zwei Revisionen, die hier mitkamen – ein paar Änderungen, die Sally auf dem Stamm gemacht hat. Das ist ein wesentlich vollständigeres Bild der Geschichte!

Auch der **svn blame**-Befehl versteht die Option `--use-merge-history` (`-g`). Falls diese Option vergessen wird, könnte jemand, der sich die zeilenweisen Anmerkungen von `button.c` ansieht, fälschlicherweise davon ausgehen, dass Sie für die Zeilen verantwortlich sind, die einen bestimmten Fehler beseitigt haben:

```
$ svn blame button.c
…
   390    user    retval = inverse_func(button, path);
   390    user    return retval;
   390    user    }
…
```

Obwohl es zutrifft, dass Sie diese drei Zeilen in Revision 390 übergeben haben, sind zwei davon tatsächlich von Sally in Revision 383 geschrieben worden:

```
$ svn blame button.c -g
…
G    383    sally   retval = inverse_func(button, path);
G    383    sally   return retval;
     390    user    }
…
```

Nun wissen wir, wer *wirklich* für die zwei Zeilen Quelltext verantwortlich ist!

Die Abstammung berücksichtigen oder ignorieren

Wenn Sie sich mit einem Subversion-Entwickler unterhalten, wird wahrscheinlich auch der Begriff *Abstammung* erwähnt. Dieses Wort wird verwendet, um die Beziehung zwischen zwei Objekten im Projektarchiv zu beschreiben: Wenn sie in Beziehung zueinander stehen, heißt es, dass ein Objekt vom anderen abstammt.

Nehmen wir an, Sie übergeben Revision 100, die eine Änderung an der Datei `foo.c` beinhaltet. Dann ist `foo.c@99` ein „Vorfahre“ von `foo.c@100`. Wenn Sie dagegen in Revision 101 die Löschung von `foo.c` übergeben und in Revision 102 eine neue Datei mit demselben Namen hinzufügen, hat es zwar den Anschein, dass `foo.c@99` und `foo.c@102` in Beziehung zueinander stehen (sie haben denselben Pfad), es handelt sich allerdings um völlig unterschiedliche Objekte im Projektarchiv. Sie haben weder eine gemeinsame Geschichte noch „Abstammung“.

Wir erwähnen das, um auf einen wichtigen Unterschied zwischen den Befehlen **svn diff** und **svn merge** hinzuweisen. Der

erstere Befehl ignoriert die Abstammung, wohingegen letzterer diese beachtet. Wenn Sie beispielsweise mit **svn diff** die Revisionen 99 und 102 von `foo.c` vergleichen, werden Sie zeilenbasierte Unterschiede sehen; der Befehl **diff** vergleicht blind zwei Pfade. Wenn Sie aber dieselben Objekte mit **svn merge** vergleichen, wird es feststellen, dass sie nicht in Beziehung stehen und versuchen, die alte Datei zu löschen und dann die neue hinzuzufügen; die Ausgabe wird eine Löschung gefolgt von einer Hinzufügung anzeigen:

```
D     foo.c
A     foo.c
```

Die meisten Zusammenführungen vergleichen Bäume, die von der Abstammung her miteinander in Beziehung stehen, deshalb verhält sich **svn merge** auf diese Weise. Gelegentlich möchten Sie jedoch mit dem **merge**-Befehl zwei Bäume vergleichen, die nicht miteinander in Beziehung stehen. Es kann z.B. sein, dass Sie zwei Quelltext-Bäume importiert haben, die unterschiedliche Lieferantenstände eines Software-Projektes repräsentieren (siehe „Lieferanten-Zweige“). Falls Sie **svn merge** dazu aufforderten, die beiden Bäume miteinander zu vergleichen, würden Sie sehen, dass der vollständige erste Baum gelöscht und anschließend der vollständige zweite Baum hinzugefügt würde! In diesen Situationen möchten Sie, dass **svn merge** lediglich einen pfadbasierten Vergleich vornimmt und Beziehungen zwischen Dateien und Verzeichnissen außer Acht lässt. Fügen Sie die Option `--ignore-ancestry` dem **merge**-Befehl hinzu, und er wird sich verhalten wie **svn diff**. (Auf der anderen Seite wird die Option `--notice-ancestry` den Befehl **svn diff** dazu veranlassen, sich wie **svn merge** zu verhalten.

Zusammenführen und Verschieben

Es ist ein verbreiteter Wunsch, Software zu refaktorieren, besonders in Java-basierten Software-Projekten. Dateien und Verzeichnisse werden hin und her geschoben und umbenannt, was häufig zu erheblichen Beeinträchtigungen für alle Projektmitarbeiter führt. Das hört sich an, als sei das der klassische Fall, um nach einem Zweig zu greifen, nicht wahr? Sie erzeugen einfach einen Zweig, schieben das Zeug herum und führen anschließend den Zweig mit dem Stamm zusammen.

Leider funktioniert dieses Szenario im Augenblick noch nicht so richtig und gilt als einer der Schwachpunkte von Subversion. Das Problem ist, das der Subversion-Befehl **svn update** nicht so stabil ist, wie er sein sollte, besonders wenn es um Kopier- und Verschiebeoperationen geht.

Wenn Sie **svn copy** zum Duplizieren einer Datei verwenden, merkt sich das Projektarchiv, woher die neue Datei kam, versäumt aber, diese Information an den Client zu senden, der **svn update** oder **svn merge** ausführt. Statt dem Client mitzuteilen: „Kopiere die Datei, die du bereits hast an diesen neuen Ort“, sendet es eine völlig neue Datei. Das kann zu Problemen führen, besonders, weil dasselbe mit umbenannten Dateien passiert. Eine weniger bekannte Tatsache über Subversion ist, dass es keine „echten Umbenennungen“ hat – der Befehl **svn move** ist weiter nichts als eine Verbindung von **svn copy** und **svn delete**.

Nehmen wir beispielsweise an, dass Sie während Ihrer Arbeit auf Ihrem privaten Zweig `integer.c` in `whole.c` umbenennen. Tatsächlich haben Sie eine neue Datei auf Ihrem Zweig erzeugt, die eine Kopie der ursprünglichen Datei ist, und letztere gelöscht. Zwischenzeitlich hat Sally einige Verbesserungen an `integer.c` in `trunk` übergeben. Nun entscheiden Sie sich, Ihren Zweig mit dem Stamm zusammenzuführen:

```
$ cd calc/trunk

$ svn merge --reintegrate http://svn.example.com/repos/calc/branches/my-calc-branch
-- Zusammenführen der Unterschiede zwischen Projektarchiv-URLs in ».«:
D   integer.c
A   whole.c
U   .
```

Auf den ersten Blick sieht es gar nicht schlecht aus, jedoch ist es nicht das, was Sie und Sally erwartet hätten. Die Zusammenführung hat die letzte Version der Datei `integer.c` gelöscht (diejenige, die Sallys Änderungen beinhaltet) und blindlings Ihre neue Datei `whole.c` hinzugefügt – die ein Duplikat der *älteren* Version von `integer.c` ist. Das Endergebnis ist, dass durch die Zusammenführung Ihrer „Umbenennung“ auf dem Zweig mit dem Stamm Sallys jüngste Änderungen aus der letzten Revision entfernt wurden.

Es ist kein echter Datenverlust. Sallys Änderungen befinden sich noch immer in der Geschichte des Projektarchivs, allerdings mag es nicht sofort ersichtlich sein, dass es passiert ist. Die Lehre, die es aus dieser Geschichte zu ziehen gilt, lautet, dass Sie sehr vorsichtig mit dem Zusammenführen von Kopien und Umbenennungen zwischen Zweigen sein sollten, solange sich Subversion an dieser Stelle nicht verbessert hat.

Abblocken von Clients, die Zusammenführungen nicht ausreichend unterstützen

Wenn Sie gerade Ihren Server auf Subversion 1.5 oder größer umgestellt haben, besteht ein signifikantes Risiko, dass Subversion-Clients einer kleineren Version als 1.5 Ihre automatische Zusammenführungs-Verfolgung durcheinander bringen können. Warum? Wenn ein älterer Subversion-Client **svn merge** ausführt, modifiziert er nicht den Wert der Eigenschaft `svn:mergeinfo`. Obwohl die anschließende Übergabe das Ergebnis einer Zusammenführung ist, wird dem Projektarchiv nichts über die duplizierten Änderungen mitgeteilt – diese Information ist verloren. Wenn später Clients, die Zusammenführungsinformationen auswerten, automatische Zusammenführungen versuchen, werden Sie wahrscheinlich in alle möglichen Konflikte laufen, die durch wiederholte Zusammenführungen hervorgerufen wurden.

Wenn Sie und Ihr Team auf die Zusammenführungs-Verfolgung von Subversion angewiesen sind, sollten Sie Ihr Projektarchiv dergestalt konfigurieren, dass ältere Clients daran gehindert werden, Änderungen zu übergeben. Die einfache Methode hierfür ist es, den „Fähigkeiten“-Parameter im `start-commit` Hook-Skript zu untersuchen. Wenn der Client meldet, dass er mit `mergeinfo` umgehen kann, kann das Skript den Beginn der Übergabe erlauben. Wenn der Client diese Fähigkeit nicht meldet, wird die Übergabe abgelehnt. Beispiel 4.1, „Hook-Skript zum Start der Übertragung als Torwächter für die Zusammenführungs-Verfolgung“ zeigt ein Beispiel für ein solches Hook-Skript:

Beispiel 4.1. Hook-Skript zum Start der Übertragung als Torwächter für die Zusammenführungs-Verfolgung

```
#!/usr/bin/env python
import sys

# Dieser Start-Commit-Hook wird aufgerufen, bevor eine
# Subversion-Transaktion im Zuge einer Übergabe begonnen wird.
# Subversion führt diesen Hook aus, indem ein Programm (Skript,
# ausführbare Datei, Binärdatei, etc.) namens "start-commit" (für die
# diese Datei als Vorlage dient) mit den folgenden geordneten Argumenten
# aufgerufen wird:
#
#   [1] REPOS-PATH   (der Pfad zu diesem Projektarchiv)
#   [2] USER         (der authentisierte Anwender, der übergeben möchte)
#   [3] CAPABILITIES (eine vom Client durch Doppelpunkte getrennte
#                     Liste von Leistungsmerkmalen; siehe Anmerkung
#                     unten)

capabilities = sys.argv[3].split(':')
if "mergeinfo" not in capabilities:
  sys.stderr.write("Übertragungen von Clients, die keine"
                   "Zusammenführungs-Verfolgung unterstützen,"
                   "sind nicht erlaubt. Bitte auf Subversion 1.5 "
                   "oder neuer aktualisieren.\n")
  sys.exit(1)
sys.exit(0)
```

Für weitergehende Informationen zu Hook-Skripten, siehe nächsten Kapitel erfahren; siehe „Erstellen von Projektarchiv-Hooks“.

Das abschließende Wort zur Zusammenführungs-Verfolgung

Unter dem Strich bedeutet das, dass die Fähigkeit von Subversion zur Zusammenführungs-Verfolgung eine höchst komplexe interne Implementierung besitzt und die Eigenschaft `svn:mergeinfo` das einzige Fenster zu diesem Räderwerk ist. Da diese

Fähigkeit relativ neu ist, kann eine Anzahl von Randfällen und mögliche unerwartete Verhaltensweisen auftauchen.

Manchmal erscheint Mergeinfo auf Dateien, von denen Sie nicht erwartet hätten, dass sie durch eine Operation berührt worden wären. Manchmal wird Mergeinfo überhaupt nicht erzeugt, obwohl Sie es erwartet hätten. Darüberhinaus umgibt die Verwaltung der Mergeinfo-Metadaten eine ganze Menge von Systematiken und Verhalten, wie „explizite“ gegenüber „implizite“ Mergeinfo, „operative“ gegenüber „inoperative“ Revisionen, besondere Mechanismen von Mergeinfo-„Auslassung“ und sogar „Vererbung“ von Eltern- zu Kindverzeichnissen.

Wir haben uns entschieden, diese detaillierten Themen aus einer Reihe von Gründen nicht in diesem Buch zu behandeln. Erstens ist der Detaillierungsgrad für einen normalen Benutzer absolut erdrückend. Zweitens glauben wir, dass das Verständnis diese Konzepte für einen typischen Benutzer nicht unbedingt erforderlich sein *sollte* während Subversion sich verbessert; letztendlich werden sie als nervige Implementierugsdetails in den Hintergrund treten. Wenn Sie, nachdem dies gesagt ist, diese Dinge mögen, können Sie einen fantastischen Überblick in einer Arbeit nachlesen, die auf der Webseite von CollabNet veröffentlicht ist:http://www.collab.net/community/subversion/articles/merge-info.html.

Fürs Erste empfiehlt CollabNet, sich an die folgenden bewährten Praktiken zu halten, wenn Sie Fehler und merkwürdiges Verhalten bei automatischen Zusammenführungen vermeiden wollen:

- Wenden Sie für kurzlebige Arbeitszweige das Verfahren an, das in „Grundlegendes Zusammenführen“ beschrieben wird.
- Machen Sie Zusammenführungen langlebiger Release-Zweige (wie in „Verbreitete Verzweigungsmuster“ beschrieben) nur im Wurzelverzeichnis des Zweigs und nicht in Unterverzeichnissen.
- Machen Sie Zusammenführungen in Arbeitsverzeichnisse niemals mit einer Mischung aus Arbeitsrevisionsnummern oder „umgeschalteten“ Unterverzeichnissen (wie als Nächstes in „Zweige durchlaufen“ beschrieben). Das Ziel einer Zusammenführung sollte eine Arbeitskopie sein, die einen *einzigen* Ort zu einem einzelnen Zeitpunkt im Projektarchiv repräsentiert.
- Editieren Sie niemals direkt die Eigenschaft `svn:mergeinfo`; verwenden Sie **svn merge** mit der Option `--record-only`, um eine gewünschte Änderung an den Metadaten zu bewirken (wie in „Änderungen blockieren“ gezeigt).
- Stellen Sie jederzeit sicher, dass Sie vollständigen Lesezugriff auf die Quellen für die Zusammenführung haben und dass Ihre Ziel-Arbeitskopie keine dünn besetzten Verzeichnisse besitzt.

Zweige durchlaufen

Der Befehl **svn switch** überführt eine bestehende Arbeitskopie, so dass sie einen anderen Zweig repräsentiert. Obwohl dieser Befehl strenggenommen für die Arbeit mit Zweigen nicht notwendig ist, stellt er eine nette Abkürzung dar. In unserem früheren Beispiel haben Sie nach dem Anlegen Ihres eigenen privaten Zweigs eine frische Arbeitskopie des neuen Projektarchiv-Verzeichnisses ausgecheckt. Stattdessen können Sie Subversion einfach mitteilen, dass es Ihre Arbeitskopie von `/calc/trunk` ändern soll, um den neuen Ort des Zweigs widerzuspiegeln:

```
$ cd calc

$ svn info | grep URL
URL: http://svn.example.com/repos/calc/trunk

$ svn switch ^/calc/branches/my-calc-branch
U    integer.c
U    button.c
U    Makefile
Aktualisiert zu Revision 341.

$ svn info | grep URL
URL: http://svn.example.com/repos/calc/branches/my-calc-branch
```

„Das Umschalten“ einer Arbeitskopie ohne lokale Änderungen auf einen anderen Zweig hat zur Folge, dass die Arbeitskopie genau so aussieht, als sei das Verzeichnis frisch ausgecheckt worden. Es ist gewöhnlicherweise effizienter, diesen Befehl zu verwenden, da sich Zweige oftmals nur in kleinen Teilen unterscheiden. Der Server sendet nur die minimale Menge von

Änderungen, die notwendig sind, damit Ihre Arbeitskopie den Inhalt des Zweig-Verzeichnisses wiedergibt.

Der Befehl **svn switch** versteht auch die Option `--revision` (`-r`), so dass Sie nicht immer gezwungen sind, Ihre Arbeitskopie auf den `HEAD` des Zweigs zu setzen.

Natürlich sind die meisten Projekte komplizierter als unser `calc`-Beispiel und enthalten mehrere Unterverzeichnisse. Subversion-Benutzer wenden bei der Verwendung von Zweigen häufig einen bestimmten Algorithmus an:

1. Kopiere den vollständigen „Stamm“ des Projektes in ein neues Zweig-Verzeichnis.

2. Schalte nur einen *Teil* der Arbeitskopie vom Stamm auf den Zweig um.

In anderen Worten: Wenn ein Benutzer weiß, dass die Arbeit auf dem Zweig nur in einem bestimmten Unterverzeichnis stattfinden muss, verwendet er **svn switch** lediglich, um dieses Unterverzeichnis auf den Zweig zu bringen. (Manchmal schalten Benutzer sogar nur eine einzelne Datei auf den Zweig um!) Auf diese Art kann ein Benutzer für einen großen Teil der Arbeitskopie weiterhin normale Aktualisierungen auf dem „Stamm“ erhalten, wohingegen die umgeschalteten Teile unberührt bleiben (es sei denn, jemand übergibt etwas an den Zweig). Diese Möglichkeit fügt dem Konzept einer „gemischten Arbeitskopie“ eine völlig neue Dimension hinzu – Arbeitskopien können nicht nur eine Mischung unterschiedlicher Revisionen enthalten, sondern auch eine Mischung unterschiedlicher Projektarchiv-Orte.

Falls Ihre Arbeitskopie eine Anzahl umgeschalteter Unterverzeichnisse aus unterschiedlichen Projektarchiv-Orten enthält, funktioniert sie immer noch normal. Wenn Sie aktualisieren, erhalten Sie entsprechende Patches für jeden Unterbaum. Wenn Sie übergeben, werden Ihre lokalen Änderungen nach wie vor als eine einzelne atomare Änderung auf das Projektarchiv angewendet.

Während es normal ist, das eine Arbeitskopie eine Mischung unterschiedlicher Projektarchiv-Orte repräsentiert, ist darauf zu achten, dass all diese Orte sich innerhalb *desselben* Projektarchivs befinden. Subversion-Projektarchive können noch nicht miteinander kommunizieren; diese Möglichkeit ist für die Zukunft geplant. [6]

Umschalten und Aktualisierungen

Ist Ihnen aufgefallen, dass die Ausgaben von **svn switch** und **svn update** gleich aussehen? Der switch-Befehl ist tatsächlich eine Obermenge des update-Befehls.

Wenn Sie **svn update** aufrufen, fordern Sie das Projektarchiv auf, zwei Bäume zu vergleichen. Das Projektarchiv macht es und schickt eine Beschreibung der Unterschiede zurück an den Client. Der einzige Unterschied zwischen **svn switch** und **svn update** ist, dass letzterer Befehl stets zwei identische Projektarchiv-Pfade miteinander vergleicht.

Das heißt, falls Ihre Arbeitskopie `/calc/trunk` widerspiegelt, wird **svn update** automatisch Ihre Arbeitskopie von `/calc/trunk` mit `/calc/trunk` in der Revision `HEAD` vergleichen. Falls Sie Ihre Arbeitskopie auf einen Zweig umschalten, wird **svn switch** Ihre Arbeitskopie von `/calc/trunk` mit einem *anderen* Zweig-Verzeichnis in der `HEAD`-Revision vergleichen.

In anderen Worten: Eine Aktualisierung bewegt Ihre Arbeitskopie durch die Zeit. Eine Umschaltung bewegt Ihre Arbeitskopie durch die Zeit *und* den Raum.

Da **svn switch** eigentlich eine Variante von **svn update** ist, teilt es dasselbe Verhalten; irgendwelche lokalen Änderungen Ihrer Arbeitskopie bleiben erhalten, wenn neue Daten aus dem Projektarchiv ankommen.

Haben Sie sich jemals dabei ertappt, dass Sie (in Ihrer `/trunk`-Arbeitskopie) komplexe Änderungen gemacht haben und plötzlich feststellen: „Verdammt, diese Änderungen sollten auf einen eigenen Zweig!“ Eine gute Technik, um das zu bewerkstelligen, lässt sich in zwei Schritten zusammenfassen:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/newbranch \
```

[6] Sie *können* jedoch **svn switch** mit der Option `--relocate` verwenden, falls sich der URL Ihres Servers geändert hat, und Sie die bestehende Arbeitskopie nicht aufgeben wollen. Siehe svn switch (sw) für weitere Informationen und ein Beispiel.

```
      -m "Zweig 'newbranch' angelegt."
Revision 353 übertragen.
$ svn switch http://svn.example.com/repos/calc/branches/newbranch
Revision 353.
```

Der Befehl **svn switch** bewahrt wie **svn update** Ihre lokalen Änderungen. An dieser Stelle spiegelt Ihre Arbeitskopie den neu erzeugten Zweig wieder, und Ihr nächster Aufruf von **svn commit** wird Ihre Änderungen dorthin senden.

Tags

Ein weiterer verbreiteter Begriff in der Versionskontrolle ist ein *Tag*. Ein Tag ist lediglich eine „Momentaufnahme“ eines Projekts. In Subversion scheint dieses Konzept bereits überall vorhanden zu sein. Jede Revision im Projektarchiv ist genau das – eine Momentaufnahme des Dateisystems nach einer Übergabe.

Allerdings möchten Menschen häufig sprechendere Namen für Tags vergeben, wie etwa `release-1.0`. Und sie möchten Momentaufnahmen kleinerer Unterverzeichnisse des Dateisystems erstellen. Schließlich ist es nicht gerade einfach, sich daran zu erinnern, dass Release 1.0 einer Software ein bestimmtes Unterverzeichnis der Revision 4822 ist.

Erzeugen eines einfachen Tags

Wieder einmal hilft Ihnen **svn copy** bei der Arbeit. Wenn Sie eine Momentaufnahme von `/calc/trunk` machen wollen, genau so, wie es in der Revision `HEAD` aussieht, machen Sie davon eine Kopie:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/tags/release-1.0 \
      -m "Ein Tag für die Ausgabe 1.0 des 'calc' Projektes anlegen."

Revision 902 übertragen.
```

Dieses Beispiel geht davon aus, dass ein Verzeichnis `/calc/tags` bereits besteht. (Falls nicht, können Sie es mit **svn mkdir** erstellen.) Nach Abschluss der Kopie ist das neue Verzeichnis `release-1.0` für immer eine Momentaufnahme des Verzeichnisses `/trunk` in der Revision `HEAD` zum Zeitpunkt, an dem Sie die Kopie erstellt haben. Natürlich können Sie auch angeben, welche Revision Sie genau kopieren möchten, für den Fall, dass jemand anderes Änderungen an das Projekt übergeben haben könnte, während Sie nicht hingeschaut haben. Wenn Sie also wissen, dass Revision 901 von `/calc/trunk` genau die Momentaufnahme ist, die Sie möchten, können Sie sie mit der Option `-r 901` an den Befehl **svn copy** übergeben.

Moment mal: ist die Erstellung eines Tags nicht dasselbe Vorgehen wie bei der Erstellung eines Zweigs? Ja, es ist es tatsächlich. In Subversion gibt es keinen Unterschied zwischen einem Tag und einem Zweig. Beides sind gewöhnliche Verzeichnisse, die durch Kopieren erzeugt werden. Genauso wie bei Zweigen, ist der einzige Grund warum ein kopiertes Verzeichnis ein „Tag“ ist, weil *Menschen* sich entschieden haben, es so zu betrachten: Solange niemand etwas an das Verzeichnis übergibt, bleibt es für immer eine Momentaufnahme. Wenn jemand damit beginnt, etwas dorthin zu übergeben, wird es ein Zweig.

Wenn Sie ein Projektarchiv verwalten, gibt es zwei Ansätze für den Umgang mit Tags. Der erste Ansatz ist „Hände weg“: Als Vereinbarung im Projekt entscheiden Sie, wohin Sie Ihre Tags kopieren möchten; stellen Sie sicher, dass alle Benutzer wissen, wie sie ihre zu kopierenden Verzeichnisse behandeln sollen, d.h., stellen Sie sicher, dass sie nichts dorthin übergeben. Der zweite Ansatz ist etwas paranoider: Sie können eins der Zugriffskontrollskripte verwenden, die mit Subversion ausgeliefert werden, um zu verhindern, dass irgendjemand etwas anderes im Tag-Bereich macht, als dort neue Kopien zu erzeugen (siehe Kapitel 6, *Konfiguration des Servers*). Der paranoide Ansatz ist normalerweise nicht notwendig. Falls ein Benutzer versehentlich eine Änderung an ein Tag-Verzeichnis übergeben hat, können Sie die Änderung einfach rückgängig machen, wie im vorhergehenden Abschnitt beschrieben. Schließlich handelt es sich um Versionskontrolle!

Erzeugen eines komplexen Tags

Manchmal möchten Sie vielleicht eine „Momentaufnahme“ machen, die komplizierter ist als ein einzelnes Verzeichnis mit einer einzigen Revision.

Stellen Sie sich beispielsweise vor, Ihr Projekt sei viel größer als unser `calc` Beispiel: Nehmen wir an, es enthalte eine große Zahl von Unterverzeichnissen und viel mehr Dateien. Während Ihrer Arbeit könnte es sein, dass Sie sich entscheiden, eine Arbeitskopie anzulegen, die bestimmte Merkmale und Fehlerbehebungen beinhaltet. Sie können dies hinbekommen, indem Sie selektiv Dateien oder Verzeichnisse auf bestimmte Revisionen zurückdatieren (unter Verwendung von **svn update** mit der Option `-r`), indem Sie Dateien und Verzeichnisse auf bestimmte Zweige umschalten (mit **svn switch**) oder sogar, indem Sie ein paar lokale Änderungen vornehmen. Wenn Sie fertig sind, ist Ihre Arbeitskopie ein Mischmasch aus Projektarchiv-Quellen verschiedener Revisionen. Nach dem Testen wissen Sie jedoch, dass das genau die Kombination ist, die Sie mit einem Tag versehen möchten.

Nun ist es an der Zeit, eine Momentaufnahme zu machen. Einen URL auf einen anderen zu kopieren hilft hier nicht weiter. In diesem Fall möchten Sie eine Momentaufnahme der exakten Anordnung Ihrer Arbeitskopie machen und sie im Projektarchiv speichern. Glücklicherweise besitzt **svn copy** vier verschiedene Anwendungsfälle (über die Sie in Kapitel 9, *Die vollständige Subversion Referenz* nachlesen können), zu denen auch die Fähigkeit gehört, einen Arbeitskopie-Baum ins Projektarchiv zu kopieren:

```
$ ls
my-working-copy/

$ svn copy my-working-copy \
           http://svn.example.com/repos/calc/tags/mytag \
           -m "Ein Tag für den Zustand meines Arbeitsverzeichnisses anlegen."

Revision 940 übertragen.
```

Nun gibt es ein neues Verzeichnis im Projektarchiv, `/calc/tags/mytag`, das eine exakte Momentaufnahme Ihrer Arbeitskopie ist – gemischte Revisionen, URLs, lokale Änderungen, usw.

Andere Benutzer haben interessante Anwendungsfälle für diese Fähigkeit von Subversion gefunden. Manchmal gibt es Situationen, in denen Sie ein paar lokale Änderungen in Ihrer Arbeitskopie gemacht haben, die ein Mitarbeiter sehen soll. Statt **svn diff** aufzurufen und eine Patch-Datei zu versenden (die allerdings weder Änderungen an Verzeichnissen, symbolischen Links oder Eigenschaften beinhaltet), können Sie **svn copy** verwenden, um Ihre Arbeitskopie in einen privaten Bereich des Projektarchivs „abzulegen“. Ihr Mitarbeiter kann dann entweder eine exakte Kopie Ihrer Arbeitskopie auschecken oder **svn merge** verwenden, um genau Ihre Änderungen zu empfangen.

Obwohl dies eine nette Methode ist, schnell eine Momentaufnahme Ihrer Arbeitskopie anzulegen, sollten Sie beachten, dass es *keine* gute Vorgehensweise ist, einen Zweig zu erstellen. Die Erzeugung eines Zweigs sollte ein Ereignis für sich sein, wohingegen diese Methode die Erzeugung eines Zweigs mit zusätzlichen Änderungen an Dateien innerhalb einer einzelnen Revision verbindet. Das macht es später sehr schwer, eine einzelne Revisionsnummer als Verzweigungspunkt zu identifizieren.

Verwaltung von Zweigen

Sie haben mittlerweile vielleicht festgestellt, dass Subversion äußerst flexibel ist. Da Zweigen und Tags derselbe Mechanismus zugrundeliegt (Verzeichniskopien) und weil Zweige und Tags im normalen Dateisystem auftauchen, finden viele Leute Subversion einschüchternd. Es ist beinahe *zu* flexibel. In diesem Abschnitt machen wir einige Vorschläge, wie Sie Ihre Daten im Laufe der Zeit organisieren und verwalten können.

Aufbau des Projektarchivs

Es gibt einige empfohlene Standards, ein Projektarchiv zu organisieren. Die meisten Leute erzeugen ein `trunk`-Verzeichnis, um die Hauptlinie der Entwicklung aufzunehmen, ein `branches`-Verzeichnis für Zweig-Kopien und ein `tags`-Verzeichnis für Tag-Kopien. Falls ein Projektarchiv nur ein Projekt beinhaltet, werden oft diese Verzeichnisse auf der obersten Ebene angelegt:

```
/
  trunk/
  branches/
  tags/
```

Falls ein Projektarchiv mehrere Projekte enthält, teilen Administratoren das Projektarchiv üblicherweise nach den Projekten ein. Lesen Sie in „Planung der Organisation Ihres Projektarchivs“ mehr über „Projekt-Wurzelverzeichnisse“; hier ist ein Beispiel für ein solches Layout:

```
/
  paint/
    trunk/
    branches/
    tags/
  calc/
    trunk/
    branches/
    tags/
```

Natürlich ist es Ihnen freigestellt, diese verbreiteten Strukturen zu ignorieren. Sie können alle möglichen Variationen erzeugen, die am besten für Sie oder Ihr Team funktionieren. Denken Sie daran, dass es, wie auch immer Sie sich entscheiden, nicht für die Ewigkeit sein muss. Sie können jederzeit Ihr Projektarchiv umorganisieren. Da Zweige und Tags gewöhnliche Verzeichnisse sind, kann der Befehl **svn move** sie nach Belieben verschieben oder umbenennen. Die Umstrukturierung ist einfach eine Sache von serverseitigen Verschiebebefehlen. Wenn Ihnen der Aufbau des Projektarchivs nicht zusagt, jonglieren Sie einfach mit den Verzeichnissen herum.

Obwohl es einfach ist, Verzeichnisse zu verschieben, sollten Sie Rücksicht auf Ihre Benutzer nehmen. Ihr Jonglieren kann verwirrend für Benutzer mit bestehenden Arbeitskopien sein. Falls ein Benutzer eine Arbeitskopie eines bestimmten Projektarchiv-Verzeichnisses hat, könnte Ihre **svn move**-Operation den Pfad von der letzten Revision entfernen. Wenn der Benutzer beim nächsten Mal **svn update** aufruft, wird ihm mitgeteilt, dass die Arbeitskopie einen Pfad repräsentiere, der nicht mehr bestehe, so dass er gezwungen ist, mit **svn switch** auf den neuen Ort umzuschalten.

Lebensdauer von Daten

Eine weitere nette Eigenschaft des Subversion-Modells ist die Möglichkeit, Zweigen und Tags eine begrenzte Lebensdauer zu geben, so wie jedem anderen versionierten Objekt. Nehmen wir beispielsweise an, dass Sie letztendlich Ihre Arbeit auf dem persönlichen Zweig des `calc`-Projektes abschließen. Nachdem Sie all Ihre Änderungen zurück nach `/calc/trunk` gebracht haben, braucht Ihr privater Zweig nicht mehr herumzuliegen:

```
$ svn delete http://svn.example.com/repos/calc/branches/my-calc-branch \
             -m "Veralteten Zweig des Projekts calc gelöscht."

Revision 375 übertragen.
```

Nun ist Ihr Zweig verschwunden. Selbstverständlich ist er nicht wirklich verschwunden: das Verzeichnis fehlt einfach in der `HEAD`-Revision, so dass es niemanden mehr ablenken kann. Wenn Sie **svn checkout**, **svn switch** oder **svn list** verwenden, um sich eine frühere Revision anzusehen, werden Sie immer noch Ihren alten Zweig sehen.

Falls es nicht ausreichen sollte, im gelöschten Verzeichnis zu stöbern, können Sie es jederzeit wieder zurückholen. Das Wiederbeleben von Daten in Subversion ist sehr einfach. Falls ein gelöschtes Verzeichnis (oder eine gelöschte Datei) wieder nach `HEAD` gebracht werden soll, verwenden Sie einfach **svn copy** zum Kopieren aus der alten Revision:

```
$ svn copy http://svn.example.com/repos/calc/branches/my-calc-branch@374 \
           http://svn.example.com/repos/calc/branches/my-calc-branch \
           -m "my-calc-branch wiederhergestellt."

Revision 376 übertragen.
```

In unserem Beispiel hatte Ihr persönlicher Zweig eine relativ kurze Lebensdauer: Sie haben ihn vielleicht angelegt, um einen Fehler zu beseitigen oder eine neue Funktion einzubauen. Wenn Ihr Arbeitspaket abgeschlossen ist, kann auch der Zweig geschlossen werden. In der Softwareentwicklung ist es allerdings auch üblich, zwei „Haupt“-Zweige zu haben, die für lange Zeit nebeneinander bestehen. Es ist zum Beispiel an der Zeit, eine stabile Version des `calc`-Projektes zu veröffentlichen, und Sie wissen, dass es wohl noch ein paar Monate dauern wird, um Fehler aus der Software zu entfernen. Sie wollen weder, dass dem Projekt neue Funktionen hinzugefügt werden, noch möchten Sie alle Entwicklern auffordern, das Programmieren einzustellen. Stattdessen erstellen Sie einen „stabilen“ Zweig der Software, auf dem sich nicht viel verändern wird:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/stable-1.0 \
           -m "Stabilen Zweig für Projekt calc angelegt."

Revision 377 übertragen.
```

Nun können Entwickler die neuesten (oder experimentellen) Funktionen `/calc/trunk` hinzufügen, während Sie zum Grundsatz erklären, dass ausschließlich Fehlerbehebungen an `/calc/branches/stable-1.0` übergeben werden. Das heißt, während auf dem Stamm weitergearbeitet wird, überträgt jemand selektiv Fehlerbehebungen auf den stabilen Zweig. Selbst wenn die Software von hier bereits ausgeliefert worden ist, werden Sie diesen Zweig wahrscheinlich noch für eine lange Zeit pflegen – das heißt, so lange, wie Sie diese Auslieferung beim Kunden unterstützen werden. Wir werden das im nächsten Abschnitt näher erörtern.

Verbreitete Verzweigungsmuster

Es gibt zahlreiche unterschiedliche Anwendungsfälle für das Verzweigen und **svn merge**; dieser Abschnitt beschreibt die verbreitetesten.

Am häufigsten wird Versionskontrolle in der Softwareentwicklung verwendet, so dass wir an dieser Stelle kurz zwei der gebräuchlichsten Verzweigungs- und Zusammenführungsmuster vorstellen, die von Entwicklerteams benutzt werden. Falls Sie Subversion nicht in der Softwareentwicklung verwenden, können Sie den Abschnitt getrost überspringen. Falls Sie ein Softwareentwickler sind, der Versionskontrolle das erste Mal verwendet, sollten Sie gut aufpassen, da es sich bei diesen Mustern um bewährte Vorgehensweisen handelt, die von erfahrenen Menschen empfohlen werden. Diese Prozesse sind nicht spezifisch für Subversion; sie sind anwendbar auf alle Versionskontrollsysteme. Trotzdem mag es hilfreich sein, wenn sie anhand von Subversion erklärt werden.

Release-Zweige

Die meiste Software hat einen typischen Lebenszyklus: Erstellung, Test, Freigabe und wieder von vorne. Bei diesem Prozess gibt es zwei Probleme. Erstens müssen Entwickler neue Funktionen schreiben, während das Qualitätssicherungsteam sich Zeit zum Testen der vermeintlich stabilen Software nimmt. Die Arbeit kann allerdings nicht liegenbleiben während die Software getestet wird. Zweitens muss das Team fast immer ältere, bereits an den Kunden herausgegebene Software unterstützen; falls im neuesten Quelltext ein Fehler entdeckt wird, besteht der Fehler wahrscheinlich auch in der herausgegebenen Version. Die Kunden möchten dann eine Fehlerbehebung, ohne auf ein größeres, neues Release zu warten.

Hier kann Versionskontrolle helfen. Die typische Vorgehensweise ist wie folgt:

1. *Entwickler übergeben alles Neue an den Stamm.* Tägliche Änderungen werden an `/trunk` übergeben: neue Funktionen, Fehlerbehebungen usw.

2. *Der Stamm wird in einen „Release“-Zweig kopiert.* Wenn das Team der Auffassung ist, dass die Software reif für eine Freigabe ist (z.B. Release 1.0), kann `/trunk` nach `/branches/1.0` kopiert werden.

3. *Die Teams arbeiten parallel.* Ein Team beginnt, den Release-Zweig sorgfältig zu testen, während ein anderes Team mit der Arbeit (z.B. für Release 2.0) in `/trunk` fortfährt. Falls hier oder dort Fehler entdeckt werden sollten, werden die Fehlerbehebungen nach Bedarf hin oder her kopiert. Zu einem gegebenen Zeitpunkt hört jedoch sogar dieser Prozess auf. Der Zweig wird für die Abschlusstests vor der Freigabe „eingefroren“.

4. *Der Zweig wird markiert und freigegeben.* Nach dem Abschluss der Tests wird `/branches/1.0` als Momentaufnahme nach `/tags/1.0.0` kopiert. Das Tag wird paketiert und an den Kunden ausgeliefert.

5. *Der Zweig wird gepflegt.* Während die Arbeit für Version 2.0 in `/trunk` weitergeht, werden weiterhin Fehlerbehebungen von `/trunk` nach `/branches/1.0` portiert. Wenn sich ausreichend Fehlerbehebungen angesammelt haben, könnte sich das Management entschließen, ein Release 1.0.1 herauszugeben: `/branches/1.0` wird nach `/tags/1.0.1` kopiert, und das Tag wird paketiert und freigegeben.

Der gesamte Prozess wiederholt sich während die Software reift: Wenn die Arbeit an 2.0 fertig ist, wird ein neuer 2.0 Release-Zweig erstellt, getestet, markiert und schließlich freigegeben. Nach einigen Jahren füllt sich das Projektarchiv mit einer Anzahl von Release-Zweigen, die weiterhin „gepflegt“ werden, und einer Zahl von Tags, die den endgültigen, ausgelieferten Versionen entsprechen.

Funktions-Zweige

Ein *Funktions-Zweig* ist die Art von Zweig, wie er im Hauptbeispiel dieses Kapitels vorkam (der Zweig, auf dem Sie gearbeitet haben, während Sally auf `/trunk` arbeitete). Es ist ein vorübergehender Zweig, der angelegt wird, um an einer komplexen Änderung zu arbeiten, ohne `/trunk` zu stören. Anders als Release-Zweige (die vielleicht ewig gepflegt werden müssen), werden Funktions-Zweige erstellt, eine Zeit lang genutzt, zurück in den Stamm integriert und schließlich gelöscht. Sie haben einen zeitlich begrenzten Nutzen.

In Projekten gehen die Meinungen oft auseinander, wann der richtige Zeitpunkt zum Anlegen eines Funktions-Zweiges gekommen ist. Manche Projekte benutzen nie Funktions-Zweige: jeder darf Änderungen in `/trunk` übergeben. Der Vorteil hier ist, dass es einfach ist – niemand benötigt eine Schulung im Verzweigen und Zusammenführen. Der Nachteil ist, dass der Code oft instabil oder nicht nutzbar ist. Andere Projekte verwenden ausschließlich Zweige: Eine Änderung darf *niemals* direkt in `/trunk` übergeben werden. Selbst die trivialsten Änderungen werden auf einem kurzlebigen Zweig durchgeführt, sorgfältig geprüft und in den Stamm zurückgeführt. Danach wird der Zweig gelöscht. Dieses Vorgehen garantiert einen außerordentlich stabilen und nutzbaren Stamm, jedoch zum Preis eines erheblichen Prozessaufwands.

Die meisten Projekte bewegen sich irgendwo dazwischen. Gewöhnlich bestehen sie darauf, dass `/trunk` stets compilierfähig bleibt und Regressionstests besteht. Ein Funktions-Zweig wird nur dann benötigt, falls eine Änderung eine große Anzahl destabilisierender Übergaben erfordert. Eine gute Faustregel ist, diese Frage zu stellen: Wäre, falls ein Entwickler nach Tagen isolierter Entwicklung die große Änderung auf einmal übergäbe (so dass `/trunk` nie instabil würde), die Änderung zu umfangreich zum Überprüfen? Falls die Antwort auf diese Frage „ja“ lautet, sollte die Änderung auf einem Funktions-Zweig durchgeführt werden. Während der Entwickler schrittweise Änderungen in den Zweig übergibt, können sie auf einfache Weise von den Kollegen geprüft werden.

Schließlich stellt sich die Frage, wie ein Funktions-Zweig am besten mit dem Stamm „synchron“ gehalten werden kann während die Arbeit weitergeht. Wie wir vorher bereits bemerkten, besteht ein großes Risiko, wenn wochen- oder monatelang auf dem Zweig gearbeitet wird; währenddessen ändert sich auch der Stamm, so dass ein Punkt erreicht werden kann, an dem sich die beiden Entwicklungslinien so sehr unterscheiden, dass es zu einem Albtraum ausarten kann, den Zweig zurück auf den Stamm zu führen.

Diese Situation wird am besten vermieden, indem regelmäßig Änderungen vom Stamm in den Zweig eingearbeitet werden. Machen Sie es zur Gewohnheit: Arbeiten Sie wöchentlich die Änderungen der vergangenen Woche vom Stamm in den Zweig ein.

Irgendwann werden Sie dann bereit sein, den „synchronisierten“ Funktions-Zweig zurück in den Stamm zu führen. Hierzu arbeiten Sie ein letztes Mal die jüngsten Änderungen vom Stamm in den Zweig ein. Danach werden die letzten Versionen auf dem Stamm und dem Zweig, bis auf Ihre Änderungen auf dem Zweig, absolut gleich sein. Dann werden Sie den Zweig mit der Option `--reintegrate` wieder mit dem Stamm zusammenführen:

```
$ cd trunk-working-copy

$ svn update
Revision 1910.

$ svn merge --reintegrate http://svn.example.com/repos/calc/branches/mybranch
-- Zusammenführen der Unterschiede zwischen Projektarchiv-URLs in ».«:
U    real.c
U    integer.c
A    newdirectory
A    newdirectory/newfile
 U   .
…
```

Aus einem anderen Winkel betrachtet ist dieser wöchentliche Abgleich vom Stamm auf den Zweig analog zum Ausführen von **svn update** in einer Arbeitskopie, wobei das finale Zusammenführen **svn commit** in einer Arbeitskopie entspricht. *Ist* denn letztendlich eine Arbeitskopie nicht ein sehr flacher privater Zweig? Es ist ein Zweig, der nur eine Änderung gleichzeitig aufnehmen kann.

Lieferanten-Zweige

Besonders in der Softwareentwicklung haben die von Ihnen versionsverwalteten Daten oft einen engen Bezug zu Daten von anderen, oder sind vielleicht abhängig davon. Allgemein wird der Bedarf ihres Projektes erfordern, dass Sie bezüglich der externen Datenquelle so aktuell wie möglich bleiben, ohne dabei die Stabilität Ihres Projektes zu opfern. Dieses Szenario entfaltet sich immer dort, wo die von einer Gruppe erzeugten Informationen direkte Auswirkungen auf diejenigen Informationen hat, die von einer anderen Gruppe erstellt werden.

So könnte es sein, dass Softwareentwickler beispielsweise an einer Anwendung arbeiten, die die Bibliothek eines Drittanbieters benötigt. Subversion hat eine solche Abhängigkeit von der Bibliothek Apache Portable Runtime (APR) (siehe „Die Bibliothek Apache Portable Runtime“). Der Quelltext von Subversion hängt zur Gewährleistung der Portabilität von der APR-Bibliothek ab. In der frühen Phase der Entwicklung von Subversion hing das Projekt ziemlich nah am wechselnden API der APR, indem es immer die neueste Version des Quelltextes verwendete. Nun, da sowohl APR und Subversion gereift sind, versucht sich Subversion nur zu wohldefinierten Zeitpunkten mit dem APR-API zu synchronisieren, nämlich wenn dieses ausreichend getestet und stabil ist.

Falls nun Ihr Projekt von den Informationen anderer abhängt, können Sie diese Informationen auf mehrere Arten mit Ihren synchronisieren. Am umständlichsten ist es, wenn Sie mündliche oder schriftliche Anweisungen an alle Projektmitarbeiter ausgeben, dass sie sicherzustellen haben, stets über die für Ihr Projekt benötigten Versionen der Drittanbieter zu verfügen. Falls die Daten des Drittanbieters sich in einem Subversion-Projektarchiv befinden, können Sie auch mithilfe der Subversion-Externals-Definition bestimmte Versionen dieser Daten mit Ihrer eigenen Arbeitskopie verbinden (siehe „Externals-Definitionen“).

Allerdings möchten Sie von Zeit zu Zeit spezielle Anpassungen des Drittanbieter-Codes in Ihrem eigenen Versionskontrollsystem verwalten. Um auf unser Beispiel aus der Softwareentwicklung zurückzukommen, müssen Entwickler manchmal die Bibliothek der Drittanbieter für ihre Zwecke verändern. Diese Änderungen können neue Funktionalitäten oder Fehlerbehebungen beinhalten und werden nur solange intern verwaltet, bis sie eines Tages Teil einer offiziellen Auslieferung der Bibliothek werden. Es kann aber auch sein, dass diese Änderungen niemals an die Entwickler der Bibliothek zurückgegeben werden, sondern lediglich als spezielle Anpassungen bestehen, um die Bibliothek für Bedürfnisse der Softwareentwickler geeigneter zu machen.

Nun sind Sie in einer interessanten Situation: Ihr Projekt könnte seine Änderungen an den Daten von Drittanbietern auf getrennte Art und Weise verwalten, etwa in Form von Patch-Dateien oder als vollständig alternative Versionen. Jedoch wird so etwas schnell zu einem Albtraum, wenn es um die Pflege geht, da es ein Mechanismus benötigt wird, um diese Änderungen auf den Code des Drittanbieters anzuwenden und diese Anpassung bei jeder Folgelieferung zu wiederholen.

Die Lösung dieses Problems besteht in der Verwendung von *Lieferanten-Zweigen*. Ein Lieferanten-Zweig ist ein Verzeichnisbaum in Ihrem eigenen Versionskontrollsystem, der Informationen enthält, die von einem Drittanbieter – oder Lieferanten – bereitgestellt wird. Jede Version der Lieferantendaten, die Sie in Ihr Projekt aufnehmen wollen, wird *Zulieferung* genannt.

Lieferanten-Zweige bieten zwei Vorteile. Erstens, wird durch das Vorhalten der aktuellen Zulieferung in Ihrem eigenen Versionskontrollsystem sichergestellt, dass für Ihre Projektmitarbeiter stets die richtige Version der Lieferantendaten verfügbar ist. Sie erhalten die richtige Version automatisch beim Aktualisieren ihrer Arbeitskopien. Zweitens, da die Daten in Ihrem eigenen Subversion-Projektarchiv vorgehalten werden, können Sie dort auch Ihren Anpassungen speichern – es besteht keine Notwendigkeit mehr, Ihre Änderungen automatisch (oder schlimmer noch, manuell) in die Zulieferungen einzuarbeiten.

Allgemeines Vorgehen für die Verwaltung von Lieferanten-Zweigen

Die Verwaltung von Lieferanten-Zweigen funktioniert im Allgemeinen so: Zunächst erzeugen Sie ein übergeordnetes Hauptverzeichnis (etwa `/vendor`), um Lieferanten-Zweige aufzunehmen. Dann importieren Sie den Code des Drittanbieters in ein Unterverzeichnis des Hauptverzeichnisses. Anschließend kopieren Sie dieses Unterverzeichnis an die passende Stelle Ihres Hauptentwicklungszweigs (z.B. `/trunk`). Ihre lokalen Änderungen nehmen Sie stets im Hauptentwicklungszweig vor. Jede erneut veröffentlichte Version des von Ihnen verfolgten Codes pflegen Sie in den Lieferanten-Zweig ein und überführen die Änderungen nach `/trunk`, wobei eventuelle Konflikte zwischen Ihren lokalen Änderungen und dem Code des Zulieferers aufgelöst werden.

Ein Beispiel hilft, um dieses Vorgehen zu erklären. Wir gehen von einem Szenario aus, in dem Ihr Entwicklerteam ein Taschenrechner-Programm entwickelt, dass mit einer Bibliothek eines Drittanbieters für die Arithmetik mit komplexen Zahlen, namens libcomplex, verlinkt wird. Wir beginnen mit dem Anlegen des Lieferanten-Zweiges und dem Import der ersten Zulieferung. Wir nennen unser Verzeichnis für den Lieferanten-Zweig `libcomplex`, und die Lieferungen werden in einem Unterverzeichnis namens `current` abgelegt. Da **svn import** alle dazwischen liegenden Elternverzeichnisse erzeugt, können wir all diese Schritte mit einem einzigen Befehl bewerkstelligen:

```
$ svn import /path/to/libcomplex-1.0 \
             http://svn.example.com/repos/vendor/libcomplex/current \
             -m "Importing der ersten 1.0 Zulieferung"
…
```

Nun haben wir die aktuelle Version des Quelltextes von libcomplex in `/vendor/libcomplex/current`. Jetzt erzeugen wir ein Tag aus dieser Version (siehe „Tags“) und kopieren sie dann in den Hauptentwicklungszweig. Unsere Kopie erzeugt ein neues Verzeichnis `libcomplex` im bestehenden `calc` Projektverzeichnis. In dieser kopierten Version der Lieferantendaten werden wir unsere Anpassungen vornehmen:

```
$ svn copy http://svn.example.com/repos/vendor/libcomplex/current  \
           http://svn.example.com/repos/vendor/libcomplex/1.0      \
           -m "Tag libcomplex-1.0"
…
$ svn copy http://svn.example.com/repos/vendor/libcomplex/1.0  \
           http://svn.example.com/repos/calc/libcomplex        \
           -m "libcomplex-1.0 in den Huptzweig bringen"
…
```

Wir checken nun den Hauptzweig unseres Projektes aus – der nun eine Kopie der ersten Zulieferung enthält – und fangen damit an, den Quelltext von libcomplex anzupassen. Ehe wir uns versehen, ist unsere angepasste Version von libcomplex vollständig in unser Taschenrechner-Programm integriert.[7]

Ein paar Wochen später veröffentlichen die Entwickler von libcomplex eine neue Version ihrer Bibliothek – Version 1.1 – die die Funktionalität enthält, die wir dringend benötigen. Wir möchten die neue Version verwenden, ohne jedoch unsere Anpassungen zu verlieren, die wir in der bestehenden Version vorgenommen haben. Unterm Strich möchten wir die bestehende Baseline-Version libcomplex 1.0 durch eine Kopie von libcomplex 1.1 ersetzen und die vorher gemachten Anpassungen an dieser Bibliothek erneut auf die neue Version anwenden. Tatsächlich gehen wir das Problem allerdings aus der anderen Richtung an, indem wir die Änderungen an libcomplex zwischen Version 1.0 und 1.1 in unsere angepasste Kopie

[7] Und er ist natürlich völlig frei von Fehlern!

einpflegen.

Um diesen Wechsel auf die neue Version durchzuführen, checken wir eine Kopie des Lieferanten-Zweigs aus und ersetzen den Code im Verzeichnis `current` mit dem neuen Quelltext von libcomplex 1.1. Wir kopieren im wahrsten Sinne des Wortes die neuen Dateien über die bestehenden, indem wir etwa das Archiv von libcomplex 1.1 in das bestehende Verzeichnis entpacken. Das Ziel ist, dass das Verzeichnis `current` nur den Code von libcomplex 1.1 enthält, und dass dieser Code unter Versionskontrolle steht. Oh, und wir wollen, dass das alles mit der geringsten Störung an der Versionskontroll-Historie passiert.

Nachdem wir den 1.0 Code mit dem 1.1 Code ersetzt haben, wird uns **svn status** sowohl Dateien mit lokalen Änderungen als auch einige unversionierte Dateien anzeigen. Wenn wir das getan haben, was wir tun sollten, sind die unversionierten Dateien nur die mit Version 1.1 von libcomplex hinzugekommenen neuen Dateien – wir rufen für diese **svn add** auf, um sie unter Versionskontrolle zu bringen. Falls der Code von 1.1 bestimmte Dateien nicht mehr beinhaltet, die noch im Baum von 1.0 vorhanden waren, kann es schwierig sein, sie zu identifizieren; Sie müssten die beiden Bäume mit einem externen Werkzeug vergleichen und dann mit **svn delete** Dateien entfernen, die in 1.0 jedoch nicht in 1.1 vorhanden sind. (Es könnte ebenso in Ordnung sein, diese Dateien ungenutzt beizubehalten!) Sobald letztendlich unsere Arbeitskopie von `current` nur den Code von libcomplex 1.1 enthält, übergeben wir die Änderungen, die uns hierher gebracht haben.

Unser `current`-Zweig enthält nun die neue Zulieferung. Wir erzeugen nun ein Tag 1.1 (genauso, wie wie es mit der Zulieferung 1.0 gemacht haben) und arbeiten dann die Unterschiede zwischen dem Tag der vorherigen Version und der neuen aktuellen Version in unseren Hauptentwicklungszweig ein:

```
$ cd working-copies/calc
$ svn merge ^/vendor/libcomplex/1.0      \
            ^/vendor/libcomplex/current  \
            libcomplex
… # alle Konflikte zwischen ihren und unseren Änderungen auflösen
$ svn commit -m "merging libcomplex-1.1 into the main branch"
…
```

Im trivialen Fall würde die neue Version der Drittanbieter-Bibliothek aus der Datei- und Verzeichnisperspektive genau so aussehen wie die vorherige Version. Keine der libcomplex Dateien wäre gelöscht, umbenannt oder an einen anderen Ort verschoben worden – die neue Version würde gegenüber der vorherigen lediglich textuelle Änderungen enthalten. In einer vollkommenen Welt würden sich unsere Anpassungen sauber in die neue Version einfügen, ganz ohne Komplikationen oder Konflikte.

Allerdings gestalten sich die Dinge nicht immer so einfach, und tatsächlich ist es normal, dass sich Quelltext-Dateien zwischen Software-Veröffentlichungen verschieben. Das verkompliziert das Vorgehen, um sicherzustellen, dass unsere Anpassungen für die neue Version immer noch gültig sind, und es kann schnell passieren, dass wir in eine Situation gelangen, in der wir unsere Anpassungen manuell in die neue Version einpflegen müssen. Sobald Subversion die Geschichte einer gegebenen Quelltext-Datei kennt – inklusive aller früheren Orte – ist das Vorgehen des Einpflegens in eine neue Version der Bibliothek recht einfach. Allerdings sind wir dafür verantwortlich, Subversion mitzuteilen, wie sich die Organisation des Quelltextes zwischen den Zulieferungen geändert hat.

svn_load_dirs.pl

Zulieferungen, die mehr als ein paar Löschungen, Hinzufügungen und Verschiebungen beinhalten, verkomplizieren das Vorgehen bei der Aktualisierung auf neuere Versionen der Drittanbieter-Daten. Aus diesem Grund stellt Subversion das Skript **svn_load_dirs.pl** zur Verfügung, das Sie dabei unterstützt. Dieses Skript automatisiert die zum Importieren notwendigen Schritte, die wir beim Vorgehen zur allgemeinen Verwaltung von Lieferanten-Zweigen erwähnten, um zu gewährleisten, dass es dabei zu möglichst wenig Fehlern kommt. Sie werden zwar immer noch dafür verantwortlich sein, mit den Zusammenführungs-Befehlen die neuen Versionen der Drittanbieter-Daten in Ihren Hauptentwicklungszweig einzupflegen, jedoch kann Ihnen **svn_load_dirs.pl** dabei helfen, diesen Punkt schneller und leichter zu erreichen.

Kurz gesagt ist **svn_load_dirs.pl** eine Verbesserung von **svn import** mit folgenden wichtigen Eigenschaften:

- Es kann jederzeit aufgerufen werden, um ein bestehendes Verzeichnis im Projektarchiv exakt mit einem externen Verzeichnis abzugleichen, wobei alle notwendigen Hinzufügungen und Löschungen ausgeführt werden und darüberhinaus

noch optionale Verschiebungen.

- Es kümmert sich um komplizierte Abfolgen von Operationen zwischen denen Subversion eine eingeschobene Übergabe erforderlich macht – etwa vor dem zweifachen Umbenennen einer Datei oder eines Verzeichnisses.
- Vom frisch importierten Verzeichnis wird optional ein Tag angelegt.
- Es legt optional beliebige Eigenschaften für Dateien und Verzeichnisse an, deren Name einem regulären Ausdruck entspricht.

svn_load_dirs.pl benötigt drei zwingend erforderliche Argumente. Das erste Argument ist der URL zum Basis-Subversion-Verzeichnis, in dem gearbeitet wird. Dieses Argument wird gefolgt von dem URL – relativ zum ersten Argument – wohin die aktuelle Zulieferung importiert werden soll. Schließlich gibt das dritte Argument an, aus welchem lokalen Verzeichnis importiert werden soll. In unserem vorigen Beispiel würde ein typischer Aufruf von **svn_load_dirs.pl** wie folgt aussehen:

```
$ svn_load_dirs.pl http://svn.example.com/repos/vendor/libcomplex \
                   current                                        \
                   /path/to/libcomplex-1.1
…
```

Sie können **svn_load_dirs.pl** mitteilen, dass Sie ein Tag von der neuen Zulieferung anlegen möchten, indem Sie die Option `-t` gefolgt von einem Tag-Namen übergeben. Dieser Tag-Name ist auch ein URL relativ zum ersten Argument des Programms.

```
$ svn_load_dirs.pl -t libcomplex-1.1                              \
                   http://svn.example.com/repos/vendor/libcomplex \
                   current                                        \
                   /path/to/libcomplex-1.1
…
```

Wenn Sie **svn_load_dirs.pl** aufrufen, untersucht es den Inhalt Ihre existierenden Zulieferung „current“ und vergleicht sie mit der vorgeschlagenen neuen Zulieferung. Im trivialen Fall werden keine Dateien ausschließlich in einer und nicht in der anderen Zulieferung vorhanden sein, so dass das Skript den Import ohne Probleme durchführt. Falls sich jedoch zwischen den Versionen Unterschiede in der Dateistruktur ergeben sollten, fragt **svn_load_dirs.pl** nach, wie die Unterschiede aufgelöst werden sollen. So haben Sie zum Beispiel die Möglichkeit, dem Skript mitzuteilen, dass die Datei `math.c` aus Version 1.0 von libcomplex in der Version 1.1 von libcomplex in `arithmetic.c` umbenannt wurde. Alle Diskrepanzen, die sich nicht durch Verschiebungen erklären lassen, werden als normale Löschungen und Hinzufügungen behandelt.

Das Skript akzeptiert auch eine gesonderte Konfigurationsdatei, in der Eigenschaften auf Dateien und Verzeichnisse gesetzt werden können, deren Name einem regulären Ausdruck entspricht und dem Projektarchiv *hinzugefügt* werden. Diese Konfigurationsdatei wird **svn_load_dirs.pl** mit der Option `-p` bekanntgegeben. Jede Zeile der Konfigurationsdatei ist eine durch Leerraum begrenzte Menge aus zwei oder vier Werten: ein regulärer Ausdruck wie in Perl, zu dem der entsprechende Pfad passen muss, ein Schlüsselwort zur Kontrolle (entweder `break` oder `cont`) und optional ein Eigenschafts-Name und ein Wert.

```
\.png$              break   svn:mime-type   image/png
\.jpe?g$            break   svn:mime-type   image/jpeg
\.m3u$              cont    svn:mime-type   audio/x-mpegurl
\.m3u$              break   svn:eol-style   LF
.*                  break   svn:eol-style   native
```

Für jeden hinzugefügten Pfad der dem regulären Ausdruck einer Zeile entspricht, werden die Eigenschafts-Änderungen der Reihe nach durchgeführt, es sei denn, das Kontroll-Schlüsselwort ist `break` (was bedeutet, dass keine weiteren Eigenschafts-

Änderungen für diesen Pfad durchgeführt werden sollen). Falls das Kontroll-Schlüsselwort `cont` ist – eine Abkürzung für `continue` (fortfahren) – wird mit der nächsten Zeile der Konfigurationsdatei fortgefahren.

Jeglicher Leerraum im regulären Ausdruck, im Namen der Eigenschaft oder im Wert der Eigenschaft muss entweder mit einfachen oder doppelten Anführungszeichen umgeben werden. Anführungszeichen, die nicht zum Umfassen von Leerraum verwendet werden, können mit einem vorangestellten umgekehrten Schrägstrich (\) maskiert werden. Der umgekehrte Schrägstrich maskiert nur Anführungszeichen beim Lesen der Konfigurationsdatei, darum sollten Sie darüberhinaus keine weiteren Zeichen maskieren.

Zusammenfassung

Wir haben in diesem Kapitel sehr viel durchgenommen. Wir haben die Konzepte hinter Tags und Zweigen besprochen und gezeigt, wie Subversion diese Konzepte durch das Kopieren von Verzeichnissen mit dem Befehl **svn copy** umsetzt. Wir zeigten, wie mit **svn merge** Änderungen von einem Zweig in einen anderen überführt werden können oder fehlerhafte Änderungen rückgängig gemacht werden. Wir besprachen, wie mit **svn switch** Arbeitskopien aus verschiedenen Projektarchiv-Quellen erstellt werden können. Und wir sprachen darüber, wie Zweige in einem Projektarchiv verwaltet werden können.

Erinnern Sie sich an das Mantra von Subversion: Zweige und Tags sind billig. Scheuen Sie nicht, sie bei Bedarf zu nutzen!

Als eine hilfreiche Erinnerung an die besprochenen Operationen sei hier noch einmal eine Referenztabelle angeführt, die Sie benutzen können, während Sie damit beginnen, Zweige zu verwenden.

Tabelle 4.1. Befehle zum Verzweigen und Zusammenführen

Aktion	Befehl
Erstellung eines Zweigs oder eines Tags	`svn copy URL1 URL2`
Umschalten einer Arbeitskopie auf einen Zweig oder ein Tag	`svn switch URL`
Synchronisierung eines Zweigs mit dem Stamm	`svn merge trunkURL; svn commit`
Anzeige der Zusammenführungs-Geschichte oder in Frage kommender Änderungsmengen	`svn mergeinfo SOURCE TARGET`
Zurückführen des Zweigs in den Stamm	`svn merge --reintegrate branchURL; svn commit`
Einarbeiten einer bestimmten Änderung	`svn merge -c REV URL; svn commit`
Einarbeiten einer Serie von Änderungen	`svn merge -r REV1:REV2 URL; svn commit`
Eine Änderung für das automatische Zusammenführen blockieren	`svn merge -c REV --record-only URL; svn commit`
Vorschau einer Zusammenführung	`svn merge URL --dry-run`
Verwerfen des Ergebnisses einer Zusammenführung	`svn revert -R .`
Etwas aus der Geschichte wiederbeleben	`svn copy URL@REV localPATH`
Eine übergebene Änderung rückgängig machen	`svn merge -c -REV URL; svn commit`
Anzeige der Geschichte unter Berücksichtigung von Zusammenführungen	`svn log -g; svn blame -g`
Erzeugen eines Tags aus einer Arbeitskopie	`svn copy . tagURL`
Einen Zweig oder ein Tag verschieben	`svn move URL1 URL2`
Einen Zweig oder ein Tag löschen	`svn delete URL`

Kapitel 5. Verwaltung des Projektarchivs

Das Subversion-Projektarchiv ist die zentrale Lagerhalle für Ihre versionierten Daten. In dieser Rolle kann es sich aller Liebe und Zuneigung des Administrators gewiss sein. Obwohl das Projektarchiv an sich im Allgemeinen wenig Pflege erfordert, ist es wichtig, zu wissen, wie es angemessen konfiguriert und gepflegt wird, um etwaige Probleme zu vermeiden und bestehende Probleme sicher aufzulösen.

In diesem Kapitel werden wir erörtern, wie ein Subversion-Projektarchiv aufgesetzt und konfiguriert wird. Wir werden auch über die Projektarchiv-Pflege reden und Beispiele dafür geben, wann und wie die mit Subversion mitgelieferten Werkzeuge **svnlook** und **svnadmin** verwendet werden. Wir werden einige verbreitete Fragen und Fehler besprechen und Vorschläge unterbreiten, wie die Daten im Projektarchiv organisiert werden können.

Falls Sie vorhaben, das Projektarchiv lediglich in der Rolle eines Benutzers zu verwenden, der seine Daten unter Versionskontrolle stellen möchte (d.h. über einen Subversion-Client), können Sie dieses Kapitel vollständig überspringen. Wenn Sie jedoch ein Subversion-Projektarchiv-Administrator sind oder werden wollen, [1] dann ist dieses Kapitel für Sie gemacht.

Das Subversion Projektarchiv, Definition

Bevor wir uns in das weite Feld der Projektarchiv-Verwaltung begeben, wollen wir definieren, was ein Projektarchiv ist. Wie sieht es aus? Wie fühlt es sich an? Trinkt es seinen Tee heiß oder mit Eis, gesüßt oder mit Zitrone? Als Administrator wird von Ihnen erwartet, dass Sie den Aufbau eines Projektarchivs sowohl auf der physischen Betriebssystemebene verstehen – wie sich ein Subversion-Projektarchiv aussieht und wie es sich gegenüber Nicht-Subversion-Werkzeugen verhält – als auch aus einer logischen Perspektive – wie Daten *innerhalb* des Projektarchivs repräsentiert werden.

Mit den Augen eines typischen Dateibrowsers (wie Windows Explorer) oder von kommandozeilenorientierten Dateisystem-Navigationswerkzeugen ist das Subversion-Projektarchiv bloß ein weiteres gewöhnliches Verzeichnis voller Zeugs. Es gibt einige Unterverzeichnisse mit Konfigurationsdateien, die für Menschen lesbar sind, einige Unterverzeichnisse mit weniger lesbaren Dateien usw. Wie in anderen Bereichen des Subversion-Designs, wird auch hier viel Wert auf Modularität gesetzt, und hierarchische Organisation wird vollgepfropftem Chaos vorgezogen. Ein flacher Blick in ein typisches Projektarchiv aus praxisbezogener Sicht reicht aus, um die grundlegenden Komponenten des Projektarchivs zu offenbaren.

```
$ ls repos
conf/  db/  format  hooks/  locks/  README.txt
```

Hier ist ein schneller, oberflächlicher Überblick über das, was Sie in diesem Verzeichnislisting sehen. (Verzetteln Sie sich nicht mit den Fachausdrücken – eine detaillierte Behandlung dieser Komponenten erfolgt an anderer Stelle in diesem und in anderen Kapiteln.)

conf
: Ein Verzeichnis, das Konfigurationsdateien enthält.

dav
: Ein Verzeichnis, das bereitgestellt wurde, damit `mod_dav_svn` seine privaten Verwaltungsdaten dort ablegen kann

db
: Der Datenspeicher für all Ihre versionierten Daten

format
: Eine Datei, die eine einzelne Ganzzahl beinhaltet, die die Version der Projektarchiv-Struktur angibt

[1] Das hört sich vielleicht prestigeträchtig und hochmütig an, doch wir meinen lediglich jeden, der an diesem mysteriösen Bereich hinter der Arbeitskopie interessiert ist, wo sich alle Daten befinden.

hooks
: Ein Verzeichnis voller Hook-Skript-Vorlagen (und den eigentlichen Hook-Skripten, sofern Sie welche installiert haben)

locks
: Ein Verzeichnis für die Sperrdateien des Subversion-Projektarchivs, die benutzt werden, um die Zugreifenden auf das Projektarchiv zu verfolgen

README.txt
: Eine Datei, deren Inhalt die Leser darüber informiert, dass sie in ein Subversion-Projektarchiv schauen

Vor Subversion 1.5 besaß die Projektarchiv-Struktur auf der Festplatte ein Unterverzeichnis `dav`. `mod_dav_svn` verwendete dieses Verzeichnis, um Informationen über WebDAV *Aktivitäten* zu speichern – Abbildungen von High-Level WebDAV Protokollkonzepten auf Subversion Übergabe-Transaktionen. Subversion 1.5 änderte dieses Verhalten, indem das Eigentum über das Aktivitäten-Verzeichnis und die Möglichkeit, dessen Ort zu konfigurieren, an `mod_dav_svn` selbst übertragen wurde. Nun benötigen neue Projektarchive nicht unbedingt ein Unterverzeichnis `dav`, es sei denn, `mod_dav_svn` wird verwendet und wurde nicht dergestalt konfiguriert, dass seine Aktivitäten-Datenbank an einem anderen Ort abgelegt wird. Siehe „Anweisungen“ für weitere Informationen.

Selbstverständlich wird diese sonst so unauffällige Ansammlung aus Dateien und Verzeichnissen, wenn auf sie über die Subversion-Bibliotheken zugegriffen wird, eine Implementierung eines virtuellen, versionierten Dateisystems, vollständig mit anpassbaren Ereignis-Triggern. Dieses Dateisystem hat seine eigene Auffassung von Verzeichnissen und Dateien, sehr ähnlich den Auffassungen, die echte Dateisysteme (wie NTFS, FAT32, ext3 usw.) von solchen Dingen haben. Aber dies ist ein spezielles Dateisystem – es hängt diese Dateien und Verzeichnisse an Revisionen und hält alle Änderungen, die Sie daran vorgenommen haben, sicher abgespeichert und für immer abrufbereit. Hier lebt die Gesamtheit Ihrer versionierten Daten.

Strategien für die Verwendung eines Projektarchivs

Größtenteils wegen der Einfachheit des Gesamtentwurfs des Subversion-Projektarchivs und der ihm zugrunde liegenden Technik, ist die Erstellung und Konfiguration eines Projektarchivs eine ziemlich unkomplizierte Aufgabe. Es gibt einige Entscheidungen, die Sie im Vorfeld treffen sollten, jedoch sind die eigentlichen Arbeitsschritte für die Einrichtung eines Subversion-Projektarchivs recht einfach und neigen zur stupiden Fleißarbeit, falls Sie mehrere davon aufzusetzen haben.

Einige Dinge, die Sie jedoch im Vorfeld sorgfältig prüfen sollten, sind:

- Welche Art von Daten sollen in Ihrem Projektarchiv (oder Projektarchiven) abgelegt werden, und wie sind diese Daten organisiert?
- Wo soll Ihr Projektarchiv untergebracht werden, und wie soll darauf zugegriffen werden?
- Welche Art von Zugriffskontrolle und Ereignisbenachrichtigung benötigen Sie?
- Welche der verfügbaren Datenspeicherungsarten möchten Sie verwenden?

In diesem Abschnitt werden wir versuchen, Ihnen beim Beantworten dieser Fragen zu helfen.

Planung der Organisation Ihres Projektarchivs

Obwohl Subversion Ihnen erlaubt, versionierte Dateien und Ordner ohne Informationsverlust hin und her zu verschieben und sogar Methoden anbietet, um versionierte Geschichte von einem Projektarchiv in ein anderes zu verschieben, kann das ziemlich den Arbeitsablauf derjenigen stören, die oft auf das Projektarchiv zugreifen und gewisse Dinge an bestimmten Orten erwarten. Bevor Sie ein neues Projektarchiv erstellen, sollten Sie also versuchen, ein wenig in die Zukunft zu schauen; planen Sie weitsichtig, bevor Sie Ihre Daten unter Versionskontrolle stellen. Durch die vorzeitige gewissenhafte „Anlage“ Ihres Projektarchivs oder mehrerer Projektarchive können Sie viel künftigen Kopfschmerz vermeiden.

Nehmen wir an, Sie seien als Projektarchiv-Administrator für die Versionskontrollsysteme mehrerer Projekte zuständig. Ihre erste Entscheidung ist, ob Sie ein einzelnes Projektarchiv für mehrere Projekte verwenden, jedem Projekt sein eigenes

Projektarchiv geben oder einen Kompromiss aus diesen beiden Lösungen haben wollen.

Ein einzelnes Projektarchiv für mehrere Projekte zu verwenden, hat einige Vorteile, am offensichtlichsten ist der vermiedene doppelte Verwaltungsaufwand. Ein einzelnes Projektarchiv bedeutet, dass es nur einen Satz Hook-Programme, ein Ding zum routinemäßigen Sichern, ein Ding für einen Auszug und zum anschließenden Laden nach einer inkompatiblen neuen Version von Subversion gibt usw. Sie können Daten auch einfach zwischen Projekten verschieben, ohne historische Versionierungsinformationen zu verlieren.

Der Nachteil bei der Verwendung eines einzelnen Projektarchivs ist, dass unterschiedliche Projekte auch unterschiedliche Anforderungen hinsichtlich der Projektarchiv-Ereignis-Trigger haben, wie etwa Benachrichtigungs-E-Mails bei Commits an unterschiedliche Verteiler, oder unterschiedliche Definitionen dazu, was eine berechtigte Übergabe ist und was nicht. Das sind natürlich keine unüberwindbaren Probleme – es bedeutet nur, dass all Ihre Hook-Skripte die Struktur Ihres Projektarchivs beachten müssen, anstatt davon auszugehen, dass das gesamte Projektarchiv von einer einzelnen Gruppe zugeordnet ist. Beachten Sie auch, dass Subversion Versionsnummern verwendet, die global für das gesamte Projektarchiv gelten. Obwohl diese Nummern keine Zauberkräfte haben, mögen manche Zeitgenossen es trotzdem nicht, dass, obwohl in letzter Zeit keine Änderungen in ihrem Projekt durchgeführt worden sind, die jüngste Versionsnummer im Projektarchiv ständig höher wird, weil andere Projekte fleißig neue Revisionen hinzufügen.[2]

Es kann auch eine Lösung in der Mitte gewählt werden. Beispielsweise können Projekte danach geordnet werden, wie stark sie miteinander verbunden sind. Sie könnten ein paar Projektarchive haben, die jeweils eine handvoll Projekte beherbergen. Auf diese Art können Projekte, die wahrscheinlich gemeinsame Daten verwenden wollen, dies auch einfach bewerkstelligen, und wenn dem Projektarchiv neue Versionen hinzugefügt werden, wissen die Entwickler wenigstens, dass diese neuen Revisionen zumindest entfernt eine Beziehung zu jedem Benutzer dieses Projektarchivs haben.

Nachdem Sie entschieden haben, wie Sie Ihre Projekte in Projektarchive aufteilen, möchten Sie sich nun vielleicht Gedanken darüber machen, welche Verzeichnishierarchien Sie im Projektarchiv anlegen wollen. Da Subversion zum Verzweigen und Etikettieren reguläre Verzeichniskopien verwendet (siehe Kapitel 4, *Verzweigen und Zusammenführen*), empfiehlt die Subversion-Gemeinschaft, dass Sie einen Ort im Projektarchiv für jedes *Projekt-Wurzelverzeichnis* wählen – das oberste Verzeichnis, das Daten für Ihr Projekt enthält – und hierunter dann drei Unterverzeichnisse anlegen: `trunk`, das Verzeichnis, in dem die Hauptentwicklung stattfindet, `branches`, zur Aufnahme verschiedener Zweige der Hauptentwicklungslinie und `tags`, als Sammlung von Momentaufnahmen des Verzeichnisbaums, die erzeugt, vielleicht gelöscht, jedoch nie verändert werden.[3]

Ihr Projektarchiv könnte z.B. so aussehen:

```
/
   calc/
      trunk/
      tags/
      branches/
   calendar/
      trunk/
      tags/
      branches/
   spreadsheet/
      trunk/
      tags/
      branches/
   …
```

Beachten Sie, dass es unerheblich ist, wo in Ihrem Projektarchiv sich das Wurzelverzeichnis Ihres Projektes befindet. Falls Sie nur ein Projekt pro Projektarchiv haben, ist der logische Ort für das Wurzelverzeichnis des Projektes das Wurzelverzeichnis des zum Projekt gehörenden Projektarchivs. Falls Sie mehrere Projekte haben, möchten Sie diese vielleicht innerhalb des Projektarchivs gruppieren, indem Sie Projekte ähnlichen Zwecks in demselben Unterverzeichnis unterbringen oder sie vielleicht nur alphabetisch gruppieren. Eine solche Anordnung könnte so aussehen:

[2] Ob es an Ignoranz oder an schlecht überlegten Konzepten zur Erstellung berechtigter Metriken für die Software-Entwicklung liegt, ist es dumm, Angst vor globalen Revisionsnummern zu haben, und es ist deshalb *kein* Kriterium, das Sie heranziehen sollten, wenn Sie abwägen, wie Sie Ihre Projekte und Projektarchive anlegen wollen.

[3] Das Trio `trunk`, `tags` und `branches` wird manchmal als „die TTB-Verzeichnisse“ bezeichnet.

```
/
   utils/
      calc/
         trunk/
         tags/
         branches/
      calendar/
         trunk/
         tags/
         branches/
      …
   office/
      spreadsheet/
         trunk/
         tags/
         branches/
      …
```

Legen Sie Ihr Projektarchiv so an, wie es Ihnen am besten passt. Subversion erwartet oder erzwingt keine bestimmte Anordnung – für Subversion ist und bleibt ein Verzeichnis ein Verzeichnis. Letztendlich sollten Sie für ein Projektarchiv eine Struktur wählen, die den Bedürfnissen der Leute gerecht wird, die an den Projekten arbeiten, die dort untergebracht sind.

Der Vollständigkeit halber erwähnen wir noch eine weitere, verbreitete Anordnung. Bei dieser Anordnung befinden sich die Verzeichnisse `trunk`, `tags` und `branches` im Wurzelverzeichnis des Projektarchivs und die Projekte in Unterverzeichnissen davon:

```
/
   trunk/
      calc/
      calendar/
      spreadsheet/
      …
   tags/
      calc/
      calendar/
      spreadsheet/
      …
   branches/
      calc/
      calendar/
      spreadsheet/
      …
```

An dieser Anordnung ist zwar nichts verkehrt, allerdings könnte es für Ihre Benutzer mehr oder weniger intuitiv sein. Besonders in Situationen mit vielen Projekten und entsprechend vielen Benutzern, kann es vorkommen, dass die Benutzer gewöhnlich nur mit einem oder zwei dieser Projekte vertraut sind. Allerdings schwächt dieser Projekt-als-Geschwister-Zweig-Ansatz die Betonung auf Projekt-Individualität und betrachtet die Gesamtmenge der Projekte als Ganzes. Das ist jedoch ein sozialer Aspekt. Wir mögen unseren ursprünglich geäußerten Vorschlag aus rein praktischen Erwägungen – es ist einfacher, in der kompletten Historie eines einzelnen Projektes zu forschen (oder sie zu verändern oder woanders hin zu migrieren), wenn es einen einzelnen Pfad im Projektarchiv gibt, der die gesamte Historie für dieses eine Projekt, und nur dieses, beinhaltet – die Vergangenheit, Tags und Zweige.

Entscheiden Sie, wo und wie Ihr Projektarchiv untergebracht werden soll

Bevor Sie Ihr Subversion-Projektarchiv anlegen, bleibt die offensichtliche Frage zu beantworten, wo das Ding hin soll. Das hängt eng mit etlichen weiteren Fragen zusammen, etwa wie auf das Projektarchiv zugegriffen werden soll (über einen

Subversion-Server oder direkt), wer darauf zugreifen soll (Benutzer hinter Ihrer Firmen-Firewall oder die weite Welt im offenen Netz), welche zusätzlichen Dienste Sie im Zusammenhang mit Subversion anbieten wollen (Schnittstellen zum Stöbern im Projektarchiv, Übergabebenachrichtigungen per E-Mail usw.), Ihre Sicherungsstrategie und vieles mehr.

Die Auswahl und Konfigurierung des Servers werden wir in Kapitel 6, *Konfiguration des Servers* behandeln; jedoch möchten wir an dieser Stelle kurz darauf hinweisen, dass die Antworten auf einige der anderen Fragen zur Folge haben, dass Sie bei der Entscheidung über den Speicherorte für das Projektarchiv keine freie Wahl mehr haben. Beispielsweise könnten bestimmte Einsatzumgebungen erfordern, dass von mehreren Rechnern über ein freigegebenes Dateisystem auf das Projektarchiv zugegriffen werden muss, so dass (wie Sie im nächsten Abschnitt lesen werden) die Wahl der dem Projektarchiv zugrunde liegenden Datenspeicherung gar keine Wahl mehr ist, da nur eine der verfügbaren Datenspeicherungsverfahren in dieser Umgebung funktionieren wird.

Es ist unmöglich, und würde auch den Rahmen dieses Buches sprengen, wenn jede erdenkliche Einsatzart von Subversion angesprochen würde. Wir ermutigen Sie einfach, Ihre Optionen zu prüfen, indem Sie diese Seiten und weitere Quellen als Referenz verwenden und weitsichtig planen.

Auswahl der Datenspeicherung

Subversion unterstützt zwei Optionen für das zugrunde liegende Datenspeicherungsverfahren – oft bezeichnet als „das Backend“ oder, etwas verwirrend, „das (versionierte) Dateisystem“ — welches jedes Projektarchiv verwendet. Das eine Verfahren speichert alles in einer Berkeley-DB- (oder BDB-) Datenbankumgebung; Projektarchive, die dieses Verfahren verwenden, werden oft „BDB-basiert“ genannt. Das andere Verfahren speichert die Daten in gewöhnlichen, flachen Dateien, die ein spezielles Format verwenden. Unter Subversion-Entwicklern hat sich dafür die Bezeichnung *FSFS*[4] eingebürgert – eine Implementierung eines versionierten Dateisystems, dass direkt das Dateisystem des Betriebssystems verwendet – statt einer Datenbankbibliothek oder einer anderen Abstraktionsebene – um Daten zu speichern

Tabelle 5.1, „Vergleich der Projektarchiv-Datenspeicherung“ liefert einen Vergleich zwischen Berkeley-DB- und FSFS-Projektarchive.

Tabelle 5.1. Vergleich der Projektarchiv-Datenspeicherung

Kategorie	Funktion	Berkeley DB	FSFS
Zuverlässigkeit	Unversehrtheit der Daten	Höchst zuverlässig, wenn es richtig aufgesetzt wird; Berkeley DB 4.4 bringt automatische Wiederherstellung	Ältere Versionen hatten einige selten aufgetretene, jedoch datenzerstörende Fehler
	Empfindlichkeit gegenüber Unterbrechungen	Sehr; Abstürze und Berechtigungsprobleme können die Datenbank „verklemmt“ hinterlassen, was eine Wiederherstellung mithilfe des Journals erfordert	Ziemlich unempfindlich
Zugriffsmöglichkeiten	Benutzbar über eine Laufwerk mit Nur-Lese-Zugriff	Nein	Ja
	Plattformunabhängige Speicherung	Nein	Ja
	Benutzbar über Netz-Dateisysteme	Im Allgemeinen nicht	Ja
	Behandlung von Gruppenberechtigungen	Empfindlich für Probleme mit der Benutzer-umask; Zugriff am besten nur durch einen Benutzer	Umgeht umask-Probleme

[4] Wenn Jack Repenning gefragt wird, ist die Aussprache oft „fass-fass“. (Jedoch geht dieses Buch davon aus, dass der Leser „eff-ess-eff-ess“ denkt)

Kategorie	Funktion	Berkeley DB	FSFS
Skalierbarkeit	Plattenplatzbedarf des Projektarchivs	Größer (besonders, wenn Protokolldateien nicht gekürzt werden)	Kleiner
	Anzahl an Revisionsbäumen	Datenbank; keine Probleme	Einige ältere Dateisysteme lassen sich mit tausenden Einträgen in einem Verzeichnis nicht gut skalieren
	Verzeichnisse mit vielen Dateien	Langsamer	Schneller
Arbeitsleistung	Auschecken der letzten Revision	Kein spürbarer Unterschied	Kein spürbarer Unterschied
	Große Übergaben	Insgesamt langsamer, aber die Kosten amortisieren sich über die Dauer der Übergabe	Insgesamt schneller, jedoch können Abschlussarbeiten zu Zeitüberschreitungen beim Client führen.

Beide dieser Verfahren haben Vor- und Nachteile. Keins davon ist „offizieller“ als das andere, obwohl das neuere FSFS seit Subversion 1.2 das Standardverfahren ist. Beide verfügen über die ausreichende Zuverlässigkeit, um ihnen Ihre versionierten Daten anzuvertrauen. Doch wie sie in Tabelle 5.1, „Vergleich der Projektarchiv-Datenspeicherung“ sehen können, bietet das FSFS-Verfahren bezüglich seiner unterstützten Einsatzumgebungen wesentlich mehr Flexibilität. Mehr Flexibilität bedeutet, dass Sie sich mehr anstrengen müssen, um es falsch einzusetzen. Dies sind die Gründe – hinzu kommt, dass beim Verzicht auf Berkeley DB sich eine Komponente weniger im System befindet – warum heutzutage beinahe jeder das FSFS-Verfahren verwendet, wenn neue Projektarchive angelegt werden.

Glücklicherweise interessiert es die meisten Programme die auf Subversion-Projektarchive zugreifen nicht, welches Speicherverfahren verwendet wird. Außerdem sind Sie mit Ihrer ersten Entscheidung für das Speicherverfahren nicht notwendigerweise festgelegt – falls Sie es sich später anders überlegen sollten, bietet Subversion Methoden zur Migration der Daten im Projektarchiv in ein anderes Projektarchiv, das ein unterschiedliches Speicherverfahren verwendet. Wir werden das später in diesem Kapitel erörtern.

Die folgenden Unterabschnitte bieten einen detaillierten Blick auf die verfügbaren Speicherverfahren.

Berkeley DB

In der anfänglichen Entwurfsphase von Subversion entschieden sich die Entwickler aus einer Reihe von Gründen, Berkeley DB zu verwenden. Hierzu zählen die quelloffene Lizenz, Transaktionsunterstützung, Zuverlässigkeit, Arbeitsleistung, Einfachheit der Programmierschnittstelle, Thread-Sicherheit, Cursor-Unterstützung usw.

Berkeley DB bietet echte Transaktionsunterstützung – vielleicht seine stärkste Funktion. Mehrere Prozesse, die auf Ihre Subversion-Projektarchive zugreifen, brauchen sich keine Sorgen machen, dass sie sich versehentlich gegenseitig die Daten zerschießen. Die durch das Transaktiossystem gebotene Isolation bedeutet, dass der Subversion-Projektarchiv-Programmcode für jede gegebene Operation eine statische Sicht auf die Datenbank hat – keine sich durch die Einflüsse anderer Prozesse ständig ändernde Datenbank – und basierend auf dieser Sicht Entscheidungen treffen kann. Falls die getroffene Entscheidung zu einem Konflikt damit führt, was eine anderer Prozess macht, wird die gesamte Transaktion zurückgerollt, als wäre sie nie passiert, und Subversion versucht höflich, die Operation mithilfe einer aktualisierten (aber immer noch statischen) Sicht der Datenbank erneut durchzuführen.

Eine weitere großartige Funktion von Berkeley DB sind *Hot Backups* – die Fähigkeit, die Datenbankumgebung zu sichern, ohne sie „vom Netz“ zu nehmen. Wir besprechen später in diesem Kapitel, wie Sie Ihr Projektarchiv sichern (in „Sicherung des Projektarchivs“), doch sollten die Vorteile offensichtlich sein, die dadurch entstehen, dass Sie vollständige Sicherheitskopien Ihrer Projektarchive ohne Wartungszeiträume machen können.

Bei Berkeley DB handelt es sich auch um ein sehr zuverlässiges Datenbanksystem, wenn es richtig verwendet wird. Subversion benutzt das Protokollierungssystem von Berkeley DB, was bedeutet, dass die Datenbank zunächst eine Beschreibung der Veränderungen in Protokolldateien auf der Platte schreibt, bevor die Veränderungen selbst durchgeführt werden. Dies stellt sicher, dass, falls etwas schief geht, die Datenbank zu einem früheren *Sicherungspunkt* zurückgehen kann – eine Stelle in den Protokolldateien, von der bekannt ist, dass sie eine nicht beschädigte Datenbank bezeichnet – und Transaktionen solange wiederholen kann, bis die Daten sich wieder in einem brauchbaren zustand befinden. Siehe

„Plattenplatzverwaltung“ weiter unten in diesem Kapitel für weitere Informationen zu Berkeley-DB-Protokolldateien.

Doch keine Rose ohne Dornen, uns so müssen wir auch einige bekannte Einschränkungen von Berkeley DB ansprechen. Erstens sind Berkeley-DB -Umgebungen nicht portierbar. Sie können nicht einfach ein unter Unix erzeugtes Subversion-Projektarchiv auf ein Windows-System kopieren und erwarten, dass es funktioniert. Obwohl ein Großteil des Berkeley-DB-Datenbankformats architekturunabhängig ist, sind es andere Teile der Umgebung nicht. Zweitens benutzt Subversion Berkeley DB auf eine Weise, die nicht auf Windows 95/98 funktioniert – falls Sie ein BDB-basiertes Projektarchiv auf einer Windows-Maschine unterbringen müssen, bleiben Sie bei Windows 2000 oder einem seiner Nachfolger.

Obwohl Berkeley DB verspricht, sich korrekt auf freigegebenen Netzlaufwerken zu verhalten, die bestimmte Anforderungen erfüllen,[5] bieten die meisten Netz-Dateisysteme *keine* derartige Unterstützung. Und keinesfalls dürfen Sie zulassen, dass gleichzeitig von mehreren Clients auf ein BDB-basiertes Projektarchiv auf einer Netzfreigabe zugegriffen wird (was eigentlich der Hauptgrund dafür ist, das Projektarchiv auf einer Netzfreigabe unterzubringen).

Falls Sie versuchen sollten, Berkeley DB auf einem Netz-Dateisystem unterzubringen, das die Anforderungen nicht erfüllt, ist das Ergebnis unvorhersehbar – es kann sein, dass Sie sofort mysteriöse Fehler wahrnehmen oder es kann Monate dauern, bis Sie bemerken, dass Ihre Projektarchiv-Datenbank fast unmerklich beschädigt ist. Sie sollten ernsthaft erwägen, das FSFS-Speicherverfahren für Projektarchive zu verwenden, die auf einer Netzfreigabe untergebracht werden sollen.

Schließlich ist Berkeley DB empfindlicher gegenüber Unterbrechungen als ein typisches relationales Datenbanksystem, da es sich um eine Bibliothek handelt, die direkt in Subversion eingebunden ist. Beispielsweise haben die meisten SQL-Systeme einen dedizierten Server-Prozess, der alle Tabellenzugriffe vermittelt. Falls ein auf die Datenbank zugreifendes Programm aus irgendeinem Grund abstürzt, bemerkt der Datenbank-Dämon die verlorene Verbindung und räumt anschließend auf. Und da der Datenbank-Dämon der einzige Prozess ist, der auf die Tabellen zugreift, brauchen sich Anwendungsprogramme nicht um Berechtigungskonflikte zu kümmern. Das trifft allerdings nicht auf Berkeley DB zu. Subversion (und jedes Programm, das die Subversion-Bibliotheken verwendet) greift direkt auf die Datenbanktabellen zu, was bedeutet, dass ein Programmabsturz die Datenbank vorübergehend in einem inkonsistenten, nicht zugreifbaren Zustand hinterlassen kann, so dass ein Administrator Berkeley DB dazu auffordern muss, zu einem Sicherungspunkt zurückzugehen, was etwas ärgerlich ist. Neben abgestürzten Prozessen können andere Dinge das Projektarchiv „verklemmen“, wie etwa Programme, die sich wegen Eigentums- und Zugriffsrechten auf Datenbankdateien ins Gehege kommen.

Berkeley DB 4.4 bietet (für Subversion 1.4 und spätere Versionen) die Fähigkeit, dass Subversion falls erforderlich automatisch und transparent Berkeley-DB-Umgebungen wiederherstellt. Wenn sich ein Subversion-Prozess an die Berkeley-DB-Umgebung hängt, verwendet er eine Art Prozess-Buchhaltung, um unsaubere Verbindungsabbrüche früherer Prozesse zu entdecken, führt die notwendige Wiederherstellung durch und fährt fort, als wäre nichts passiert. Dies verhindert das Vorkommen von Verklemmungen zwar nicht vollständig, verringert allerdings erheblich den Aufwand an menschlichen Eingriffen, um sich hiervon zu erholen.

Während ein Berkeley-DB-Projektarchiv ziemlich schnell und skalierbar ist, wird es am besten von einem einzelnen Server-Prozess unter einer Benutzerkennung verwendet – so wie Apaches **httpd** oder **svnserve** (siehe Kapitel 6, *Konfiguration des Servers*) – statt darauf mit mehreren Benutzern über `file://` oder `svn+ssh://` URLs zuzugreifen. Falls Sie mit mehreren Benutzern direkt auf ein Berkeley-DB-Projektarchiv zugreifen wollen, sollten Sie unbedingt „Unterstützung mehrerer Zugriffsmethoden auf das Projektarchiv“ weiter unten in diesem Kapitel lesen.

FSFS

Mitte 2004 entstand ein zweiter Typ eines Projektarchiv-Speichersystems – eins, das überhaupt keine Datenbank verwendet. Ein FSFS-Projektarchiv speichert die zu einer Revision gehörenden Änderungen in einer einzelnen Datei, so dass sich alle Revisionen eines Projektarchivs in einem Verzeichnis voller nummerierter Dateien befinden. Transaktionen werden als individuelle Dateien in getrennten Verzeichnissen erzeugt. Sobald sie vollständig ist, wird die Transaktionsdatei umbenannt und in das Revisionsverzeichnis verschoben, so dass die Atomizität von Übergaben gewährleistet ist. Und da eine Revisionsdatei dauerhaft und unveränderlich ist, kann das Projektarchiv auch im laufenden Betrieb gesichert werden, genauso wie ein BDB-basiertes Projektarchiv.

Revisionsdateien und Scherben

[5] Berkeley DB setzt voraus, dass das zugrunde liegende Dateisystem strenge POSIX-Sperrmechanismen implementiert und, noch wichtiger, die Fähigkeit mitbringt, Dateien direkt in den Prozessspeicher abzubilden.

FSFS Projektarchive beinhalten sowohl Dateien, die die in einer einzelnen Revision gemachten Änderungen beschreiben als auch Dateien, die die Revisionseigenschaften beinhalten, die zu einer einzelnen Revision gehören. Projektarchive, die mit Subversion-Versionen vor 1.5 erzeugt worden sind, speichern diese Dateien in zwei Verzeichnissen, eins für jede Art von Datei. Wenn dem Projektarchiv neue Revisionen hinzugefügt werden, hinterlegt Subversion immer mehr Dateien in diese beiden Verzeichnisse; im Lauf der Zeit kann die Anzahl dieser Dateien in jedem dieser Verzeichnisse recht groß werden. Dabei wurde beobachtet, dass es in bestimmten netzbasierten Dateisystemen zu Leistungsproblemen kommen kann.

Subversion 1.5 erstellt Projektarchive mit FSFS mit einem etwas veränderten Layout, bei dem der Inhalt dieser beiden Verzeichnisse in *Scherben* zerlegt wird, oder über mehrere Unterverzeichnisse verteilt wird. Das kann erheblich zur Verkürzung der Zeit beitragen, die das System benötigt, um irgendeine dieser Dateien zu finden, und somit die Gesamtleistung von Subversion beim Lesen des Projektarchivs erhöhen.

Subversion 1.6 geht bei diesem Scherben-Design noch einen Schritt weiter und erlaubt Administratoren, optional jede dieser Projektarchiv-Scherben in eine einzelne Multi-Revisions-Datei zu *packen*. Das kann sowohl Leistungsvorteile bringen als auch die Plattennutzung verbessern. Siehe „FSFS Filtersystem packen“ für weitere Informationen.

Die FSFS-Revisionsdateien beschreiben die Verzeichnisstruktur einer Revision, Dateiinhalte und Deltas zu Dateien in anderen Revisionsbäumen. Anders als eine Berkeley-DB-Datenbank, ist dieses Speicherformat auf verschiedene Betriebssysteme übertragbar und nicht von einer CPU-Architektur abhängig. Da weder Journaldateien noch Dateien für gemeinsam benutzten Speicher verwendet werden, kann auf das Projektarchiv sicher über ein Netzdateisystem zugegriffen und es in einer Nur-Lese-Umgebung untersucht werden. Das Fehlen der Datenbankverwaltung bedeutet ebenfalls eine etwas geringere Größe des Projektarchivs.

FSFS hat auch eine unterschiedliche Charakteristik, was den Durchsatz anbelangt. Wenn eine große Zahl an Dateien übergeben wird, kann FSFS schneller Verzeichniseinträge hinzufügen. Andererseits hat FSFS beim Abschluss einer Übergabe eine längere Verzögerung während es Aufgaben durchführt, die das BDB-Backend über die Laufzeit der Übergabe abwickelt, was in extremen Fällen dazu führen kann, dass bei Clients Zeitüberschreitungen beim Warten auf eine Antwort auftreten.

Der wichtigste Unterschied jedoch ist die Unempfänglichkeit von FSFS gegenüber Verklemmungen, wenn etwas schief geht. Falls ein Prozess, der eine Berkeley-DB-Datenbank benutzt, ein Berechtigungsproblem bekommt oder plötzlich abstürzt, kann das die Datenbank in einen unbrauchbaren Zustand bringen, bis ein Administrator sie wiederherstellt. Falls ein Prozess, der ein FSFS-Projektarchiv benutzt, in dieselbe Situation gerät, ist das Projektarchiv hiervon überhaupt nicht betroffen. Das Schlimmste, was passieren kann, ist, dass einige Transaktionsdaten nicht abgearbeitet werden können.

Anlegen und konfigurieren Ihres Projektarchivs

Weiter oben in diesem Kapitel (in „Strategien für die Verwendung eines Projektarchivs“), schauten wir auf einige der wichtigen Entscheidungen, die zu treffen sind, bevor Ihr Subversion-Projektarchiv angelegt und konfiguriert wird. Jetzt schaffen wir es endlich, uns die Hände schmutzig zu machen! In diesem Abschnitt werden wir sehen, wie ein Subversion-Projektarchiv überhaupt angelegt wird und wie es konfiguriert wird, damit es bei bestimmten Projektarchiv-Ereignissen spezielle Aktionen ausführt.

Anlegen des Projektarchivs

Das Anlegen eines Subversion-Projektarchivs ist eine unglaublich einfache Aufgabe. Das mit Subversion gelieferte Dienstprogramm **svnadmin** stellt ein Unterbefehl (**svnadmin create**) zur Verfügung, der genau das macht.

```
$ # Ein Projektarchiv anlegen
$ svnadmin create /var/svn/repos
$
```

Vorausgesetzt, dass das Verzeichnis `/var/svn/repos` vorhanden ist, Sie die zum Ändern erforderlichen Rechte besitzen, legt der obige Befehl ein neues Projektarchiv mit dem Standard-Dateisystem-Speicherverfahren (FSFS) an. Sie können den Dateisystemtypen ausdrücklich wählen, indem Sie das Argument `--fs-type` benutzen, das als Parameter entweder `fsfs` oder `bdb` zulässt.

```
$ # Ein FSFS-basiertes Projektarchiv anlegen
$ svnadmin create --fs-type fsfs /var/svn/repos
$
```

```
# Ein Berkeley-DB-basiertes Projektarchiv anlegen
$ svnadmin create --fs-type bdb /var/svn/repos
$
```

Nach dem Ausführen dieses einfachen Befehls haben Sie ein Subversion-Projektarchiv. Abhängig davon, wie Anwender künftig auf dieses neue Projektarchiv zugreifen sollen, müssten Sie gegebenenfalls an den Dateisystemberechtigungen feilen. Da allerdings die grundsätzliche Systemverwaltung nicht Gegenstand dieses Textes ist, betrachten wir die weitere Untersuchung dieses Themas als Übung für den Leser.

Das Pfad-Argument zu **svnadmin** ist bloß ein gewöhnlicher Pfad im Dateisystem und kein URL wie ihn das Client-Programm **svn** verwendet, um auf Projektarchive zu verweisen. Sowohl **svnadmin** als auch **svnlook** werden als serverseitige Dienstprogramme betrachtet – sie werden auf dem Rechner benutzt, auf dem das Projektarchiv untergebracht ist, um Aspekte des Projektarchivs zu untersuchen oder zu verändern; tatsächlich sind sie nicht in der Lage, Aufgaben über das Netz zu erledigen. Ein verbreiteter Fehler von Subversion-Neulingen ist der Versuch, URLs (sogar „lokale“ vom Typ `file://`) an diese zwei Programme zu übergeben.

Im Unterverzeichnis `db/` Ihres Projektarchivs befindet sich die Implementierung des versionierten Dateisystems. Das Leben des versionierten Dateisystems Ihres Projektarchivs beginnt mit Revision 0, die aus nichts anderem als dem Wurzelverzeichnis (`/`) besteht. Zu Beginn hat die Revision 0 eine einzige Revisions-Eigenschaft, `svn:date`, das als Wert die Angabe des Zeitpunktes besitzt, zu dem das Projektarchiv angelegt wurde.

Da Sie nun ein Projektarchiv haben, ist es an der Zeit, es anzupassen.

Während einige Teile des Projektarchivs – wie die Konfigurationsdateien und Hook-Skripte – für eine manuelle Untersuchung und Bearbeitung gedacht sind, sollten Sie nicht (und sie sollten es auch nicht nötig haben) an den anderen Teilen des Projektarchivs „händisch“ herumdoktern. Das Dienstprogramm **svnadmin** sollte für alle notwendigen Änderungen an Ihrem Projektarchiv ausreichen; sie können auch Dienstprogramme von Drittanbietern (wie das Werkzeugpaket von Berkeley DB) verwenden, um in entsprechenden Unterabschnitten des Projektarchivs Änderungen vorzunehmen. Versuchen Sie *nicht*, die Historie Ihrer Versionskontrolle manuell zu verändern, indem Sie in den Speicherdateien des Projektarchivs herumstochern!

Erstellen von Projektarchiv-Hooks

Ein *Hook* (Haken) ist ein Programm, das durch einige Projektarchiv-Ereignisse gestartet wird, wie etwa die Erzeugung einer neuen Revision oder die Veränderung einer unversionierten Eigenschaft. Einige Hooks (die sogenannten „Pre-Hooks“) starten vor einer Projektarchiv-Operation und bieten eine Möglichkeit sowohl zu berichten, was passieren wird, als auch zu verhindern, dass es überhaupt passiert. Andere Hooks (die „Post-Hooks“) starten nach Abschluss eines Projektarchiv-Ereignisses und sind nützlich für Aufgaben, die das Projektarchiv inspizieren – aber nicht verändern. Jedem Hook wird ausreichend Information übergeben, damit er feststellen kann, um welches Ereignis es sich handelt (oder handelte), welche genauen Änderungen am Projektarchiv beabsichtigt sind (oder durchgeführt wurden) und wie der Name des Benutzers lautet, der das Ereignis ausgelöst hat.

Das Unterverzeichnis `hooks` beinhaltet standardmäßig Vorlagen für verschiedene Projektarchiv-Hooks:

```
$ ls repos/hooks/
post-commit.tmpl          post-unlock.tmpl  pre-revprop-change.tmpl
```

```
post-lock.tmpl            pre-commit.tmpl   pre-unlock.tmpl
post-revprop-change.tmpl  pre-lock.tmpl     start-commit.tmpl
$
```

Es gibt eine Vorlage für jeden Hook, den Subversion unterstützt. Sie können sehen, wodurch jedes dieser Skripte gestartet wird und welche Daten übergeben werden, indem Sie den Inhalt der Skripte inspizieren. In vielen dieser Vorlagen befinden sich auch Beispiele dafür, wie dieses Skript zusammen mit anderen Programmen aus dem Lieferumfang von Subversion verwendet werden kann, um häufige, nützliche Aufgaben zu erledigen. Um einen funktionierenden Hook zu installieren, brauchen Sie nur ein ausführbares Programm oder Skripte im Verzeichnis `repos/hooks` abzulegen, das unter dem Namen des Hooks (etwa **start-commit** oder **post-commit**) gestartet werden kann.

Auf Unix Plattformen bedeutet das, ein Skript oder Programm bereitzustellen (welches ein Shell-Skript, ein Python-Programm, ein übersetztes C-Binärprogramm oder sonst etwas sein kann), das genauso heißt wie der Hook. Natürlich sind die Vorlagen nicht nur zur Information da – die einfachste Möglichkeit, unter Unix einen Hook zu installieren, ist es, einfach die passende Vorlagedatei in eine Datei zu kopieren, der die Dateiendung `.tmpl` fehlt, den Inhalt anzupassen und sicherzustellen, dass das Skript ausführbar ist. Unter Windows werden jedoch Dateiendungen verwendet, um festzustellen, ob ein Programm ausführbar ist, so dass Sie ein Programm zur Verfügung stellen müssen, dessen Basisname dem Hook entspricht und dessen Endung einer derjenigen entspricht, die Windows für ausführbare Programme hält, etwa `.exe` für Programme und `.bat` für Batch-Dateien.

Aus Sicherheitsgründen führt Subversion Hook-Programme in einer leeren Umgebung aus – d.h., es sind überhaupt keine Umgebungsvariablen gesetzt, nicht einmal `$PATH` (oder `%PATH%` unter Windows). Deshalb sind viele Administratoren verwirrt, wenn deren Hook-Programme normal starten, wenn sie manuell aufgerufen werden, aber nicht laufen, wenn sie Subversion startet. Stellen Sie sicher, dass Sie entweder alle notwendigen Umgebungsvariablen in Ihren Hook-Programmen ausdrücklich setzen und/oder absolute Pfade zu Programmen verwenden.

Subversion führt die Hooks unter der Benutzerkennung aus, die auch der Prozess besitzt, der auf das Projektarchiv zugreift. Meistens wird auf das Projektarchiv über einen Subversion-Server zugegriffen, so dass die Benutzerkennung der des Serverprozesses entspricht. Die Hooks müssen deshalb mit den entsprechenden Berechtigungen des Betriebssystems versehen werden, damit diese Benutzerkennung sie ausführen kann. Das bedeutet auch, dass der direkte oder indirekte Zugriff auf irgendwelche Programme oder Dateien (einschließlich des Subversion-Projektarchivs) durch den Hook auch unter derselben Kennung erfolgt. Mit anderen Worten: Achten Sie auf mögliche Probleme im Zusammenhang mit Zugriffsrechten, die den Hook daran hindern könnten, die Ihm zugeteilten Aufgaben wahrzunehmen.

Es gibt mehrere im Subversion-Projektarchiv implementierte Hooks; Details zu jedem können Sie in „Projektarchiv-Hooks“ nachlesen. Als Projektarchiv-Administrator müssen Sie entscheiden, welche Hooks sie einrichten wollen (indem Sie ein entsprechend benanntes und mit den nötigen Zugriffsrechten versehenes Hook-Programm bereitstellen) und wie Sie sie einsetzen wollen. Wenn Sie diese Entscheidung treffen, dann behalten Sie das Gesamtbild des Projektarchiv-Einsatzes im Auge. Wenn Sie beispielsweise die Konfiguration des Servers verwenden, um festzustellen, welche Benutzer Änderungen an Ihr Projektarchiv übergeben dürfen, benötigen Sie für diese Zugriffskontrolle nicht das Hook-System.

Es gibt keinen Mangel an Subversion-Hook-Programmen und Skripten, die frei verfügbar sind, entweder von der Subversion-Gemeinschaft oder von woanders her. Diese Skripte decken ein breites Spektrum ab – grundlegende Zugriffskontrolle, Kontrolle der Prozesstreue, Fehlersystemanbindung, E-Mail-basierte oder syndizierte Benachrichtigungen bei Übergaben und noch viel mehr. Oder, falls Sie Ihren eigenen schreiben wollen, siehe Kapitel 8, *Subversion integrieren*.

Obwohl Hook-Skripte fast alles machen können, gibt es eine Dimension, in der sich Hook-Skript-Autoren zurückhalten sollten: Ändern Sie *keine* Übergabe-Transaktion mithilfe von Hook-Skripten. Trotz der Verlockung, Hook-Skripte zur automatischen Korrektur von Fehlern, Unzulänglichkeiten oder Prozessverletzungen innerhalb der zu übergebenden Dateien einzusetzen, kann das zu Problemen führen. Subversion hält bestimmte Projektarchiv-Daten in client-seitigen Caches vor, und wenn Sie auf diese Art eine Übergabe-Transaktion verändern, werden die im Cache befindlichen Informationen ungültig, ohne dass jemand etwas merkt. Diese Inkonsistenz kann zu überraschendem und unerwartetem Verhalten führen. Statt die Transaktion zu verändern, sollten Sie sie einfach im `pre-commit`-Hook auf *Gültigkeit* prüfen und die Übergabe ablehnen, falls sie den Anforderungen nicht entspricht. Als Nebeneffekt werden Ihre Benutzer lernen, wie wertvoll eine sorgfältige, sich an den Vorgaben orientierende Arbeitsweise ist.

Konfiguration von Berkeley DB

Eine Berkeley-DB-Umgebung ist eine Kapselung einer oder mehrerer Datenbanken, Protokolldateien, Regionsdateien und Konfigurationsdateien. Die Berkeley-DB-Umgebung hat ihre eigene Menge an Konfigurationswerten für Dinge wie die Maximalzahl von Datenbanksperren zu einem gegebenen Zeitpunkt, die Obergrenze für die Größe der Protokolldateien usw. Darüber hinaus wählt Subversions Dateisystemlogik Standardwerte für einige der Berkeley-DB-Konfigurationsoptionen. Manchmal jedoch benötigt Ihr besonderes Projektarchiv, welches eine einzigartige Sammlung von Daten und Zugriffsmustern darstellt, eine unterschiedliche Menge von Konfigurationswerten.

Den Herstellern von Berkeley-DB ist bewusst, dass unterschiedliche Anwendungen und Datenbankumgebungen auch unterschiedliche Anforderungen haben, so dass sie einen Mechanismus zur Verfügung gestellt haben, der es ermöglicht, während der Laufzeit viele der Konfigurationseinstellungen für die Berkeley-DB-Umgebung zu überschreiben. BDB prüft, ob es eine Datei namens `DB_CONFIG` im Umgebungsverzeichnis (das Verzeichnis `db` des Projektarchivs) gibt und liest die in dieser Datei vorhandenen Optionen. Subversion erzeugt diese Datei selbst, wenn der Rest eines Projektarchivs erstellt wird. Anfangs beinhaltet diese Datei einige Standardoptionen sowie Verweise zur Berkeley-DB-Dokumentation im Netz, so dass Sie nachschlagen können, was diese Optionen bewirken. Selbstverständlich können Sie beliebige von Berkeley DB unterstützte Optionen der Datei `DB_CONFIG` hinzufügen. Beachten Sie jedoch, dass Sie es vermeiden sollten, obwohl Subversion niemals versucht, den Inhalt der Datei zu lesen oder zu interpretieren und auch sonst keinen direkten Gebrauch von den dortigen Optionseinstellungen macht, die Konfiguration so zu verändern, dass sich Berkeley DB anders verhält, als es Subversion erwartet. Die Änderungen an `DB_CONFIG` werden außerdem erst nach einer Wiederherstellung der Datenbankumgebung (mit **svnadmin recover**) gültig.

FSFS Konfiguration

Seit Subversion 1.6 besitzen FSFS Dateisysteme mehrere konfigurierbare Parameter, die ein Administrator zur Feinabstimmung der Leistungsfähigkeit oder der Plattennutzung seines Projektarchivs verwenden kann. Sie können diese Optionen und deren Dokumentation in der Datei `db/fsfs.conf` im Projektarchiv finden.

Projektarchiv-Wartung

Die Wartung eines Subversion-Projektarchivs kann abschreckend sein, was an der Komplexität liegt, die Systemen innewohnt, die auf Datenbanken aufbauen. Die Arbeit gut zu machen, bedeutet, die Werkzeuge zu kennen – was sie sind, wann sie zu verwenden sind und wie. Dieser Abschnitt stellt Ihnen die Projektarchiv-Verwaltungswerkzeuge vor, die Subversion mitbringt und erörtert, wie sie gehandhabt werden, um Aufgaben zu erledigen, wie etwa Projektarchiv-Datenmigration, Aktualisierungen, Sicherungen und Aufräumarbeiten.

Der Werkzeugkasten eines Administrators

Subversion stellt eine Handvoll Dienstprogramme zur Verfügung, die nützlich zum Erstellen, Untersuchen, Verändern und Reparieren Ihres Projektarchivs sind. Wir wollen uns diese Werkzeuge einmal genauer ansehen. Anschließend werden wir kurz einige der zum Berkeley-DB-Paket gehörenden Dienstprogramme untersuchen, die auf die Besonderheiten der von Subversion verwendeten Datenbank zugeschnittene Funktionen anbieten, die mit Subversions eigenen Werkzeugen nicht verfügbar sind.

svnadmin

Das Programm **svnadmin** ist der beste Freund des Projektarchiv-Administrators. Neben der Fähigkeit, Subversion-Projektarchive zu erzeugen, erlaubt Ihnen dieses Programm verschiedene Wartungsarbeiten auf diesen Projektarchive auszuführen. Die Syntax von **svnadmin** ist ähnlich wie bei anderen Kommandozeilenprogrammen von Subversion:

```
$ svnadmin help
Aufruf: svnadmin UNTERBEFEHL ARCHIV_PFAD [Optionen & Parameter ...]
Geben Sie »svnadmin help <Unterbefehl>« ein, um Hilfe zu einem Unterbefehl
          zu erhalten.
Geben Sie »svnadmin --version« ein, um die Programmversion und die Datei-
          systemmodule zu sehen.

Verfügbare Unterbefehle:
   crashtest
   create
```

```
   deltify
…
```

Früher in diesem Kapitel (in „Anlegen des Projektarchivs“), wurde uns der Unterbefehl **svnadmin create** vorgestellt. Die meisten anderen Unterbefehle von **svnadmin** werden wir später in diesem Kapitel behandeln. Und in „svnadmin – Subversion Projektarchiv-Verwaltung“ können Sie in einer vollständigen Aufstellung der Unterbefehle nachlesen, was jeder zu bieten hat.

svnlook

svnlook ist ein von Subversion mitgeliefertes Dienstprogramm zum Untersuchen der mannigfaltigen Revisionen und *Transaktionen* (bei denen es sich um Revisionen in Entstehung handelt) in einem Projektarchiv. Kein Teil dieses Programms versucht, das Projektarchiv zu verändern. **svnlook** wird üblicherweise von Projektarchiv-Hooks verwendet, um die abzuliefernden Änderungen zu melden (im Fall des **pre-commit**-Hooks) oder die gerade an das Projektarchiv übergeben wurden (im Fall des **post-commit**-hooks). Ein Projektarchiv-Administrator kann dieses Programm zur Diagnose benutzen.

svnlook besitzt eine überschaubare Syntax:

```
$ svnlook help
Aufruf: svnlook UNTERBEFEHL ARCHIV_PFAD [Optionen & Parameter ...]
Hinweis: Alle Unterbefehle, die die Parameter »--revision« und »--transaction«
         akzeptieren, werden ohne diese Parameter die neueste
         Revision des Projektarchivs verwenden.
Geben Sie »svnlook help <Unterbefehl>« ein, um Hilfe zu einem Unterbefehl
         zu erhalten.
Geben Sie »svnlook --version« ein, um die Programmversion und die Datei-
         systemmodule zu sehen.
…
```

Die meisten Unterbefehle von **svnlook** können entweder auf einem Revisions- oder auf einem Transaktionsbaum arbeiten, indem sie Informationen über den Baum an sich ausgeben oder darüber, inwiefern er sich von einer früheren Revision des Projektarchivs unterscheidet. Sie verwenden die Optionen `--revision` (`-r`) und `--transaction` (`-t`), um die zu untersuchende Revision bzw. Transaktion anzugeben. Ohne eine der Optionen `--revision` (`-r`) und `--transaction` (`-t`) untersucht Subversion die jüngste (oder `HEAD`) Revision des Projektarchivs. Das heißt, die beiden folgenden Befehle machen genau dasselbe, wenn 19 die jüngste Revision im Projektarchiv unter `/var/svn/repos` ist:

```
$ svnlook info /var/svn/repos
$ svnlook info /var/svn/repos -r 19
```

Eine Ausnahme von diesen Regeln zu Unterbefehlen ist der Unterbefehl **svnlook youngest**, der keine Optionen entgegennimmt und einfach die jüngste Revisionsnummer des Projektarchivs ausgibt:

```
$ svnlook youngest /var/svn/repos
19
$
```

Beachten Sie, dass Sie nur Transaktionen untersuchen können, die noch nicht übergeben sind. Die meisten Projektarchive haben keine derartigen Transaktionen, da Transaktionen entweder übergeben (in diesem Fall sollten Sie darauf mit der Option `--revision` (`-r`) zugreifen) oder abgebrochen und entfernt sind.

Die Ausgabe **svnlook** ist so gestaltet, dass sie sowohl für Menschen als auch für Maschinen lesbar ist. Nehmen wir zum Beispiel die Ausgabe des Unterbefehls **svnlook info**:

```
$ svnlook info /var/svn/repos
sally
```

```
2002-11-04 09:29:13 -0600 (Mon, 04 Nov 2002)
43
Den üblichen griechischen
Baum hinzugefügt.
$
```

Die Ausgabe von **svnlook info** besteht aus dem Folgenden in entsprechender Reihenfolge:

1. Der Autor gefolgt von einem Zeilenvorschub
2. Das Datum gefolgt von einem Zeilenvorschub
3. Die Anzahl der Zeichen der Protokollnachricht gefolgt von einem Zeilenvorschub.
4. Die eigentliche Protokollnachricht gefolgt von einem Zeilenvorschub

Diese Ausgabe ist für Menschen lesbar, d.h., Dinge wie der Zeitstempel werden als Text dargestellt statt als irgendetwas Obskures (wie die Anzahl der Nanosekunden seit der Mann von Bofrost das letzte Mal da war). Jedoch ist die Ausgabe auch maschinenlesbar – weil die Protokollnachricht mehrere Zeilen umfassen und von der Länge her unbegrenzt sein kann, liefert **svnlook** die Länge der Nachricht vor der eigentlichen Nachricht. Das erlaubt Skripten und anderen Programmen, die um diesen Befehl herum geschrieben wurden, intelligente Entscheidungen in Bezug auf die Protokollnachricht zu treffen, etwa wie viel Speicher für die Nachricht anzufordern ist oder zumindest wie viele Bytes zu überspringen sind, falls diese Ausgabe nicht das letzte Stück im Datenstrom sein sollte.

svnlook kann eine Auswahl anderer Abfragen ausführen: Teilmengen der bereits erwähnten Informationen ausgeben, versionierte Verzeichnisbäume rekursiv auflisten, berichten, welche Pfade in einer bestimmten Revision oder Transaktion verändert wurden, textuelle und property-basierte Unterschiede an Dateien und Verzeichnissen aufzeigen, usw. Siehe „svnlook – Subversion Projektarchiv-Untersuchung“ für eine vollständige Referenz der Funktionen von **svnlook**.

svndumpfilter

Obwohl es nicht das am meisten verwendete Werkzeug im Sortiment des Administrators sein wird, bietet **svndumpfilter** eine ganz besondere Art von nützlichen Funktionen – die Fähigkeit, schnell und einfach Datenströme aus der Projektarchiv-Historie zu verändern, indem es als ein pfadbasierter Filter arbeitet.

Die Syntax von **svndumpfilter** lautet wie folgt:

```
$ svndumpfilter help
Aufruf: svndumpfilter UNTERBEFEHL [Optionen & Parameter ...]
Geben Sie »svndumpfilter help <Unterbefehl>« ein, um Hilfe zu einem
         Unterbefehl zu erhalten.
Geben Sie »svndumpfilter --version« ein, um die Programmversion zu sehen.

Verfügbare Unterbefehle:
   exclude
   include
   help (?, h)
```

Es gibt nur zwei interessante Unterbefehle: **svndumpfilter exclude** und **svndumpfilter include**. Sie erlauben Ihnen, zwischen einer impliziten oder expliziten Einbeziehung von Pfaden im Datenstrom zu wählen. Sie können mehr über diese Unterbefehle und den einzigartigen Zweck von **svndumpfilter** später in diesem Kapitel unter „Filtern der Projektarchiv-Historie“ erfahren.

svnsync

Der Befehl **svnsync**, der in Subversion 1.4 neu hinzugekommen ist, bietet Funktionen zum Verwalten eines Nur-Lese-Spiegels des Subversion-Projektarchivs. Das Programm hat eine Aufgabe – die versionierte Historie eines Projektarchivs in ein anderes zu übertragen. Und während es nicht viele Möglichkeiten gibt, dies zu tun, liegt seine hauptsächliche Stärke darin, das es aus der Ferne eingesetzt werden kann – das „Quell“- und „Ziel“-Projektarchiv können auf verschiedenen Rechnern liegen und auf einem anderen Rechner als **svnsync** selbst.

Wie Sie vielleicht erwarten, hat **svnsync** eine Syntax, die allen anderen Programmen aus diesem Kapitel gleicht:

```
$ svnsync help
Aufruf: svnsync UNTERBEFEHL ZIEL_URL [Optionen & Parameter ...]
Geben Sie »svnsync help <Unterbefehl>« ein, um Hilfe zu einem
         Unterbefehl zu erhalten.
Geben Sie »svnsync --version« ein, um die Programmversion und die Zugriffs-
         module zu sehen.

Verfügbare Unterbefehle:
   initialize (init)
   synchronize (sync)
   copy-revprops
   help (?, h)

$
```

Später in diesem Kapitel werden wir mehr über das Replizieren von Projektarchiven mit **svnsync** reden (siehe „Projektarchiv Replikation“).

fsfs-reshard.py

Obwohl es kein offizielles Glied in der Werkzeugkette von Subversion ist, handelt es sich bei dem Skript **fsfs-reshard.py** (zu finden im Verzeichnis `tools/server-side` des Subversion-Quelltext-Paketes) um ein nützliches Werkzeug zur Leistungssteigerung für Administratoren von FSFS-basierten Subversion-Projektarchiven. FSFS-Projektarchive enthalten Dateien, die die Änderungen in einer einzelnen Revision beschreiben sowie Dateien, die die zu einer Revision gehörenden Eigenschaften beinhalten. Projektarchive, die in einer früheren Version als Subversion 1.5 erzeugt wurden, legen diese Dateien in zwei Verzeichnissen ab – eins pro Dateityp. Während neue Revisionen an das Projektarchiv übergeben werden, legt Subversion dort immer mehr Dateien ab – im Lauf der Zeit kann die Anzahl der Dateien recht groß werden. Es wurde festgestellt, dass dies bei bestimmten netzbasierten Dateisystemen zu Leistungseinbußen kommen kann.

Subversion 1.5 legt FSFS-basierte Projektarchive mit einer geringfügig veränderten Struktur an, in der der Inhalt dieser beiden Verzeichnisse *aufgebrochen* ist, d.h. über mehrere Unterverzeichnisse aufgeteilt ist. Das kann die Zeit erheblich beschleunigen, die benötigt wird, um eine dieser Dateien zu finden und führt somit zu einer allgemeinen Leistungssteigerung beim Lesen aus dem Projektarchiv. Die Anzahl der Unterverzeichnisse für diese Dateien ist jedoch konfigurierbar, und hier setzt **fsfs-reshard.py** an. Dieses Skript mischt die Dateistruktur des Projektarchivs und ordnet sie gemäß der Anzahl der gewünschten Unterverzeichnisse neu an. Das ist insbesondere dann nützlich, wenn ein älteres Projektarchiv in die neue Struktur von Subversion 1.5 überführt werden soll (was Subversion nicht automatisch für Sie macht) oder falls ein bereits aufgeteiltes Projektarchiv noch feiner eingestellt werden soll.

Dienstprogramme von Berkeley DB

Falls Sie ein Projektarchiv verwenden, das auf Berkeley DB basiert, befindet sich die gesamte Struktur und die Daten Ihres versionierten Dateisystems in einer Menge von Datenbanktabellen innerhalb des Unterverzeichnisses `db/` Ihres Projektarchivs. Dieses Unterverzeichnis ist ein gewöhnliches Verzeichnis einer Berkeley-DB-Umgebung und kann deshalb mit irgendeinem der Berkeley Datenbankwerkzeuge verwendet werden, die normalerweise mit Berkeley DB ausgeliefert werden.

Für die tägliche Arbeit mit Subversion werden diese Werkzeuge nicht benötigt. Die meisten Funktionen, die üblicherweise für Subversion-Projektarchive gebraucht werden, sind in **svnadmin** integriert worden. Beispielsweise liefern **svnadmin list-unused-dblogs** und **svnadmin list-dblogs** eine Teilmenge dessen, was vom Berkeley-Dienstprogramm **db_archive** angeboten wird, und **svnadmin recover** spiegelt die verbreiteten Anwendungsfälle von **db_recover** wieder.

Trotzdem gibt es noch ein paar Berkeley-DB-Werkzeuge, die Ihnen nützlich sein könnten. Die Programme **db_dump** und **db_load** schreiben bzw. lesen ein spezielles Dateiformat, das die Schlüssel und Werte in einer Berkeley-DB-Datenbank beschreibt. Da Berkeley-Datenbanken nicht zwischen Rechnerarchitekturen portierbar sind, stellt dieses Format ein nützliches Verfahren zur Übertragung der Datenbanken zwischen Maschinen zur Verfügung, wobei die Architektur oder das Betriebssystem keine Rolle spielen. Später in diesem Kapitel werden wir noch beschreiben, wie Sie auch **svnadmin dump** und **svnadmin load** für ähnliche Zwecke verwenden können, doch **db_dump** und **db_load** können bestimmte Aufgaben genauso gut und viel schneller erledigen. Sie können auch dabei dienlich sein, wenn der erfahrene Berkeley-DB-Hacker aus irgendwelchen Gründen die Daten in einem BDB-basierten Projektarchiv direkt vor Ort anpassen muss, was die Dienstprogramme von Subversion nicht erlauben. Ferner liefert das Dienstprogramm **db_stat** nützliche Informationen über

den Zustand Ihrer Berkeley-DB-Umgebung, wozu ausführliche Statistiken über das Sperr- und Speicher-Teilsystem gehören.

Besuchen Sie für weitergehende Informationen zur Berkeley-Werkzeugsammlung den Dokumentationsabschnitt der Berkeley-DB-Abteilung auf der Seite von Oracle bei http://www.oracle.com/technology/documentation/berkeley-db/db/.

Berichtigung des Protokolleintrags

Manchmal kommt es vor, dass ein Benutzer einen Fehler im Protokolleintrag gemacht hat (einen Tippfehler oder vielleicht eine Fehlinformation). Falls das Projektarchiv entsprechend eingestellt ist (indem der Hook `pre-revprop-change` verwendet wird; siehe „Erstellen von Projektarchiv-Hooks“), um Änderungen am Protokolleintrag vorzunehmen nachdem die Übergabe abgeschlossen ist, kann der Benutzer den Protokolleintrag aus der Ferne mit dem Befehl **svn propset** (siehe svn propset (pset, ps)) „berichtigen“. Wegen der Möglichkeit, dadurch für immer Informationen zu verlieren, sind Subversion-Projektarchive allerdings standardmäßig nicht so eingestellt, dass Änderungen an unversionierten Eigenschaften erlaubt sind – außer für einen Administrator.

Falls ein Protokolleintrag durch einen Administrator geändert werden muss, kann das mit **svnadmin setlog** geschehen. Dieser Befehl ändert den Protokolleintrag (die Eigenschaft `svn:log`) einer gegebenen Revision eines Projektarchivs, indem der neue Inhalt aus einer angegebenen Datei gelesen wird.

```
$ echo "Hier ist der neue, korrekte Protokolleintrag" > newlog.txt
$ svnadmin setlog myrepos newlog.txt -r 388
```

Auch der Befehl **svnadmin setlog** ist standardmäßig durch dieselben Schutzmechanismen gegen die Veränderung unversionierter Eigenschaften eingeschränkt wie ein Client aus der Ferne – die Hooks `pre-` und `post-revprop-change` werden immer noch ausgelöst und müssen entsprechend eingestellt werden, um solche Änderungen zuzulassen. Allerdings kann ein Administrator diese Schutzmechanismen umgehen, indem er die Option `--bypass-hooks` an den Befehl **svnadmin setlog** übergibt.

Denken Sie trotzdem daran, dass beim Umgehen der Hooks auch Dinge umgangen werden wie E-Mail-Benachrichtigungen bei Eigenschafts-Änderungen, Sicherungssysteme, die Änderungen an unversionierten Eigenschaften verfolgen, usw. Mit anderen Worten: Seien Sie sehr vorsichtig bei der Auswahl dessen, was Sie ändern und wie Sie es ändern.

Plattenplatzverwaltung

Obwohl die Kosten für Speicherplatz in den letzten Jahren unglaublich gefallen sind, ist Plattenplatz immer noch ein berechtigtes Anliegen für Administratoren, die große Mengen von Daten zu versionieren haben. Jedes im aktiven Projektarchiv gespeicherte Bisschen Information über die Versionshistorie muss zu einem anderen Ort gesichert werden; vielleicht sogar öfter, falls eine zyklische Sicherungsstrategie angewendet wird. Es ist zweckdienlich zu wissen, welche Teile von Subversions Projektarchiv am Ort verbleiben müssen, welche gesichert werden müssen und welche ruhig entfernt werden können.

Wie Subversion Plattenplatz spart

Um das Projektarchiv klein zu halten, verwendet Subversion innerhalb des Projektarchivs *Delta-Kodierung* (oder Deltaspeicherung). Unter Delta-Kodierung wird die Kodierung eines Datensatzes als eine Sammlung von Unterschieden gegenüber einem anderen Datensatz verstanden. Falls die beiden Datensätze sehr ähnlich sind, bewirkt diese Delta-Kodierung eine Einsparung an Speicherplatz für den als Delta gespeicherten Datensatz – anstatt den Platz der Originaldaten zu belegen, wird hierbei nur soviel Platz benötigt, um zu sagen: „Schau mal, ich sehe genau so aus, wie der andere Datensatz da drüben, bis auf die folgenden paar Änderungen.“ Das Ergebnis ist, dass die meisten der Daten im Projektarchiv, die normalerweise recht voluminös sind – nämlich der Inhalt versionierter Dateien – in einer viel geringeren Größe gespeichert werden als der ursprüngliche Volltext dieser Daten. Und für Projektarchive, die mit Subversion 1.4 oder später angelegt wurden, ist die Platzersparnis sogar noch besser – jetzt sind die Volltexte der Dateiinhalte selbst komprimiert.

Da alle delta-kodierten Daten in einem BDB-basierten Projektarchiv in einer einzigen Berkeley-DB-Datenbankdatei gespeichert werden, wird die verringerte Größe der gespeicherten Werte nicht unmittelbar die Größe der Datenbankdatei verringern. Berkeley DB führt jedoch intern Buch über unbenutzte Bereiche der Datenbankdatei und wird zunächst jene aufbrauchen, bevor die Datenbankdatei selbst vergrößert wird. Während Delta-Kodierung also nicht unmittelbare Platzersparnis bringt, kann sie jedoch das künftige Wachstum der

Datenbank drastisch verlangsamen.

Entfernen unvollendeter Transaktionen

Obwohl es selten vorkommt, gibt es Umstände, unter denen der Übergabeprozess mit einem Fehler abbricht und die Reste einer Revision in Spe hinterlässt – eine unvollendete Transaktion samt aller Datei- und Verzeichnisänderungen, die dazugehören. Dies kann aus verschiedenen Gründen passieren: Vielleicht wurde die Operation des Clients vom Benutzer unsauber beendet oder es trat mittendrin ein Netzfehler auf. Aus welchem Grund auch immer, es können unvollendete Transaktionen auftreten. Sie verursachen keine tatsächlichen Schäden, außer Plattenplatz zu verschwenden. Ein penibler Administrator möchte sie vielleicht dennoch entfernen.

Sie können den Befehl **svnadmin lstxns** verwenden, um die Namen der aktuell ausstehenden Transaktionen anzuzeigen:

```
$ svnadmin lstxns myrepos
19
3a1
a45
$
```

Jeder Eintrag der Ausgabe kann dann mit dem Befehl **svnlook** (und seiner Option `--transaction` (`-t`)) aufgerufen werden, um festzustellen, wer die Transaktion erzeugt hat, wann sie erzeugt wurde und welche Änderungen sie beinhaltet – Informationen, die bei der Entscheidung helfen können, ob eine Transaktion ein sicherer Kandidat zum Löschen ist! Wenn Sie tatsächlich eine Transaktion löschen wollen, kann deren Name an den Befehl **svnadmin rmtxns** übergeben werden, der dann die Transaktion aufräumt. **svnadmin rmtxns** kann seine Eingabe auch direkt aus der Ausgabe von **svnadmin lstxns** beziehen!

```
$ svnadmin rmtxns myrepos `svnadmin lstxns myrepos`
$
```

Falls Sie auf diese Weise diese beiden Unterbefehle verwenden, sollten Sie vorübergehend das Projektarchiv für Clients unzugänglich machen. So kann niemand eine berechtigte Transaktion beginnen, bevor Sie aufgeräumt haben. Beispiel 5.1, „txn-info.sh (ausstehende Transaktionen anzeigen)“ enthält ein kleines Shell-Skript, das schnell eine Übersicht über jede ausstehende Transaktion in Ihrem Projektarchiv erzeugen kann.

Beispiel 5.1. txn-info.sh (ausstehende Transaktionen anzeigen)

```
#!/bin/sh

### Erzeuge Informationen über alle ausstehenden Transaktionen eines
### Subversion Projektarchivs.

REPOS="${1}"
if [ "x$REPOS" = x ] ; then
  echo "Aufruf: $0 REPOS_PATH"
  exit
fi

for TXN in `svnadmin lstxns ${REPOS}`; do
  echo "---[ Transaktion ${TXN} ]-------------------------------------------"
  svnlook info "${REPOS}" -t "${TXN}"
done
```

Die Ausgabe des Skriptes ist im Grunde genommen eine Aneinanderreihung mehrerer Teile von **svnlook info**-Ausgaben (siehe „svnlook“) und sieht etwa so aus:

```
$ txn-info.sh myrepos
---[ Transaktion 19 ]-------------------------------------------
sally
2001-09-04 11:57:19 -0500 (Tue, 04 Sep 2001)
0
---[ Transaktion 3a1 ]------------------------------------------
harry
2001-09-10 16:50:30 -0500 (Mon, 10 Sep 2001)
39
Versuch, über eine schlechte Netzverbindung abzuliefern.
---[ Transaktion a45 ]------------------------------------------
sally
2001-09-12 11:09:28 -0500 (Wed, 12 Sep 2001)
0
$
```

Eine vor langer Zeit aufgegebene Transaktion bedeutet normalerweise eine Art fehlgeschlagenen oder unterbrochenen Übergabeversuch. Der Zeitstempel einer Transaktion kann eine interessante Information sein – ist es beispielsweise wahrscheinlich, dass eine vor neun Monaten begonnene Operation immer noch aktiv ist?

Kurz gesagt, Entscheidungen zur Bereinigung von Transaktionen sollten klug getroffen werden. Verschiedene Informationsquellen – hierzu gehören die Fehler- und Zugriffsprotokolldateien von Apache, die operativen Protokolldateien von Subversion, die Revisions-Historie von Subversion usw. – können während des Entscheidungsprozesses hinzugezogen werden. Natürlich kann sich ein Administrator auch einfach mit dem Eigentümer einer anscheinend abgebrochenen Transaktion in Verbindung setzen (z.B. per E-Mail), um sicherzustellen, dass die Transaktion sich tatsächlich in einem Zombiezustand befindet.

Entfernen unbenutzter Protokolldateien von Berkeley DB

Bis vor kurzer Zeit waren die größten Plattenplatzfresser bei BDB-basierten Subversion-Projektarchive die Protokolldateien, in die Berkeley DB zunächst alle Schritte hineinschreibt, bevor es die eigentlichen Datenbankdateien verändert. Diese Dateien halten alle Aktionen der Datenbank auf dem Weg von einem Zustand zum nächsten fest – während die Datenbankdateien zu jeder Zeit einen bestimmten Zustand widerspiegeln, beinhalten die Protokolldateien all die vielen Änderungen auf dem Weg *zwischen* den Zuständen. Somit können sie sehr schnell wachsen und sich anhäufen.

Glücklicherweise hat die Datenbankumgebung beginnend mit der Version 4.2 der Berkeley DB die Fähigkeit, ihre eigenen unbenutzten Protokolldateien automatisch zu entfernen. Alle Projektarchive, die mit einem **svnadmin** angelegt wurden, das mit Berkeley DB Version 4.2 oder später übersetzt wurde, werden mit automatischer Protokolldateientfernung konfiguriert. Wenn Sie diese Funktion nicht möchten, geben Sie dem Befehl **svnadmin create** einfach die Option `--bdb-log-keep` mit. Sollten Sie das vergessen oder es sich später anders überlegen, editieren Sie einfach die Datei `DB_CONFIG` im Verzeichnis `db` Ihres Projektarchivs indem Sie die Zeile mit der Direktive `set_flags DB_LOG_AUTOREMOVE` auskommentieren und starten dann **svnadmin recover** auf Ihrem Projektarchiv, um die Konfigurationsänderung zu aktivieren. Siehe „Konfiguration von Berkeley DB“ für weitere Informationen zur Datenbankkonfiguration.

Ohne eine Art automatische Protokolldateientfernung aktiviert zu haben, häufen sich die Protokolldateien während der Nutzung des Projektarchivs an. Es ist eigentlich ein Merkmal des Datenbanksystems – Sie sollten ausschließlich mit Hilfe der Protokolldateien in der Lage sein, Ihre gesamte Datenbank zu rekonstruieren, so dass diese Protokolldateien sehr nützlich für eine Wiederherstellung im Katastrophenfall sein können. Jedoch möchten Sie normalerweise die nicht mehr von Berkeley DB verwendeten Protokolldateien archivieren und sie zur Platzersparnis von der Platte entfernen. Verwenden Sie den Befehl **svnadmin list-unused-dblogs**, um die unbenutzten Protokolldateien anzuzeigen:

```
$ svnadmin list-unused-dblogs /var/svn/repos
/var/svn/repos/log.0000000031
/var/svn/repos/log.0000000032
/var/svn/repos/log.0000000033
…
$ rm `svnadmin list-unused-dblogs /var/svn/repos`
## Plattenplatz zurückgewonnen!
```

BDB-basierte Projektarchive, deren Protokolldateien ein Bestandteil eines Sicherungs- oder Notfallplans sind, sollten *nicht* die automatische Entfernung verwenden. Die Wiederherstellung der Daten eines Projektarchivs kann

nur gewährleistet werden, wenn *alle* Protokolldateien verfügbar sind. Falls einige der Protokolldateien von der Platte entfernt werden, bevor das Sicherungssystem die Gelegenheit bekommt, sie woandershin zu kopieren, ist die unvollständige Menge gesicherter Protokolldateien tatsächlich nutzlos.

FSFS Filtersystem packen

Wie in der Anmerkung Revisionsdateien und Scherben beschrieben, erzeugen auf FSFS basierende Subversion-Projektarchive standardmäßig für jede dem Projektarchiv hinzugefügte Revision eine neue Datei auf der Platte. Wenn tausende dieser Dateien auf dem Subversion-Server liegen, kann dies ineffizient werden, selbst dann, wenn sie in getrennten Scherben-Verzeichnissen untergebracht sind.

Das erste Problem besteht darin, dass das Betriebssystem innerhalb kurzer Zeit auf viele verschiedene Dateien beziehen muss. Das führt zur ineffektiven Verwendung von Zwischenspeicherungen der Platte und hat zur Folge, dass bei großen Platten viel Zeit zum Suchen verbraucht wird. Daher kommt es beim Zugriff von Subversion auf versionierte Daten zu Leistungseinbußen.

Das zweite Problem ist etwas subtiler. Aufgrund der Art und Weise wie die meisten Dateisysteme Plattenplatz zuweisen, benötigt eine Datei mehr Platz, als sie eigentlich belegt. Der Umfang des zusätzlichen Platzes, um eine einzelne Datei unterzubringen, kann, je nach verwendetem Dateisystem durchschnittlich irgendwo zwischen 2 und 16 Kilobyte *pro Datei* liegen. Das bedeutet pro Revision eine Einbuße bei der Plattennutzung für Projektarchive auf FSFS Basis. Besonders deutlich wird der Effekt bei Projektarchiven mit vielen kleinen Revisionen, da hier platzmäßig die Kosten der Speicherung der Revisionsdatei schnell den Umfang der eigentlich zu speichernden Daten überschreiten.

Zur Lösung dieser Probleme führte Subversion 1.6 den Befehl **svnadmin pack** ein. Durch das Aneinanderfügen aller Dateien einer vollständigen Scherbe in eine einzelne „pack“-Datei und das anschließende Entfernen der ursprünglichen Dateien, die pro Revision angelegt wurden, verringert **svnadmin pack** die Anzahl der Dateien innerhalb einer gegebenen Scherbe auf nur eine einzelne Datei. Das kommt den Zwischenspeichern des Dateisystems entgegen und verringert die Anzahl der Kosten für Dateispeicherung auf eins.

Subversion kann bestehende zerlegte Projektarchive packen, die auf das Dateisystemformat von 1.6 aktualisiert worden sind (siehe svnadmin upgrade). Lassen Sie dafür einfach **svnadmin pack** über das Projektarchiv laufen:

```
$ svnadmin pack /var/svn/repos
Packe 0...erledigt.
Packe 1...erledigt.
Packe 2...erledigt.
…
Packe 34...erledigt.
Packe 35...erledigt.
Packe 36...erledigt.
$
```

Da der Packprozess die benötigten Sperren erlangt, bevor er seine Arbeit beginnt, können Sie ihn auf in Benutzung befindliche Projektarchive anwenden oder sogar als Teil eines Hooks nach der Übergabe. Das erneute Packen bereits gepackter Scherben ist legal, hat allerdings keine Auswirkungen auf den Plattenverbrauch des Projektarchivs.

svnadmin pack hat keine Auswirkungen auf Subversion-Projektarchive auf BDB-Basis.

Wiederherstellung von Berkeley DB

Wie in „Berkeley DB“ erwähnt wurde, kann ein Berkeley-DB-Projektarchiv manchmal einfrieren, falls es nicht ordnungsgemäß geschlossen wird. Wenn das passiert, muss ein Administrator die Datenbank in einen konsistenten Zustand zurückfahren. Das gilt aber nur für BDB-basierte Projektarchive – falls Sie FSFS-basierte verwenden, sind Sie davon nicht betroffen. Und falls Sie Subversion 1.4 mit Berkeley DB 4.4 oder später verwenden, werden Sie feststellen, dass Subversion für diese Situationen wesentlich unempfindlicher geworden ist. Trotzdem kommt es vor, dass sich Berkeley-DB-Projektarchive verklemmen, und Administratoren müssen wissen, wie sie sicher damit umgehen.

Um die Daten in Ihrem Projektarchiv zu schützen, verwendet Berkeley DB einen Sperrmechanismus. Dieser Mechanismus stellt sicher, dass Teile der Datenbank nicht gleichzeitig durch mehrere Zugriffe verändert werden und jeder Prozess die Daten

beim Lesen aus der Datenbank im korrekten Zustand sieht. Wenn ein Prozess irgendetwas in der Datenbank ändern muss, prüft er zunächst, ob eine Sperre auf den Zieldaten liegt. Sind die Daten nicht gesperrt, sperrt der Prozess die Daten, nimmt die Änderungen vor und entsperrt die Daten wieder. Andere Prozesse müssen auf die Freigabe der Sperre warten, bevor sie wieder auf diesen Datenbankabschnitt zugreifen dürfen. (Das hat nichts mit den Sperren zu tun, die Sie als Benutzer auf versionierte Dateien im Projektarchiv vergeben können; wir versuchen die Verwirrung, die durch diese Terminologie verursacht wird, in Die drei Bedeutungen von „Sperre“ zu klären.)

Während der Nutzung Ihres Projektarchivs können fatale Fehler oder Unterbrechungen einen Prozess daran hindern, die von ihm in der Datenbank gesetzten Sperren wieder zu entfernen. Als Ergebnis ist das Datenbanksystem „verklemmt“. Wenn das passiert, laufen alle Versuche ins Leere, auf die Datenbank zuzugreifen (da jeder neue Prozess darauf wartet, dass die Sperre entfernt wird – was aber nicht passieren wird).

Keine Panik, falls das Ihrem Projektarchiv widerfahren sollte! Das Berkeley-DB-Dateisystem nutzt die Vorteile von Datenbanktransaktionen, Sicherungspunkten sowie vorausschreibender Journalierung, um zu gewährleisten, dass nur die katastrophalsten Ereignisse[6] dauerhaft die Datenbankumgebung zerstören können. Ein ausreichend paranoider Projektarchiv-Administrator wird irgendwie Sicherungen der Daten des Projektarchivs an einem anderen Ort verwahren, doch rennen Sie noch nicht zum Schrank mit den Sicherungsbändern.

Verwenden Sie stattdessen das folgende Rezept, um Ihr Projektarchiv zu „entklemmen“:

1. Stellen Sie sicher, dass keine Prozesse auf das Projektarchiv zugreifen (oder einen Zugriffsversuch machen). Für netzbasierte Projektarchive bedeutet das, auch den Apache-HTTP-Server oder den svnserve-Dämon zu stoppen.

2. Melden Sie sich als der Benutzer an, dem das Projektarchiv gehört und der es verwaltet. Das ist wichtig, da eine Wiederherstellung unter einer falschen Benutzerkennung dazu führen kann, dass die Berechtigungen auf den Dateien eines Projektarchivs derart verändert werden können, dass der Zugriff auf das Projektarchiv auch dann nicht mehr möglich wird, wenn es „entklemmt“ ist.

3. Starten Sie den Befehl **`svnadmin recover /var/svn/repos`**. Sie sollten eine Ausgabe ähnlich dieser sehen:

   ```
   Exklusiven Zugriff auf das Projektarchiv erlangt
   Bitte warten, die Wiederherstellung des Projektarchivs kann einige Zeit dauern ...

   Wiederherstellung vollständig abgeschlossen.
   Die neueste Revision des Projektarchivs ist 19.
   ```

   Die Ausführung dieses Befehls kann viele Minuten dauern.

4. Machen Sie einen Neustart des Server-Prozesses.

Dieses Vorgehen behebt fast jeden Fall von Projektarchiv-Verklemmung. Stellen Sie sicher, dass Sie diesen Befehl als der Benutzer ausführen, der Eigentümer und Verwalter der Datenbank ist, nicht einfach als `root`. Ein Teil des Wiederherstellungsprozesses könnte diverse Datenbankdateien völlig neu erzeugen (z.B. gemeinsame Speicherbereiche). Wenn Sie die Wiederherstellung als `root` ausführen, werden diese Dateien dem Benutzer `root` zugeordnet, was bedeutet, dass selbst nach der Wiederherstellung der Verbindung zur Außenwelt gewöhnliche Benutzer keinen Zugriff mehr bekommen werden.

Falls das oben beschriebene Vorgehen aus irgendwelchen Gründen die Verklemmung Ihres Projektarchivs nicht beseitigt, sollten Sie zwei Dinge tun. Schieben Sie zunächst ihr beschädigtes Projektarchiv an die Seite (indem Sie es etwa in `repos.BROKEN` umbenennen) und spielen seine jüngste Sicherung ein. Schicken Sie dann eine E-Mail an die Subversion-Mailing-Liste (`<users@subversion.apache.org>`), in der Sie Ihr Problem detailliert beschreiben. Die Integrität der Daten genießt bei den Entwicklern von Subversion allerhöchste Priorität.

Projektarchiv-Daten woanders hin verschieben

Ein Subversion-Dateisystem hält seine Daten in Dateien, die auf eine Art und Weise über das Projektarchiv verstreut sind, die

[6] Beispielsweise Festplatte + starker Elektromagnet = Desaster.

im Allgemeinen nur die Subversion-Entwickler selbst verstehen (und auch nur sie interessieren). Allerdings können es bestimmte Umstände erforderlich machen, alle Daten oder nur Teile davon in ein anderes Projektarchiv zu kopieren oder zu verschieben.

Subversion stellt solche Funktionen durch *Projektarchiv-Auszugs-Datenströme* (repository dump streams) bereit. Ein Projektarchiv-Auszugs-Datenstrom (oft als „Auszugsdatei“ bezeichnet, wenn er als Datei auf Platte gespeichert wird) ist ein portables, flaches Dateiformat, das die zahlreichen Revisionen in Ihrem Projektarchiv beschreibt – was geändert wurde, von wem usw. Dieser Datenstrom ist der primäre Mechanismus zum Herumschieben der versionierten Historie – als Ganzes oder in Teilen, mit oder ohne Änderung – zwischen Projektarchiven. Und Subversion stellt die Werkzeuge zum Erzeugen und Laden dieser Datenströme zur Verfügung: die Unterbefehle **svnadmin dump** bzw. **svnadmin load**.

Obwohl das Format der Subversion Auszugsströme menschenlesbare Teile enthält und das Format eine gewohnte Struktur besitzt (es gleicht einem RFC 822 Format, das meistens für E-Mail verwendet wird), ist es *kein* reines Textformat. Es ist ein Binärformat, das sehr empfindlich gegenüber Herumgepfusche ist. Beispielsweise würden viele Texteditoren die Datei beschädigen, indem sie automatisch die Zeilenenden umformen.

Es gibt viele Gründe, Auszüge von Subversion-Projektarchiv-Daten zu machen und zu laden. In der Anfangsphase von Subversion war der häufigste Grund die Weiterentwicklung von Subversion an sich. Während Subversion reifte, gab es Zeiten, als Änderungen an der Datenbankbasis zu Kompatibilitätsproblemen mit früheren Projektarchiv-Versionen führten, so dass Benutzer mit der vorherigen Version von Subversion Auszüge von ihren Projektarchiv-Daten machen und sie mit der neueren Version von Subversion in ein frisch erzeugtes Projektarchiv laden mussten. Diese Schemaänderungen haben seit Subversion 1.0 nicht mehr stattgefunden, und die Subversion-Entwickler versprechen, dass die Benutzer zwischen Unterversionen von Subversion (wie etwa von 1.3 nach 1.4) keine Abzüge ihrer Projektarchive machen und neu laden müssen. Jedoch gibt es noch andere Gründe, die es erforderlich machen, zu denen Dinge gehören wie das erneute Aufsetzen eines Berkeley-DB-Projektarchivs auf einem neuen Betriebssystem oder einer CPU-Architektur, der Wechsel von einem Berkeley-DB-basierten auf ein FSFS-basiertes Projektarchiv oder (was wir später in diesem Kapitel in „Filtern der Projektarchiv-Historie“ behandeln werden) das Entfernen versionierter Daten aus der Projektarchiv-Historie.

Das Auszugsformat eines Subversion Projektarchivs beschreibt nur versionierte Änderungen. Es beinhaltet keine Informationen über unvollendete Transaktionen, von Benutzern gesetzte Sperren auf Pfade im Projektarchiv, Anpassungen an Projektarchiv- oder Server-Konfigurationen (inklusive Hook-Skripten) usw.

Welche Gründe für den Umzug der Projektarchiv-Historie für Sie auch immer eine Rolle spielen, die Verwendung der Unterbefehle **svnadmin dump** und **svnadmin load** sind der direkte Weg. **svnadmin dump** gibt ein Intervall von Projektarchiv-Revisionen im speziellen Subversion-Auszugsformat aus. Der Auszug wird zur Standardausgabe geschrieben, während Mitteilungen an die Standardfehlerausgabe gehen. Das erlaubt Ihnen, den Ausgabestrom in eine Datei umzuleiten, während Sie Statusausgaben im Terminalfenster verfolgen können. Zum Beispiel:

```
$ svnlook youngest myrepos
26
$ svnadmin dump myrepos > dumpfile
* Revision 0 ausgegeben.
* Revision 1 ausgegeben.
* Revision 2 ausgegeben.
…
* Revision 25 ausgegeben.
* Revision 26 ausgegeben.
```

Am Ende haben Sie eine einzelne Datei (im vorangegangenen Beispiel `dumpfile`), die alle im Projektarchiv gespeicherten Daten aus dem gewählten Intervall von Revisionen beinhaltet. Beachten Sie, dass **svnadmin dump** wie jeder andere „lesende“ Prozess (z.B. **svn checkout**) Revisionsbäume aus dem Projektarchiv liest, so dass Sie diesen Befehl jederzeit aufrufen können.

Der andere Unterbefehl dieses Paars, **svnadmin load**, liest den Standardeingabestrom als eine Subversion-Projektarchiv-Auszugsdatei und spielt diese Revisionen aus dem Auszug gewissermaßen neu in das Ziel-Projektarchiv. Auch dieser Befehl erzeugt Meldungen, dieses Mal aber über die Standardausgabe:

```
$ svnadmin load newrepos < dumpfile
<<< Neue Transaktion basierend auf Originalrevision 1 gestartet
     * Füge Pfad hinzu: A ... erledigt.
     * Füge Pfad hinzu: A/B ... erledigt.
     …
------- Neue Revision 1 übertragen (geladen aus Original 1) >>>

<<< Neue Transaktion basierend auf Originalrevision 2 gestartet
     * Bearbeite Pfad: A/mu ... erledigt.
     * Bearbeite Pfad: A/D/G/rho ... erledigt.

------- Neue Revision 2 übertragen (geladen aus Original 2) >>>

…

<<< Neue Transaktion basierend auf Originalrevision 25 gestartet
     * Bearbeite Pfad: A/D/gamma ... erledigt.

------- Neue Revision 25 übertragen (geladen aus Original 25) >>>

<<< Neue Transaktion basierend auf Originalrevision 26 gestartet
     * Füge Pfad hinzu: A/Z/zeta ... erledigt.
     * Bearbeite Pfad: A/mu ... erledigt.

------- Neue Revision 26 übertragen (geladen aus Original 26) >>>
```

Das Ergebnis eines Ladevorgangs sind neue Revisionen, die dem Projektarchiv hinzugefügt wurden – dasselbe, was Sie erhalten, wenn Sie mit einem normalen Subversion-Client Übergaben an das Projektarchiv machen. Ebenso wie bei einer Übergabe können können Sie Hook-Programme verwenden, um Aktionen vor und nach jeder Übergabe während des Ladevorgangs auszuführen. Indem Sie die Optionen `--use-pre-commit-hook` und `--use-post-commit-hook` an **svnadmin load** übergeben, können Sie Subversion befehlen, für jede zu ladende Revision die Hook-Programme pre-commit bzw. post-commit auszuführen. Sie könnten diese beispielsweise verwenden, um sicherzustellen, dass die geladenen Revisionen dieselben Validierungsschritte durchlaufen müssen wie reguläre Übergaben. Natürlich sollten Sie diese Optionen mit Sorgfalt verwenden – wenn Ihr post-commit-Hook für jede neue Übergabe E-Mails an eine Mailing-Liste verschickt, wollen Sie bestimmt nicht, das innerhalb kürzester Zeit hunderte oder tausende Übergabe-E-Mails in diese Liste hineinhageln! Sie können mehr über Hook-Skripte in „Erstellen von Projektarchiv-Hooks“ lesen.

Beachten Sie, dass Menschen, die sich besonders gewitzt fühlen, weil **svnadmin** für den Auszug und den Ladevorgang den Standardeingabe- und den Standardausgabestrom benutzt, Dinge wie dieses ausprobieren können (vielleicht sogar unterschiedliche Versionen von **svnadmin** auf jeder Seite der Pipe):

```
$ svnadmin create newrepos
$ svnadmin dump oldrepos | svnadmin load newrepos
```

Im Normalfall wird die Auszugsdatei ziemlich groß – viel größer als das Projektarchiv selbst. Das liegt daran, dass standardmäßig jede Version jeder Datei als vollständiger Text in der Auszugsdatei dargestellt wird. Dies ist das schnellste und einfachste Verhalten, und es ist nett, wenn Sie die Auszugsdaten über eine Pipe direkt an einen weiteren Prozess weiterleiten (etwa ein Komprimierprogramm, ein Filterprogramm oder einen Prozess zum Laden). Wenn Sie jedoch eine Auszugsdatei für die Langzeitspeicherung erzeugen, möchten Sie wahrscheinlich Plattenplatz sparen, indem Sie die Option `--deltas` verwenden. Mit dieser Option werden aufeinanderfolgende Revisionen von Dateien als komprimierte binäre Unterschiede ausgegeben – so wie Dateirevisionen im Projektarchiv gespeichert werden. Diese Option ist langsamer, führt jedoch zu einer Größe der Auszugsdatei, die der Größe des Original-Projektarchivs näher kommt.

Wir haben eben erwähnt, dass **svnadmin dump** einen Bereich von Revisionen ausgibt. Verwenden Sie die Option `--revision` (`-r`), um eine einzelne Revision oder einen Bereich von Revisionen für den Auszug anzugeben. Wenn Sie diese Option weglassen, wird ein Auszug aller Projektarchiv-Revisionen erstellt.

```
$ svnadmin dump myrepos -r 23 > rev-23.dumpfile
$ svnadmin dump myrepos -r 100:200 > revs-100-200.dumpfile
```

Beim Erstellen eines Auszugs jeder Revision gibt Subversion gerade soviel Information aus, dass später ein Ladeprozess in der Lage ist, diese Revision auf der Basis der Vorgängerrevision wiederherzustellen. Mit anderen Worten: Für jede Revision befinden sich nur die Dinge in der Auszugsdatei, die sich in dieser Revision geändert haben. Die einzige Ausnahme von dieser Regel ist die erste Revision, die mit dem aktuellen **svnadmin dump** erstellt wird.

Standardmäßig wird Subversion den Auszug der ersten Revision nicht bloß als Unterschied ausdrücken, der auf die Vorgängerrevision anzuwenden ist. Zum Ersten gibt es keine Vorgängerrevision in der Auszugsdatei. Und zum Zweiten kann Subversion den Zustand des Projektarchivs, in das der Auszug (falls überhaupt) geladen werden soll, nicht kennen. Um sicherzustellen, dass die Ausgabe jedes Aufrufs von **svnadmin dump** unabhängig ist, ist der Auszug der ersten Revision standardmäßig eine vollständige Darstellung jedes Verzeichnisses, jeder Datei und jeder Eigenschaft aus dieser Revision im Projektarchiv.

Sie können dieses Standardverhalten jedoch ändern. Falls Sie die Option `--incremental` angeben, vergleicht **svnadmin** die erste Revision für die ein Auszug erstellt werden soll mit der vorhergehenden Revision im Projektarchiv – auf dieselbe Art und Weise, wie jede andere Revision behandelt wird, für die ein Auszug erstellt werden soll – indem lediglich die Änderungen aus dieser Revision erwähnt werden. Der Vorteil dabei ist, dass Sie mehrere kleinere Auszugsdateien erstellen können, die hintereinander geladen werden können, anstatt eine große:

```
$ svnadmin dump myrepos -r 0:1000 > dumpfile1
$ svnadmin dump myrepos -r 1001:2000 --incremental > dumpfile2
$ svnadmin dump myrepos -r 2001:3000 --incremental > dumpfile3
```

Diese Auszugsdateien können mit der folgenden Befehlsfolge in ein neues Projektarchiv geladen werden:

```
$ svnadmin load newrepos < dumpfile1
$ svnadmin load newrepos < dumpfile2
$ svnadmin load newrepos < dumpfile3
```

Ein weiterer toller Trick, den Sie mit der Option `--incremental` anwenden können besteht darin, einen neuen Bereich von Revisionsauszügen an eine existierende Revisionsdatei anzuhängen. Beispielsweise könnten Sie einen `post-commit`-Hook haben, der der Datei einen Auszug derjenigen Revision anfügt, die den Hook ausgelöst hat. Oder Sie haben ein Skript, das jede Nacht läuft, um Auszüge sämtlicher Revisionen seit dem letzten Lauf anzufügen. Wenn es auf diese Weise verwendet wird, stellt **svnadmin dump** eine Möglichkeit dar, laufend die Änderungen an Ihrem Projektarchiv für den Fall eines Systemabsturzes oder eines anderen katastrophalen Ereignisses zu sichern.

Das Auszugsformat kann auch dazu verwendet werden, um die Inhalte mehrerer verschiedener Projektarchive in ein Projektarchiv zusammenzuführen. Indem Sie die Option `--parent-dir` von **svnadmin load** benutzen, können Sie ein neues virtuelles Wurzelverzeichnis für den Ladevorgang angeben. Das heißt, falls Sie beispielsweise die Auszugsdateien von drei Projektarchiven haben – etwa `calc-dumpfile`, `cal-dumpfile` und `ss-dumpfile` – können Sie zunächst ein Projektarchiv anlegen, das alle beherbergt:

```
$ svnadmin create /var/svn/projects
$
```

Erstellen Sie dann neue Verzeichnisse im Projektarchiv, die den Inhalt der vorherigen drei Projektarchive aufnehmen werden:

```
$ svn mkdir -m "Initial project roots" \
      file:///var/svn/projects/calc \
      file:///var/svn/projects/calendar \
      file:///var/svn/projects/spreadsheet
Revision 1 übertragen.
$
```

Laden Sie schließlich die Auszugsdateien an ihren jeweiligen Ort im neuen Projektarchiv:

```
$ svnadmin load /var/svn/projects --parent-dir calc < calc-dumpfile
…
$ svnadmin load /var/svn/projects --parent-dir calendar < cal-dumpfile
…
$ svnadmin load /var/svn/projects --parent-dir spreadsheet < ss-dumpfile
…
$
```

Zum Schluss erwähnen wir noch einen Anwendungsfall für das Auszugsformat – die Umwandlung aus einem unterschiedlichen Speicherverfahren oder gar aus einem unterschiedlichen Versionskontrollsystem. Da das Format der Auszugsdatei größtenteils menschenlesbar ist, sollte es einfach sein, gewöhnliche Änderungsmengen – von denen jede als Revision behandelt werden sollte – mit diesem Format zu beschreiben. Tatsächlich verwendet das Dienstprogramm **cvs2svn** (siehe „Ein Projektarchiv von CVS nach Subversion überführen“) dieses Auszugsformat, um den Inhalt eines CVS-Projektarchivs darzustellen, so dass er in ein Subversion-Projektarchiv kopiert werden kann.

Filtern der Projektarchiv-Historie

Da Subversion Ihre versionierte Historie mindestens mit binären Differenzalgorithmen und Datenkompression abspeichert (optional in einem völlig undurchsichtigen Datenbanksystem), ist der Versuch manueller Eingriffe unklug, zumindest schwierig und unter allen Umständen nicht angeraten. Sobald Daten im Projektarchiv gespeichert sind, bietet Subversion im Allgemeinen keine einfache Möglichkeit, diese Daten zu entfernen.[7] Doch zwangsläufig werden sich Gelegenheiten ergeben, bei denen Sie die Historie Ihres Projektarchivs manipulieren müssen. Es könnte sein, dass Sie alle Instanzen einer Datei entfernen müssen, die versehentlich dem Projektarchiv hinzugefügt worden ist, aber aus welchen Gründen auch immer nicht hineingehört). [8] Oder Sie haben vielleicht mehrere Projekte, die sich ein Projektarchiv teilen und entscheiden sich nun, jedem Projekt sein eigenes Projektarchiv zu geben. Um Aufgaben wie diese bewerkstelligen zu können, benötigen Administratoren eine besser handhabbare und bearbeitbare Repräsentation der Daten in den Projektarchiven – das Subversion-Projektarchiv-Auszugsformat.

Wie bereits in „Projektarchiv-Daten woanders hin verschieben“ beschrieben, ist das Subversion-Projektarchiv-Auszugsformat eine menschenlesbare Wiedergabe der Änderungen, die Sie an Ihren versionierten Daten im Laufe der Zeit vorgenommen haben. Verwenden Sie den Befehl **svnadmin dump**, um den Auszug anzulegen und **svnadmin load**, um ein neues Projektarchiv damit zu füllen. Das Tolle an der Menschenlesbarkeit des Auszugsformates ist, dass Sie, sofern es Ihnen nicht egal ist, die Daten manuell untersuchen und verändern können. Natürlich besteht ein Nachteil darin, dass eine Auszugsdatei eines Projektarchivs, in das über drei Jahre Änderungen eingeflossen sind, riesig groß sein wird, und es Sie eine lange, lange Zeit kosten wird, die Daten manuell zu untersuchen und zu verändern.

Hierbei hilft **svndumpfilter**. Dieses Programm verhält sich wie ein pfadbasierter Filter für Auszugsströme. Geben Sie ihm einfach eine Liste von Pfaden mit, die Sie behalten möchten oder eine Liste von Pfaden, die Sie nicht behalten möchten, und leiten Sie Ihre Auszugsdaten durch diesen Filter. Das Ergebnis ist ein modifizierter Strom der Auszugsdaten, der nur die versionierten Pfade beinhaltet, die Sie (explizit oder implizit) verlangt haben.

Lassen Sie uns an einem realistischen Beispiel betrachten, wie Sie diesen Programm verwenden könnten. Früher in diesem Kapitel (siehe „Planung der Organisation Ihres Projektarchivs“) erörterten wir das Entscheidungsfindungsverfahren, wie Sie

[7] Das ist doch überhaupt der Grund dafür, Versionskontrolle einzusetzen, oder?

[8] Das bewusste, vorsichtige Entfernen bestimmter Teile versionierter Daten wird tatsächlich von wirklichen Anwendungsfällen verlangt. Das ist der Grund, warum eine „Auslösch“-Funktion eine der am häufigsten gewünschten Funktionen von Subversion ist, von der die Subversion-Entwickler hoffen, sie bald zur Verfügung stellen zu können.

Ihre Daten im Projektarchiv anordnen sollen – ein Projektarchiv pro Projekt oder kombiniert, wie Sie die Daten im Projektarchiv verteilen usw. Doch manchmal, nachdem bereits einige Revisionen hinzugekommen sind, überdenken Sie die Anordnung und würden gerne einige Änderungen vornehmen. Eine verbreitete Änderung ist die Entscheidung, mehrere Projekte, die sich ein Projektarchiv teilen, auf getrennte Projektarchive pro Projekt aufzuteilen.

Unser imaginäres Projektarchiv beinhaltet drei Projekte: `calc`, `calendar` und `spreadsheet`. Sie waren miteinander in der folgenden Anordnung abgelegt:

```
/
  calc/
    trunk/
    branches/
    tags/
  calendar/
    trunk/
    branches/
    tags/
  spreadsheet/
    trunk/
    branches/
    tags/
```

Um diese drei Projekte in ihre eigenen Projektarchive zu bekommen, erstellen wir zunächst einen Auszug des gesamten Projektarchivs:

```
$ svnadmin dump /var/svn/repos > repos-dumpfile
* Revision 0 ausgegeben.
* Revision 1 ausgegeben.
* Revision 2 ausgegeben.
* Revision 3 ausgegeben.
…
$
```

Dann leiten wir die Auszugsdatei durch die Filter, wobei jedes Mal nur jeweils eins der obersten Verzeichnisse ausgewählt wird. Als Ergebnis erhalten wir drei Auszugsdateien:

```
$ svndumpfilter include calc < repos-dumpfile > calc-dumpfile
…
$ svndumpfilter include calendar < repos-dumpfile > cal-dumpfile
…
$ svndumpfilter include spreadsheet < repos-dumpfile > ss-dumpfile
…
$
```

An dieser Stelle müssen sie eine Entscheidung treffen. Jede Ihrer Auszugsdateien wird ein gültiges Projektarchiv erzeugen, allerdings unter Beibehaltung der Pfade wie sie im ursprünglichen Projektarchiv waren. Das bedeutet, dass, obwohl Sie ein Projektarchiv ausschließlich für Ihr `calc` Projekt haben, wird es immer noch ein Wurzelverzeichnis namens `calc` besitzen. Falls Sie möchten, dass die Verzeichnisse `trunk`, `tags` und `branches` direkt im Wurzelverzeichnis Ihres Projektarchivs liegen, sollten Sie Ihre Auszugsdateien editieren, indem Sie die Einträge `Node-path` und `Node-copyfrom-path` verändern, so dass sie nicht mehr die erste Komponente `calc/` im Pfad haben. Sie sollten auch den Abschnitt entfernen, der das Verzeichnis `calc` anlegt. Es sollte etwa wie folgt aussehen:

```
Node-path: calc
Node-action: add
Node-kind: dir
Content-length: 0
```

Falls Sie sich entscheiden sollten, die Auszugsdatei manuell zu editieren, um eins der obersten Verzeichnisse zu entfernen, sollten Sie sicherstellen, dass Ihr Editor nicht automatisch Zeilenenden in das native Format umwandelt (z.B. `\r\n` in `\n`), da sonst der Inhalt nicht zu den Metadaten passt. Das würde Ihre Auszugsdatei nutzlos machen.

Alles, was jetzt noch übrig bleibt, ist, Ihre drei neuen Projektarchive zu erstellen und jede Auszugsdatei in das richtige Projektarchiv zu laden, wobei die UUID aus dem Auszugsstrom ignoriert wird:

```
$ svnadmin create calc
$ svnadmin load --ignore-uuid calc < calc-dumpfile
<<< Neue Transaktion basierend auf Originalrevision 1 gestartet
     * Füge Pfad hinzu: Makefile ... erledigt.
     * Füge Pfad hinzu: button.c ... erledigt.
…
$ svnadmin create calendar
$ svnadmin load --ignore-uuid calendar < cal-dumpfile
<<< Neue Transaktion basierend auf Originalrevision 1 gestartet
     * Füge Pfad hinzu: Makefile ... erledigt.
     * Füge Pfad hinzu: cal.c ... erledigt.
…
$ svnadmin create spreadsheet
$ svnadmin load --ignore-uuid spreadsheet < ss-dumpfile
<<< Neue Transaktion basierend auf Originalrevision 1 gestartet
     * Füge Pfad hinzu: Makefile ... erledigt.
     * Füge Pfad hinzu: ss.c ... erledigt.
…
$
```

Beide Unterbefehle von **svndumpfilter** akzeptieren Optionen, die angeben, wie „leere“ Revisionen behandelt werden sollen. Falls eine Revision nur Änderungen an herausgefilterten Pfaden beinhaltet, könnte die neue Revision als uninteressant oder gar unerwünscht gelten. Um dem Benutzer die Kontrolle darüber zu geben, wie hiermit verfahren werden soll, bietet **svndumpfilter** die folgenden Kommandozeilenoptionen:

`--drop-empty-revs`
: Überhaupt keine leeren Revisionen erzeugen – einfach auslassen.

`--renumber-revs`
: Falls leere Revisionen ausgelassen werden (mit der Option `--drop-empty-revs`), die Nummern der übrig gebliebenen Revisionen ändern, so dass keine Lücken in der Nummernfolge auftreten.

`--preserve-revprops`
: Falls leere Revisionen nicht ausgelassen werden, die Eigenschaften der leeren Revisionen bewahren (Protokolleintrag, Autor, Datum, Eigenschaften usw.). Sonst beinhalten leere Revisionen lediglich den Zeitstempel und einen erzeugten Protokolleintrag, der darauf hinweist, dass diese Revision von **svndumpfilter** geleert wurde.

Obwohl **svndumpfilter** sehr nützlich und eine Zeitersparnis sein kann, gibt es unglücklicherweise ein paar Fallstricke. Erstens ist das Dienstprogramm überempfindlich gegenüber der Pfadsemantik. Achten Sie darauf, ob die Pfade in Ihrer Auszugsdatei mit oder ohne führende Schrägstriche angegeben werden. Sie sollten sich die Einträge `Node-path` und `Node-copyfrom-path` ansehen.

```
…
Node-path: spreadsheet/Makefile
…
```

Falls die Pfade führende Schrägstriche haben, sollten auch Sie Schrägstriche in den Pfaden angeben, die Sie an **svndumpfilter include** und **svndumpfilter exclude** übergeben (und wenn sie keine haben, sollten Sie auch keine angeben). Falls Ihre Auszugsdatei aus irgendwelchen Gründen einen nicht konsistenten Gebrauch von führenden Schrägstrichen macht, [9] sollten Sie diese Pfade normalisieren, so dass sie alle entweder Schrägstriche haben oder nicht.

Ebenso können kopierte Pfade Probleme bereiten. Subversion unterstützt Kopieroperationen im Projektarchiv, bei denen ein neuer Pfad erzeugt wird, indem ein bereits bestehender kopiert wird. Es kann vorkommen, dass Sie zu irgendeinem Zeitpunkt der Lebenszeit Ihres Projektarchivs eine Datei oder ein Verzeichnis von einer durch **svndumpfilter** ausgelassenen Stelle an eine durch **svndumpfilter** berücksichtigte Stelle kopiert haben. Um die Auszugsdateien unabhängig zu machen, muss **svndumpfilter** trotzdem das Hinzufügen des neuen Pfades anzeigen – mit dem Inhalt aller durch die Kopie erzeugten Dateien – allerdings nicht als eine Kopie aus einer Quelle, die es gar nicht im gefilterten Auszugsstrom gibt. Da allerdings das Subversion Auszugsdateiformat nur Änderungen von Revisionen beinhaltet, kann es sein, dass der Inhalt der Quelle der Kopie nicht verfügbar ist. Wenn Sie mutmaßen, dass Sie solche Kopien in Ihrem Projektarchiv haben, sollten Sie die Auswahl der ausgelassenen/berücksichtigten Pfade überdenken, indem Sie vielleicht die Pfade, die als Quellen für die problematischen Kopien dienten, hinzunehmen.

Schließlich behandelt **svndumpfilter** Pfadfilterung ziemlich wörtlich. Wenn Sie die Historie eines Projektes mit dem Wurzelverzeichnis `trunk/my-project` kopieren und sie in ein eigenes Projektarchiv verschieben möchten, werden Sie selbstverständlich den Befehl **svndumpfilter include** verwenden, um alle Änderungen in und unterhalb von `trunk/my-project` zu bewahren. Doch macht die entstehende Auszugsdatei keinerlei Annahmen bezüglich des Projektarchivs, in das Sie die Daten zu laden beabsichtigen. In diesem besonderen Fall könnten die Auszugsdaten mit der Revision beginnen, die das Verzeichnis `trunk/my-project` hinzugefügt hat, doch sie werden *keine* Direktiven enthalten, dir das Verzeichnis `trunk` selbst anlegen (weil `trunk` nicht zum Filter der zu berücksichtigenden Pfade passt). Sie müssen sicherstellen, dass alle Verzeichnisse, die der Auszugsstrom erwartet, tatsächlich im Ziel-Projektarchiv vorhanden sind, bevor Sie versuchen, den Strom in dieses Projektarchiv zu laden.

Projektarchiv Replikation

Es gibt mehrere Szenarien, in denen es sehr passend ist, ein Subversion-Projektarchiv zu haben, dessen Versionshistorie genau dieselbe wie die eines anderen Projektarchivs ist. Vielleicht das offensichtlichste ist die Aufrechterhaltung eines Projektarchivs als einfache Sicherheitskopie, das verwendet wird, wenn das primäre Projektarchiv wegen Materialdefekt, Netzausfall oder ähnlichen Ärgernissen unzugänglich geworden ist. Andere Szenarien umfassen den Einsatz von Spiegel-Projektarchiven, um heftige Subversion-Last über mehrere Server zu verteilen, zum sanften Aufrüsten usw.

Subversion stellt ein Programm zur Handhabung solcher Szenarien zur Verfügung – **svnsync**. Im Wesentlichen funktioniert das, indem der Subversion-Server aufgefordert wird, Revisionen zu „wiederholen“, eine nach der anderen. Dann wird die Information dieser Revision benutzt, um eine Übergabe derselben an ein anderes Projektarchiv zu imitieren. Keins der Projektarchive muss lokal auf der Maschine liegen, auf der **svnsync** läuft – seine Parameter sind Projektarchiv-URLs, und es verrichtet seine gesamte Arbeit über die Projektarchiv-Access-Schnittstellen (RA) von Subversion. Das Einzige, was benötigt wird, ist Lesezugriff auf das Quell-Projektarchiv und Lese-/Schreibzugriff auf das Ziel-Projektarchiv.

Wenn Sie **svnsync** mit einem entfernt liegenden Quell-Projektarchiv verwenden, muss auf dem Subversion-Server für dieses Projektarchiv Subversion 1.4 oder neuer laufen.

Angenommen, Sie haben bereits ein Projektarchiv, das Sie gerne spiegeln möchten. Als nächstes brauchen Sie ein leeres Ziel-Projektarchiv, das als Spiegel dienen soll. Dieses Projektarchiv kann eins der verfügbaren Speicherverfahren benutzen (siehe „Auswahl der Datenspeicherung“), doch es darf noch keine Versionshistorie enthalten. Das von **svnsync** verwendete Protokoll zur Übermittlung der Revisionsinformation ist sehr empfindlich gegenüber nicht übereinstimmenden Versionshistorien im Quell- und Ziel-Projektarchiv. Aus dem Grund, dass **svnsync** nicht *verlangen* kann, dass das Ziel-Projektarchiv nur lesbar ist,[10] ist die Katastrophe programmiert, wenn erlaubt wird, die Revisions-Historie im Ziel-Projektarchiv mit anderen Mitteln

[9] Obwohl **svnadmin dump** ein konsistentes Vorgehen bezüglich führender Schrägstriche vorweisen kann (indem es sie nicht einfügt), sind andere Programme, die Auszugsdateien erzeugen eventuell nicht so konsistent.

als durch das Spiegeln zu verändern.

Verändern Sie ein Spiegel-Projektarchiv *nicht* auf eine Art und Weise, die dazu führt, dass die Versionshistorie von der des Original-Projektarchivs abweicht. Die einzigen Übergaben und Änderungen an Revisions-Eigenschaften die in diesem Spiegel-Projektarchiv stattfinden, sollten ausschließlich durch den Befehl **svnsync** vorgenommen werden.

Eine weitere Anforderung an das Ziel-Projektarchiv ist, dass dem **svnsync**-Prozess erlaubt wird, Revisions-Eigenschaften zu verändern. Da **svnsync** im Rahmen des Hook-Systems ausgeführt wird, ist der standardmäßige Zustand des Projektarchivs (welcher keine Änderungen an Revisions-Eigenschaften zulässt; siehe pre-revprop-change) nicht ausreichend. Sie müssen ausdrücklich den pre-revprop-change-Hook bereitstellen, der **svnsync** erlaubt, Revisions-Eigenschaften zu definieren und zu ändern. Mit diesen Vorkehrungen sind Sie gerüstet, um Projektarchiv-Revisionen zu spiegeln.

Es ist eine gute Idee, Autorisierungsmaßnahmen zu ergreifen, um Ihrem Projektarchiv-Replikations-Prozess die Arbeit zu ermöglichen, wohingegen anderen Benutzern die Veränderung der Inhalte des Spiegel-Projektarchivs verwehrt wird.

Lassen Sie uns nun die Benutzung von **svnsync** bei einem Rundgang in einem typischen Spiegel-Szenario erklären. Wir werden diesen Diskurs mit Empfehlungen würzen, die Sie jedoch getrost missachten können, falls sie für Ihre Umgebung nicht benötigt werden oder nicht passend sind.

Wir wollen das öffentliche Subversion-Projektarchiv spiegeln, das den Quelltext des vorliegenden Buchs beherbergt, und diesen Spiegel von einer anderen Maschine als der, auf der das ursprüngliche Subversion-Quelltext-Projektarchiv untergebracht ist, im Internet veröffentlichen. Dieser entfernt liegende Rechner besitzt eine globale Konfiguration, die es anonymen Benutzern erlaubt, den Inhalt von Projektarchivs auf diesem Rechner zu lesen, aber zum Ändern dieser Projektarchive eine Authentifizierung der Benutzer erforderlich macht. (Vergeben Sie uns bitte, dass wir für den Augenblick über die Details der Subversion-Server-Konfiguration hinwegsehen – sie werden in Kapitel 6, *Konfiguration des Servers* behandelt.) Und aus dem alleinigen Grund, es noch interessanter machen zu wollen, werden wir den Replikations-Prozess von einer dritten Maschine aus steuern – diejenige, die wir aktuell benutzen.

Zunächst erstellen wir das Projektarchiv, das unser Spiegel sein soll. Dieser und die folgenden paar Schritte erfordern einen Shell-Zugang auf die Maschine, die das Spiegel-Projektarchiv beherbergen soll. Sobald das Projektarchiv jedoch konfiguriert ist, sollten wir nicht mehr direkt darauf zugreifen müssen.

```
$ ssh admin@svn.example.com "svnadmin create /var/svn/svn-mirror"
admin@svn.example.com's password: ********
$
```

Zu diesem Zeitpunkt haben wir unser Projektarchiv, und wegen unserer Server-Konfiguration ist das Projektarchiv nun „live“ im Internet. Da wir aber außer unserem Replikations-Prozess niemanden erlauben wollen, das Projektarchiv zu ändern, benötigen wir eine Möglichkeit, diesen Prozess von anderen potentiellen Zugriffen zu unterscheiden. Um dies zu machen, verwenden wir einen ausgezeichneten Benutzernamen für unseren Prozess. Nur Übergaben und Änderungen an Revisions-Eigenschaften unter dem Anwenderkonto `syncuser` werden erlaubt.

Wir verwenden das Hook-System des Projektarchivs sowohl, um dem Replikations-Prozess seine Arbeit zu ermöglichen, als auch, um sicherzustellen, dass nur er diese Dinge tut. Wir bewerkstelligen dies, indem wir zwei der Projektarchiv-Ereignis-Hooks implementieren – pre-revprop-change und start-commit. Unser `pre-revprop-change`-Hook-Skript finden Sie in Beispiel 5.2, „pre-revprop-change-Hook-Skript des Spiegel-Projektarchivs“; grundsätzlich stellt es sicher, dass der Benutzer, der die Eigenschaften ändern möchte, unser `syncuser` ist. Falls dies zutrifft, ist die Änderung erlaubt, anderenfalls wird die Änderung abgelehnt.

Beispiel 5.2. pre-revprop-change-Hook-Skript des Spiegel-Projektarchivs

[10] Tatsächlich kann es gar nicht nur lesbar sein, denn sonst hätte **svnsync** ein echtes Problem, die Versionshistorie hineinzukopieren.

```
#!/bin/sh

USER="$3"

if [ "$USER" = "syncuser" ]; then exit 0; fi

echo "Ausschließlich der Benutzer syncuser darf Revisions-Eigenschaften ändern" >&2
exit 1
```

Das deckt Änderungen an Revisions-Eigenschaften ab. Nun müssen wir sicherstellen, dass nur der Benutzer `syncuser` neue Revisionen an das Projektarchiv übergeben darf. Wir machen das, indem wir ein `start-commit`-Hook-Skript wie das in Beispiel 5.3, „start-commit-Hook-Skript des Spiegel-Projektarchivs“ benutzen.

Beispiel 5.3. start-commit-Hook-Skript des Spiegel-Projektarchivs

```
#!/bin/sh

USER="$2"

if [ "$USER" = "syncuser" ]; then exit 0; fi

echo "Ausschließlich der Benutzer syncuser darf neue Revisionen übergeben" >&2
exit 1
```

Nachdem wir unsere Hook-Skripte installiert und uns vergewissert haben, dass sie auf dem Subversion-Server ausführbar sind, sind wir mit dem Aufsetzen des Spiegel-Projektarchivs fertig. Nun kommen wir zum eigentlichen Spiegeln.

Das Erste, was wir machen müssen ist, unserem Ziel-Projektarchiv mit **svnsync** zu sagen, dass es ein Spiegel des Quell-Projektarchivs sein wird. Wir machen das mit dem Unterbefehl **svnsync initialize**. Die URLs, die wir mitgeben, zeigen auf die Wurzelverzeichnisse des Ziel- bzw. Quell-Projektarchivs. In Subversion 1.4 ist das erforderlich – nur die vollständige Spiegelung von Projektarchiven ist erlaubt. Beginnend mit Subversion 1.5 jedoch können Sie **svnsync** auch zum Spiegeln von Teilbäumen des Projektarchivs verwenden.

```
$ svnsync help init
initialize (init): Aufruf: svnsync initialize ZIEL_URL QUELL_URL

Bereitet ein Zielprojektarchiv auf die Synchronisation mit einem
anderen Projektarchiv vor.
…
$ svnsync initialize http://svn.example.com/svn-mirror \
                     http://svnbook.googlecode.com/svn \
                     --sync-username syncuser --sync-password syncpass
Eigenschaften für Revision 0 kopiert (svn:sync-* Eigenschaften übergangen).
$
```

Unser Ziel-Projektarchiv wird sich nun erinnern, dass es ein Spiegel des öffentlichen Subversion-Quelltext-Projektarchivs ist. Beachten Sie, dass wir einen Benutzernamen und ein Passwort an **svnsync** übergeben haben – das war für den pre-revprop-change-Hook in unserem Spiegel-Projektarchiv erforderlich.

In Subversion 1.4 wurden die an die Kommandozeilenoptionen `--username` und `--password` von **svnsync** übergebenen Werte sowohl für die Authentisierung gegenüber dem Quell-Projektarchiv als auch gegenüber dem Ziel-Projektarchiv verwendet. Das führte zu Problemen, falls die Zugangsdaten eines Benutzers nicht für beide Projektarchive identisch waren, insbesondere im nicht-interaktiven Modus (mit der Option `--non-interactive`). Dies ist in Subversion 1.5 mit der Einführung von zwei neuen Optionspaaren behoben

worden. Benutzen Sie `--source-username` und `--source-password` für die Zugangsdaten des Quell-Projektarchivs sowie `--sync-username` und `--sync-password` für das Ziel-Projektarchiv. (Die alten Optionen `--username` und `--password` bleiben aus Kompatibilitätsgründen bestehen, doch raten wir von deren Verwendung ab.)

Und nun kommt der lustige Teil. Mit einem einfachen Unterbefehl können wir **svnsync** auffordern, alle bislang ungespiegelten Revisionen aus dem Quell-Projektarchiv zum Ziel zu kopieren.[11] Der Unterbefehl **svnsync synchronize** wird die bereits vorher im Ziel-Projektarchiv gespeicherten besonderen Revisions-Eigenschaften untersuchen und sowohl ermitteln, welches Projektarchiv es spiegelt und dass die zuletzt gespiegelte Revision die Revision 0 war. Dann fragt es das Quell-Projektarchiv ab, welches die jüngste Revision in diesem Projektarchiv ist. Schließlich fordert es den Server des Quell-Projektarchivs auf, alle Revisionen zwischen 0 und dieser letzten Revision zu wiederholen. Sobald **svnsync** die entsprechende Antwort vom Quell-Projektarchiv-Server erhält, leitet es diese Revisionen als neue Übergaben an den Server des Ziel-Projektarchivs weiter.

```
$ svnsync help synchronize
synchronize (sync): Aufruf: svnsync synchronize ZIEL_URL

Überträgt alle laufenden Revisionen von der Quelle, mit der es
initialisiert wurde, zum Ziel.
…
$ svnsync synchronize http://svn.example.com/svn-mirror
Revision 1 übertragen.
Eigenschaften für Revision 1 kopiert.
Revision 2 übertragen.
Eigenschaften für Revision 2 kopiert.
Übertrage Daten .
Revision 3 übertragen.
Eigenschaften für Revision 3 kopiert.
…
Übertrage Daten .
Revision 4063 übertragen.
Eigenschaften für Revision 4063 kopiert.
Übertrage Daten .
Revision 4064 übertragen.
Eigenschaften für Revision 4064 kopiert.
Übertrage Daten .
Revision 4065 übertragen.
Eigenschaften für Revision 4065 kopiert.
$
```

Von besonderem Interesse ist hier, dass für jede gespiegelte Revision zunächst eine Übergabe der Revision an das Ziel-Projektarchiv erfolgt und dann die Änderungen der Eigenschaften folgen. Diese zweiphasige Replizierung ist notwendig, da die anfängliche Übergabe durch den Benutzer `syncuser` durchgeführt (und ihm auch zugeschrieben) wird und mit dem Zeitstempel der Erzeugung dieser Revision versehen wird. **svnsync** hat hinterher unmittelbaren Serie von Änderungen an den Eigenschaften vorzunehmen, die all die Eigenschaften dieser Revision vom Quell-Projektarchiv ins Ziel-Projektarchiv kopieren, was auch den Effekt hat, dass der Autor und der Zeitstempel so korrigiert werden, dass diese den entsprechenden Werten im Quell-Projektarchiv entsprechen.

Bemerkenswert ist ebenfalls, dass **svnsync** eine sorgfältige Buchführung vornimmt, die es ihm erlaubt, sicher unterbrochen und erneut gestartet zu werden, ohne die Integrität der gespiegelten Daten zu gefährden. Falls während des Spiegelns ein Netzproblem entsteht, wiederholen Sie einfach den Befehl **svnsync synchronize**, und er wird einfach damit weitermachen, womit er aufgehört hat. Das ist tatsächlich genau das, was Sie machen, um Ihren Spiegel aktuell zu halten, wenn neue Revisionen im Quell-Projektarchiv auftauchen.

svnsync-Buchhaltung

svnsync muss in der Lage sein, Revisions-Eigenschaften im Ziel-Projektarchiv zu setzen und zu verändern, da diese Eigenschaften Teil der Daten sind, die es spiegeln soll. Wenn sich diese Eigenschaften im Quell-Projektarchiv ändern,

[11] Seien Sie jedoch vorgewarnt, dass, obwohl der durchschnittliche Leser nur ein paar Sekunden benötigt, um diesen Absatz und die ihm folgende Beispielausgabe zu erfassen, die tatsächlich für eine vollständige Spiegelung erforderliche Zeit um Einiges länger ist.

müssen sie im Ziel-Projektarchiv nachgezogen werden. Allerdings verwendet **svnsync** auch eine Menge von speziellen Revisions-Eigenschaften – gespeichert in Revision 0 des Spiegel-Projektarchivs – für seine eigene interne Buchhaltung. Diese Eigenschaften beinhalten Informationen wie etwa der URL oder die UUID des Quell-Projektarchivs plus einige Informationen zur Zustandsverfolgung.

Ein Teil dieser Zustandsverfolgungsinformation ist ein Flag, das bedeutet: „momentan findet eine Synchronisierung statt“. Dies wird verwendet, um zu verhindern, dass mehrere **svnsync**-Prozesse miteinander kollidieren, während sie versuchen, Daten in dasselbe Ziel-Projektarchiv zu spiegeln. Im Allgemeinen brauchen Sie auf *keins* dieser besonderen Eigenschaften zu achten (sie beginnen alle mit dem Präfix `svn:sync-`). Gelegentlich jedoch, falls eine Synchronisierung unerwartet fehlschlägt, bekommt Subversion keine Chance, dieses besondere Zustands-Flag zu entfernen. Das führt dazu, dass alle weiteren Synchronisierungsversuche fehlschlagen, da es scheint, dass eine Synchronisierung gerade durchgeführt wird, obwohl tatsächlich keine stattfindet. Glücklicherweise kann dies behoben werden, indem einfach die Eigenschaft `svn:sync-lock` von Revision 0 des Spiegel-Projektarchivs entfernt wird, das als dieses Flag dient.

```
$ svn propdel --revprop -r0 svn:sync-lock http://svn.example.com/svn-mirror
Eigenschaft »svn:sync-lock« wurde von Revision 0 im Projektarchiv gelöscht
$
```

Dass **svnsync** den URL des Quell-Projektarchivs zur Buchhaltung in einer Eigenschaft des Spiegel-Projektarchivs speichert, ist der Grund dafür, dass Sie diesen URL nur einmal angeben müssen: bei **svnsync init**. Künftige Synchronisierungs-Operationen mit diesem Spiegel lesen einfach die besondere Eigenschaft `svn:sync-from-url`, das auf dem Spiegel gespeichert ist, um zu wissen, woher synchronisiert werden soll. Dieser Wert wird vom Synchronisierungsprozess jedoch wortwörtlich benutzt. Seien Sie vorsichtig, wenn Sie nicht vollqualifizierte Domainnamen verwenden (etwa wenn Sie sich auf `svnbook.red-bean.com` einfach mit `svnbook` beziehen, da das funktioniert, wenn Sie direkt mit dem `red-bean.com` Netz verbunden sind), Domainnamen, die sich nicht auflösen lassen oder sich unterschiedlich auflösen, je nachdem von wo aus Sie arbeiten oder IP-Adressen (die sich im Lauf der Zeit ändern können). Aber auch hier können Sie die Eigenschaft zur Buchhaltung ändern, falls ein bestehender Spiegel einen unterschiedlichen URL für dasselbe Quell-Projektarchiv benötigt:

```
$ svn propset --revprop -r0 svn:sync-from-url NEUER-QUELL-URL \
      http://svn.example.com/svn-mirror
Eigenschaft »svn:sync-from-url« wurde für Revision 0 im Projektarchiv gesetzt
$
```

Eine weitere interessante Angelegenheit an dieser speziellen Eigenschaft zur Buchhaltung ist, dass **svnsync** nicht versucht, diese Eigenschaften zu spiegeln, wenn sie im Quell-Projektarchiv gefunden werden. Der Grund ist wahrscheinlich offensichtlich, aber im Grunde läuft es darauf hinaus, dass **svnsync** nicht zu unterscheiden vermag, welche der speziellen Eigenschaften es bloß aus dem Quell-Projektarchiv kopiert hat und welche es für seine Buchhaltung benötigt und verwaltet. Diese Situation kann auftreten, falls Sie beispielsweise einen Spiegel eines Spiegels eines dritten Projektarchivs vorhalten. Wenn **svnsync** seine eigenen speziellen Eigenschaften in Revision 0 des Quell-Projektarchivs entdeckt, ignoriert es sie einfach.

Ein Unterbefehl **svnsync info** wurde in Subversion 1.6 hinzugefügt, um die besonderen Buchhaltungs-Eigenschaften im Ziel-Projektarchiv anzeigen zu lassen.

```
$ svnsync help info
Aufruf: svnsync info ZIEL_URL

Gibt Informationen über das Zielprojektarchiv der Synchronisation aus,
das sich unter ZIEL_URL befindet.
…
$ svnsync info http://svn.example.com/svn-mirror
Quell-URL: http://svnbook.googlecode.com/svn
UUID des Quellprojektarchivs: 931749d0-5854-0410-9456-f14be4d6b398
```

```
Letzte zusammengeführte Revision: 4065
$
```

In diesem Prozess ist jedoch eine kleine Unfeinheit. Da die Revisions-Eigenschaften von Subversion jederzeit während der Lebenszeit eines Projektarchivs geändert werden können, ohne zu protokollieren, wann sie geändert wurden, müssen replizierende Prozesse ein besonderes Augenmerk auf sie richten. Wenn Sie bereits die ersten 15 Revisionen eines Projektarchivs gespiegelt haben, und dann jemand eine Revisions-Eigenschaft von Revision 12 ändert, weiß **svnsync** nicht, dass es zurückgehen und die Kopie der Revision 12 korrigieren muss. Sie müssen es ihm manuell mitteilen, indem Sie den Unterbefehl **svnsync copy-revprops** verwenden, der einfach alle Eigenschaften einer bestimmten Revision oder eines Revisionsintervalls erneut repliziert.

```
$ svnsync help copy-revprops
copy-revprops: Aufruf: svnsync copy-revprops ZIEL_URL [REV[:REV2]]

Kopiert die Revisionseigenschaften in einem gegebenen Revisionsbereich
von der Quelle, mit der es initialisiert wurde, auf das Ziel.
…
$ svnsync copy-revprops http://svn.example.com/svn-mirror 12
Eigenschaften für Revision 12 kopiert.
$
```

Das ist Projektarchiv-Replikation in aller Kürze. Sehr wahrscheinlich möchten Sie einen solchen Prozess etwas automatisieren. Während unser Beispiel ein Ziehen-und-Schieben-Szenario beschrieb, möchten Sie vielleicht, dass Ihr primäres Projektarchiv als Teil der post-commit- und post-revprop-change-Hooks Änderungen an einen oder mehrere ausgesuchte Spiegel weiterschiebt. Das würde es ermöglichen, dass der Spiegel beinahe in Echtzeit aktuell gehalten werden kann.

Es ist auch möglich, wenn auch nicht sehr verbreitet, dass **svnsync** Projektarchive spiegelt, in denen der Benutzer unter dessen Kennung es läuft, nur eingeschränkte Rechte besitzt. Es werden dann einfach nur die Teile des Projektarchivs kopiert, die der Benutzer sehen darf. Offensichtlich taugt so ein Spiegel nicht als Sicherheitskopie.

In Subversion 1.5 entwickelte **svnsync** auch die Fähigkeit, eine Teilmenge eines Projektarchivs statt des Ganzen zu spiegeln. Das Anlegen und Pflegen eines solchen Spiegels unterscheidet sich nicht vom Spiegeln eines kompletten Projektarchivs; anstatt den Wurzel-URL des Quell-Projektarchivs bei **svnsync init** anzugeben, nennen Sie einfach den URL eines Unterverzeichnisses dieses Projektarchivs. Hierbei gibt es allerdings einige Einschränkungen. Als Erstes können Sie nicht mehrere disjunkte Unterverzeichnisse des Quell-Projektarchivs in ein einzelnes Ziel-Projektarchiv spiegeln – stattdessen müssen Sie ein Eltern-Verzeichnis spiegeln, das allen gemeinsam ist. Zum Zweiten ist die Filterlogik vollständig pfadbasiert, so dass bei Verzeichnissen, die in der Vergangenheit einmal umbenannt wurden, Ihr Spiegel nur die Revisionen seit dem Zeitpunkt enthält an dem das Verzeichnis unter diesem URL zu finden war. Auch wenn das Unterverzeichnis künftig umbenannt wird, werden Revisionen nur bis zu dem Zeitpunkt gespiegelt, an dem der URL ungültig wird.

Was das Zusammenspiel von Benutzern mit Projektarchiven und Spiegeln betrifft, *ist* es möglich eine einzelne Arbeitskopie zu haben, die mit beiden kommuniziert, doch müssen Sie hierfür einige Verrenkungen machen. Zunächst müssen Sie sicherstellen, dass sowohl das primäre Projektarchiv als auch das Spiegel-Projektarchiv dieselbe Projektarchiv-UUID haben (was standardmäßig nicht der Fall ist). Mehr darüber unter „Verwaltung von Projektarchiv UUIDs“ später in diesem Kapitel.

Sobald beide Projektarchive dieselbe UUID haben, können Sie **svn switch** mit der Option `--relocate` benutzen, um das Projektarchiv auszuwählen, mit dem Sie arbeiten wollen; dieser Prozess ist in svn switch (sw) beschrieben. Eine mögliche Gefahr besteht allerdings, wenn das Haupt- und das Spiegel-Projektarchiv nicht zeitnah synchronisiert sind. Eine Arbeitskopie, die auf das Haupt-Projektarchiv zeigt und gegenüber diesem aktuell ist, wird nach dem Umschalten auf den nicht aktuellen Spiegel durch den plötzlichen Verlust von Revisionen, die sie dort erwartet, verwirrt werden und deshalb Fehler ausgeben. Falls dies auftritt, können Sie entweder Ihre Arbeitskopie wieder zurück auf das Haupt-Projektarchiv schalten und warten bis das Spiegel-Projektarchiv aktuell ist oder Ihre Arbeitskopie auf eine Revision zurücksetzen, von der Sie wissen, dass sie im synchronisierten Projektarchiv vorhanden ist, und dann noch einmal das Umschalten versuchen.

Zum Schluss sollte Ihnen bewusst sein, dass die von **svnsync** angebotene revisionsbasierte Replikation genau das ist – die

Replikation von Revisionen. Nur die durch das Format der Subversion-Auszugsdateien übertragene Information ist replizierbar. Somit hat **svnsync** dieselben Einschränkungen wie der Auszugsstrom und beinhaltet nicht Dinge wie Hook-Implementierungen, Projektarchiv- oder Server-Konfigurationen, unvollständige Transaktionen oder Anwendersperren auf Projektarchiv-Pfaden.

Sicherung des Projektarchivs

Trotz zahlreicher technischer Fortschritte seit der Geburt des modernen Computers bleibt eine Sache unglücklicherweise wahr: manchmal geht etwas richtig schief. Eine kleine Auswahl von schlimmen Dingen, die das Schicksal auch auf den gewissenhaftesten Administrator loslassen kann, sind Stromausfälle, Netzzusammenbrüche, defekter Speicher und Festplattenabstürze. So kommen wir zu einem sehr wichtigen Thema: Wie mache ich Sicherheitskopien von den Daten meines Projektarchivs?

Dem Administrator stehen zwei Arten von Sicherungsmethoden zur Verfügung: vollständig und inkrementell. Eine vollständige Sicherungskopie des Projektarchivs beinhaltet eine umfassende Speicherung aller Informationen, die für die Wiederherstellung des Projektarchivs im Katastrophenfall benötigt werden. Dies bedeutet gewöhnlich eine Kopie des gesamten Projektarchiv-Verzeichnisses (inklusive der Berkeley-DB- oder FSFS-Umgebung). Inkrementelle Sicherungen haben einen geringeren Umfang: nur die Teile des Projektarchivs, die sich seit der letzten Sicherung geändert haben.

Was eine vollständige Sicherung betrifft, scheint der naive Ansatz vernünftig zu sein; jedoch besteht beim einfachen rekursiven Kopieren des Verzeichnisses das Risiko, eine fehlerhafte Sicherung zu erstellen, sofern nicht alle anderen Zugriffe auf das Projektarchiv verhindert werden. Für Berkeley DB beschreibt die Dokumentation eine bestimmte Reihenfolge, in der die Datenbankdateien kopiert werden können, um eine gültige Sicherungskopie zu gewährleisten. Eine ähnliche Reihenfolge gibt es für FSFS-Daten. Allerdings brauchen Sie diese Algorithmen nicht selbst zu implementieren, da das Subversion-Entwicklerteam das bereits getan hat. Der Befehl **svnadmin hotcopy** kümmert sich um die Details, die für eine Sicherungskopie während des Betriebes erforderlich sind. Der Aufruf ist so trivial wie die Bedienung von Unix' **cp** oder Windows' **copy**:

```
$ svnadmin hotcopy /var/svn/repos /var/svn/repos-backup
```

Das Ergebnis der Sicherung ist ein vollständig funktionsfähiges Subversion-Projektarchiv, das jederzeit die Aufgaben Ihres Projektarchivs übernehmen kann, falls irgendetwas Schlimmes passieren sollte.

Bei der Erstellung von Kopien eines Berkeley-DB-Projektarchivs können Sie **svnadmin hotcopy** sogar mitteilen, nach Abschluss der Kopie unbenötigte Berkeley-DB-Protokolldateien (siehe „Entfernen unbenutzter Protokolldateien von Berkeley DB“) aus dem Original-Projektarchiv zu löschen. Geben Sie einfach die Option `--clean-logs` auf der Kommandozeile an.

```
$ svnadmin hotcopy --clean-logs /var/svn/bdb-repos /var/svn/bdb-repos-backup
```

Ein zusätzliches Werkzeug für diesen Befehl steht auch zur Verfügung. Im Verzeichnis `tools/backup/` des Subversion-Quelltextpaketes liegt das Skript **hot-backup.py**. Dieses Skript ergänzt **svnadmin hotcopy** um ein wenig Sicherungsverwaltung, indem es Ihnen erlaubt, lediglich eine konfigurierbare Anzahl der letzten Sicherungskopien jedes Projektarchivs zu behalten. Es verwaltet automatisch die Namen der gesicherten Projektarchiv-Verzeichnisse, um Kollisionen mit vorherigen Sicherungen zu vermeiden und löscht ältere Sicherungen, so dass nur die jüngsten übrig bleiben. Selbst wenn Sie ebenfalls eine inkrementelle Sicherung haben, sollten Sie dieses Programm regelmäßig aufrufen. Sie könnten beispielsweise **hot-backup.py** mit einem Programmstarter (so wie **cron** auf Unix Systemen) verwenden, der es jede Nacht (oder in einem Zeitintervall, das Ihnen sicher erscheint) aufruft.

Einige Administratoren verwenden einen unterschiedlichen Sicherungsmechanismus, der auf der Erzeugung und Speicherung von Projektarchiv-Auszugs-Daten basiert. In „Projektarchiv-Daten woanders hin verschieben“ haben wir beschrieben, wie **svnadmin dump** mit der Option `--incremental` verwendet werden kann, um eine inkrementelle Sicherung einer Revision oder eines Bereichs von Revisionen zu erstellen. Natürlich können Sie davon eine vollständige Sicherung bekommen, wenn Sie die Option `--incremental` weglassen. Der Vorteil dieser Methode besteht darin, dass das Format der gesicherten Information flexibel ist – es erfordert keine bestimmte Plattform, keinen bestimmten Typ eines versionierten Dateisystems, keine bestimmte Version von Subversion oder Berkeley DB. Diese Flexibilität kommt allerdings zu dem Preis, dass die Wiederherstellung der Daten sehr lange dauern kann – länger mit jeder neuen Revision, die ins Projektarchiv übergeben wird.

Wie bei vielen verschiedenen Sicherungsmethoden werden auch hier Änderungen an Revisions-Eigenschaften bereits gesicherter Revisionen nicht berücksichtigt, sofern es sich um eine nicht-überlappende inkrementelle Sicherung handelt. Wir raten aus diesen Gründen davon ab, sich ausschließlich auf Sicherungsstrategien zu verlassen, die alleine auf Auszügen basieren.

Wie Sie sehen können, hat jeder der verschiedenen Sicherungstypen seine Vor- und Nachteile. Bei weitem am einfachsten ist die vollständige Sicherungskopie im laufenden Betrieb, die stets ein perfektes, einsatzfähiges Abbild Ihres Projektarchivs erzeugt. Falls Ihrem Projektarchiv irgendetwas Schlimmes widerfahren sollte, können Sie es durch eine einfache rekursive Verzeichniskopie aus der Sicherung wiederherstellen. Falls Sie mehrere Sicherungen Ihres Projektarchivs vorhalten, benötigt leider jede dieser vollständigen Kopien genauso viel Plattenplatz wie das Original. Im Gegensatz dazu lassen sich inkrementelle Sicherungen schneller erzeugen und platzsparender sichern. Allerdings kann die Wiederherstellung eine Plage sein, da oft mehrere inkrementelle Sicherungen eingespielt werden müssen. Andere Methoden wiederum haben auch ihre Besonderheiten. Administratoren müssen das Gleichgewicht zwischen den Kosten der Sicherung und den Kosten der Wiederherstellung finden.

Das Programm **svnsync** (siehe „Projektarchiv Replikation“) bietet tatsächlich einen handlichen Ansatz dazwischen. Falls Sie regelmäßig einen nur lesbaren Spiegel mit Ihrem Haupt-Projektarchiv synchronisieren, stellt der Spiegel einen ausgezeichneten Kandidaten dar, um für Ihr Haupt-Projektarchiv einzuspringen, falls es mal umkippt. Der Hauptnachteil dieses Ansatzes besteht darin, dass nur versionierte Projektarchiv-Daten synchronisiert werden – Projektarchiv-Konfigurationsdateien, benutzerdefinierte Sperren auf Projektarchiv-Pfaden und andere Dinge, die sich zwar im physikalischen Projektarchiv-Verzeichnis befinden können, jedoch nicht *innerhalb* des virtuellen versionierten Dateisystems des Projektarchivs, werden durch **svnsync** nicht berücksichtigt.

In jedem Sicherungsszenario müssen sich Projektarchiv-Administratoren bewusst sein, inwiefern Änderungen an unversionierten Revisions-Eigenschaften Auswirkungen auf die Sicherungen haben. Da diese Änderungen allein keine Revisionen erzeugen, werden auch keine post-commit-Hooks ausgelöst; es kann sogar sein, dass die Hooks pre-revprop-change und post-revprop-change nicht ausgelöst werden. [12] Und da Sie Revisions-Eigenschaften ohne Rücksicht auf die zeitliche Abfolge ändern können – Sie können jederzeit die Eigenschaften jeder Revision ändern – könnte eine inkrementelle Sicherung der letzten paar Revisionen eine Änderung an einer Revision aus einer vorangegangenen Sicherung übersehen.

Im Allgemeinen braucht nur ein echter Paranoiker nach jeder Übergabe eine vollständige Sicherung des Projektarchivs. Eine vollständige Sicherheitskopie des Projektarchivs im laufenden Betrieb im Rahmen einer systemweiten, nächtlichen Sicherung sollte ein Projektarchiv-Administrator jedoch erwägen, unter der Voraussetzung, dass das Projektarchiv bereits irgendeinen Redundanzmechanismus mit der nötigen Granularität verwendet (etwa Übergabe-E-Mails oder inkrementelle Auszüge). Es sind Ihre Daten – schützen Sie sie, wie es Ihnen passt.

Oftmals ist der beste Ansatz für die Projektarchiv-Sicherung ein diversifizierter, der die Stärken von Kombinationen der hier beschriebenen Methoden ausspielt. Die Subversion-Entwickler beispielsweise sichern jede Nacht das Subversion-Quelltext-Projektarchiv mit **hot-backup.py** und einem **rsync** dieser vollständigen Sicherungen von einem entfernten Standort aus; sie halten mehrere Archive aller Übergabe- und Eigenschafts-Änderungs-E-Mails vor und sie haben Spiegel des Projektarchivs, die von Freiwilligen mit **svnsync** verwaltet werden. Ihre Lösung könnte ähnlich aussehen, sollte aber Ihren Bedürfnissen entsprechen und das empfindliche Gleichgewicht zwischen Bequemlichkeit und Paranoia aufrechterhalten. Egal, was Sie machen: überprüfen Sie Ihre Sicherungen ab und an – was nutzt ein Reservereifen mit einem Loch? Obwohl all das Ihr Material nicht vor der eisernen Faust des Schicksals zu retten vermag, sollte es Ihnen sicherlich helfen, sich aus diesen schwierigen Zeiten zu erholen.

Verwaltung von Projektarchiv UUIDs

Subversion-Projektarchive haben eine mit ihnen verknüpfte, universelle, eindeutige Identifizierung (universally unique identifier, UUID). Dieser UUID wird von Subversion-Clients verwendet, um die Identität eines Projektarchivs zu verifizieren, falls andere Methoden nicht ausreichend sind (wie die Überprüfung des Projektarchiv-URLs, der sich im Lauf der Zeit ändern kann). Selten, wenn überhaupt, müssen sich Subversion-Projektarchiv-Administratoren weitergehende Gedanken über Projektarchiv UUIDs machen, anstatt sie als triviales Implementierungsdetail von Subversion zu betrachten. Manchmal jedoch gibt es einen Grund, der Aufmerksamkeit für dieses Detail verlangt.

Im Allgemeinen möchten Sie, dass die UUIDs Ihrer aktiven Projektarchive eindeutig sind. Das ist schließlich der Sinn von UUIDs. Jedoch gibt es Gelegenheiten, bei denen Sie möchten, dass die UUIDs zweier Projektarchive identisch sind. Wenn Sie beispielsweise zu Sicherungszwecken eine Kopie eines Projektarchivs machen, möchten Sie, dass die Sicherungskopie ein perfektes Abbild des Originals ist, so dass die Benutzer nach einer Wiederherstellung des Projektarchivs aus der Sicherheitskopie nicht das Gefühl haben, es mit einem unterschiedlichen Projektarchiv zu tun zu haben. Beim Erstellen bzw.

[12] **svnadmin setlog** kann auf eine Art aufgerufen werden, dass die Hook-Schnittstelle völlig umgangen wird.

beim Laden eines Auszugs der Projektarchiv-Historie (wie oben in „Projektarchiv-Daten woanders hin verschieben“ beschrieben) können Sie entscheiden, ob der im Auszugsstrom befindliche UUID auf das Projektarchiv angewendet werden soll, in das Sie die Daten laden. Die besonderen Umstände diktieren hier das richtige Verhalten.

Eine Projektarchiv-UUID kann auf verschiedene Art und Weise gesetzt (oder zurückgesetzt) werden, falls sie es müssen. Seit Subversion 1.5 wird einfach der Befehl **svnadmin setuuid** verwendet. Wenn Sie diesem Befehl einen ausdrücklichen UUID mitgeben, wird die Wohlgeformtheit des UUID überprüft und der UUID des Projektarchivs auf diesen Wert gesetzt. Wenn Sie den UUID weglassen, wird ein nagelneuer UUID für Ihr Projektarchiv erzeugt.

```
$ svnlook uuid /var/svn/repos
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$ svnadmin setuuid /var/svn/repos   # neuen UUID erzeugen
$ svnlook uuid /var/svn/repos
3c3c38fe-acc0-11dc-acbc-1b37ff1c8e7c
$ svnadmin setuuid /var/svn/repos \
            cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec  # alten UUID wiederherstellen
$ svnlook uuid /var/svn/repos
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$
```

Für Benutzer älterer Versionen als Subversion 1.5 sieht die Sache etwas komplizierter aus. Sie können den UUID eines Projektarchivs ausdrücklich setzen, indem Sie einen Projektarchiv-Auszugs-Fragment mit dem neuen UUID durch den Befehl **`svnadmin load --force-uuid REPOS-PATH`** leiten.

```
$ svnadmin load --force-uuid /var/svn/repos <<EOF
SVN-fs-dump-format-version: 2

UUID: cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
EOF
$ svnlook uuid /var/svn/repos
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$
```

Die Erzeugung eines nagelneuen UUID mit älteren Versionen von Subversion gestaltet sich jedoch nicht so einfach. Am besten finden Sie eine andere Möglichkeit zum Erzeugen des UUIDs und setzen anschließend den Projektarchiv-UUID auf diesen Wert.

Verschieben und Entfernen von Projektarchiven

Sämtliche Daten eines Subversion-Projektarchivs befinden sich innerhalb des Projektarchiv-Verzeichnisses. Als solches können Sie ein Subversion-Projektarchiv an einen anderen Ort auf der Platte verschieben, ein Projektarchiv umbenennen, kopieren oder vollständig löschen, indem Sie die Werkzeuge Ihres Betriebssystems zum Manipulieren von Verzeichnissen verwenden – **mv**, **cp -a** und **rm -r** auf Unix-Plattformen; **copy**, **move** und **rmdir /s /q** unter Windows; eine riesige Anzahl an Maus- und Menüoperationen in verschiedenen graphischen Dateiverwaltungs-Anwendungen, usw.

Natürlich gehört bei derartigen Änderungen mehr dazu, wenn deren Auswirkungen sauber behandelt werden sollen. Beispielsweise sollten Sie ihre Server-Konfiguration aktualisieren, so dass sie auf den neuen Ort des verschobenen Projektarchivs zeigt oder die Konfigurationseiträge für ein nun gelöschtes Projektarchiv entfernen. Sollten Sie automatisierte Prozesse haben, die Informationen aus Ihrem oder über Ihr Projektarchiv veröffentlichen, sollten auch sie gegebenenfalls aktualisiert werden. Auch Hook-Skripte könnten eventuell eine Neukonfigurierung benötigen. Benutzer müssten vielleicht benachrichtigt werden. Die Liste könnte beliebig verlängert werden oder zumindest bis zu dem Punkt, dass Ihre um das Subversion-Projektarchiv gebauten Prozesse und Prozeduren berücksichtigt werden.

Im Fall eines kopierten Projektarchivs sollten Sie auch die Tatsache berücksichtigen, dass Subversion Projektarchiv-UUIDs zur Unterscheidung von Projektarchivs benutzt. Wenn Sie ein Subversion-Projektarchiv mit einem typischen rekursiven Kommandozeilen-Kopierprogramm kopieren, haben Sie nachher zwei völlig identische Projektarchive – einschließlich ihrer

UUIDs. Manchmal mag das erwünscht sein; anderenfalls müssen Sie für eins dieser identischen Projektarchivs einen neuen UUID erzeugen. Für weitere Informationen über Projektarchiv-UUIDs, siehe „Verwaltung von Projektarchiv UUIDs“.

Zusammenfassung

Sie sollten bis jetzt ein grundlegendes Verständnis darüber haben, wie Subversion-Projektarchive angelegt, konfiguriert und gewartet werden. Wir haben Ihnen die verschiedenen Werkzeuge vorgestellt, die Ihnen bei diesen Aufgaben helfen. Im Verlauf dieses Kapitels haben wir auf verbreitete Fallstricke bei der Verwaltung hingewiesen und Vorschläge zu deren Vermeidung gegeben.

Was jetzt noch bleibt ist, dass Sie entscheiden müssen, welche aufregenden Daten Sie in Ihrem Projektarchiv unterbringen und wie sie schließlich über das Netz verfügbar gemacht werden sollen. Das nächste Kapitel ist ganz dem Netz gewidmet.

Kapitel 6. Konfiguration des Servers

Der Zugriff auf ein Subversion-Projektarchiv kann problemlos von mehreren Clients, welche auf demselben Rechner wie Subversion laufen, gleichzeitig erfolgen – unter Verwendung von URLs mit dem `file://`-Schema. Aber typischerweise läuft der Subversion-Server auf einem separaten Rechner, und der Zugriff erfolgt von Clients auf vielen verschiedenen Computern aus der ganzen Firma – ja sogar der ganzen Welt.

In diesem Kapitel erklären wir, wie Sie ihr Subversion-Projektarchiv für den Fernzugriff von Clients fit machen. Wir werden ausführlich auf alle aktuell verfügbaren Servermechanismen von Subversion eingehen und über ihre Konfiguration und Verwendung reden. Nach dem Lesen dieses Kapitels sollten Sie in der Lage sein, zu entscheiden, welche Netzwerk-Konfiguration Ihren Bedürfnissen entspricht und wie diese auf ihrem Server eingerichtet wird.

Überblick

Subversion wurde mit einer abstrakten Projektarchiv-Zugriffs-Schicht entworfen. Dies bedeutet, dass auf ein Projektarchiv automatisiert von beliebigen Server-Prozessen zugegriffen werden kann, und die für Clients vorhandene „Projektarchiv-Zugriffs“-API (Programmierschnittstelle) erlaubt es Programmierern, Plugins zu entwickeln, die relevante Netzwerkprotokolle verstehen. Theoretisch ermöglicht dies Subversion, eine unbegrenzte Zahl an Netzwerkprotokollen zu verwenden. Praktisch gibt es heute allerdings nur zwei weitverbreitete Server.

Apache ist ein sehr beliebter Webserver, welcher mittels des **mod_dav_svn**-Moduls auf Projektarchive zugreifen und diese für Clients verfügbar machen kann. Verwendet wird dabei das WebDAV/DeltaV-Protokoll, welches eine Erweiterung von HTTP ist. Da Apache ein stark erweiterbarer Webserver ist, bietet er eine Menge an „frei verfügbaren“ Funktionen/Modulen, wie mittels SSL verschlüsselte Verbindungen, Protokollierung, sowie die Integration diverser Authentifikationssysteme von Drittanbietern und einen eingeschränkten Web-Browser-gestützten Projektarchiv-Lesezugriff.

In der anderen Ecke befindet sich **svnserve**: ein kleiner, leichtgewichtiger Server, der ein einfaches Netzwerkprotokoll für die Zugriffe der Clients verwendet. Da dieses Protokoll für die Verwendung mit Subversion entwickelt wurde und, im Gegensatz zu HTTP, zustandsorientiert ist, bietet es einen deutlich schnelleren Netzwerkzugriff – spart allerdings auch einige wichtige Funktionen aus. So bietet er eine SASL-basierte Verschlüsselung und Authentifikation, hat aber keine Protokollierungsfunktionen oder eingebauten Web-Browser-Zugriff. Wie auch immer, er ist extrem einfach einzurichten und für kleinere Teams, welche einfach nur schnell mit Subversion "loslegen" wollen, die beste Wahl.

Das Netzwerkprotokoll, das **svnserve** spricht, kann auch über eine SSH-Verbindung getunnelt werden. Diese Option zum Einsatz von **svnserve** unterscheidet sich erheblich von der traditionellen Nutzung von **svnserve**. SSH wird zur Verschlüsselung der gesamten Kommunikation verwendet. Ebenso zur Authentifikation, was die Verwendung von realen Anwenderkonten auf dem Subversion-Server notwendig macht (anders als beim einfachen **svnserve**, der seine eigene Anwenderverwaltung hat). Des weiteren ist es notwendig – da jeder angemeldete Nutzer einen eigenen **svnserve**-Prozess startet – einer Gruppe von lokalen Nutzern (aus Sicht der Rechtevergabe) vollen Zugriff auf das Projektarchiv via `file://` URLs zu ermöglichen. Pfad-basierte Zugriffskontrolle schließt sich in diesem Fall aus, da die Nutzer direkt auf die Datenbank-Dateien zugreifen.

Tabelle 6.1, „Vergleich der Serveroptionen für Subversion“ zeigt eine kurze Zusammenfassung der drei typischen Server-Konfigurationen.

Tabelle 6.1. Vergleich der Serveroptionen für Subversion

Funktion	Apache + mod_dav_svn	svnserve	svnserve via SSH
Authentifikationsmöglichkeiten	HTTP Basic oder Digest Auth, X.509 Zertifikate, LDAP, NTLM, oder jede andere für den Apache Webserver verfügbare Methode	CRAM-MD5 als Voreinstellung, LDAP, NTLM oder jede andere für SASL verfügbare Methode	SSH
Anwenderkonfigurationen	private Datei `users` oder jede andere für den Apache Webserver verfügbare Methode (LDAP, SQL, usw.)	private Datei `users` oder jede andere für SASL verfügbare Methode (LDAP, SQL, usw.)	lokale Anwenderkonten auf dem Server

Funktion	Apache + mod_dav_svn	svnserve	svnserve via SSH
Autorisierungsmöglichkeiten	Lese-/Schreibzugriff auf das komplette Projektarchiv oder pfadbasierte Rechtevergabe	Lese-/Schreibzugriff auf das komplette Projektarchiv oder pfadbasierte Rechtevergabe	Lese-/Schreibzugriff nur auf ganzes Projektarchiv einstellbar
Verschlüsselung	optional mit SSL (https)	optional mit der SASL-Funktionen	Bestandteil der SSH-Verbindung
Protokollierung	Protokollierung der Subversion-Aktivitäten auf hoher Ebene, dazu detaillierte Protokollierung auf der Ebene der HTTP-Anfragen	Nur Protokollierung der Aktivitäten auf hoher Ebenen	Nur Protokollierung der Aktivitäten auf hoher Ebenen
Interoperabilität	Zugriff durch andere WebDAV-Clients	Verbindung nur mit svn-Clients möglich	Verbindung nur mit svn-Clients möglich
web-basierte Anzeige des Projektarchivs	eingeschränkte Unterstützung, alternativ mittels Programmen von Drittanbietern, wie etwa ViewVC, erweiterbar	nur mittels Programmen von Drittanbietern, wie etwa ViewVC	nur mittels Programmen von Drittanbietern, wie etwa ViewVC
Master-Slave-Server Replizierungen	transparenter Schreib-Proxy vom Slave zum Master	beschränkt auf nur lesbare Slave-Server	beschränkt auf nur lesbare Slave-Server
Geschwindigkeit	ein wenig langsamer	ein wenig schneller	ein wenig schneller
Erstkonfiguration	eher komplexer	sehr einfach	durchschnittlich

Auswahl einer Serverkonfiguration

Also dann, welchen Server sollten Sie nun verwenden? Welcher ist der beste?

Auf diese Frage gibt es offensichtlich nicht die eine, richtige Antwort. Denn jedes Team stellt andere Anforderungen, und die verschieden Server bieten unterschiedliche Funktionen und Voraussetzungen. Das Subversion-Projekt selbst bevorzugt keinen der genannten Server oder betrachtet einen als etwas „offizieller“ als die anderen.

Wir beleuchten nun die einzelnen Gründe, die für die eine oder andere Konstellation sprechen, ebenso auch Gründe, welche vielleicht *gegen* eine der Möglichkeiten sprechen.

Der svnserve-Server

Gründe, die für eine Nutzung sprechen:

- Das Aufsetzen geht schnell und einfach.
- Das verwendete Netzwerkprotokoll ist zustandsorientiert und merklich schneller als WebDAV.
- Es müssen keine lokalen Anwenderkonten auf dem Server eingerichtet werden.
- Das Passwort wird nicht über das Netzwerk übertragen.

Gründe, warum Sie svnserve eventuell nicht verwenden wollen:

- Es gibt standardmäßig nur eine Authentifikationsmethode, das Netzwerkprotokoll ist unverschlüsselt und das Passwort wird vom Server im Klartext gespeichert. (Mit SASL können diese Probleme zwar umgangen werden, dies erfordert aber eine etwas aufwendigere Konfiguration.)
- Keine erweiterte Protokollierung.
- Keinen eingebauten Web-Browser-gestützten Lesezugriff. (Wenn Sie dies wünschen, müssen Sie einen eigenständigen Webserver sowie Projektarchiv-Browser-Software installieren.)

svnserve über SSH

Gründe, die für eine Nutzung sprechen:

- Das verwendete Netzwerkprotokoll ist zustandsorientiert und merklich schneller als WebDAV.
- Sie können bestehende Anwenderzugänge des SSH-Servers verwenden.
- Der gesamte Netzwerkverkehr ist verschlüsselt.

Gründe, warum Sie auf diese Konstellation eventuell verzichten wollen:

- Es steht nur eine Authentifikationsmöglichkeit zur Verfügung.
- Keine erweiterten Protokollierungsmöglichkeiten.
- Die verwendeten Nutzer müssen in derselben Anwendergruppe (auf dem Server) sein, oder sich einen SSH-Schlüssel teilen.
- Bei unsachgemäßer Verwendung kann es zu Problemen mit den Dateirechten kommen.

Der Apache HTTP Server

Gründe, die für eine Nutzung sprechen:

- Subversion hat damit Zugriff auf alle für den Apache verfügbaren Authentifikationsmethode (und das sind viele).
- Es müssen auf dem Server keine Anwenderkonten angelegt werden.
- Apache protokolliert nach Wunsch (fast) alles.
- Der Netzwerkverkehr kann mittels SSL verschlüsselt werden.
- In der Regel lässt sich das HTTP(S)-Protokoll problemlos durch Firewalls routen.
- Auf das Projektarchiv kann lesend auch via Web-Browser zugegriffen werden.
- Das Projektarchiv lässt sich als Netzlaufwerk einhängen (mounten). Änderungen an den Dateien unterliegen trotzdem der Versionskontrolle. (siehe „Autoversionierung“.)

Was gegen den Apache Webserver spricht:

- Er ist merklich langsamer als **svnserve**, da HTTP als zustandsloses Protokoll eine höhere Netzwerklast verursacht.
- Die Ersteinrichtung kann etwas schwierig sein.

Empfehlungen

Im Allgemeinen empfehlen die Autoren dieses Buches eine einfache **svnserve**-Installation für kleine Teams, denen an einer schnellen und unkomplizierten Nutzung von Subversion gelegen ist. Dies ist die Variante, welche sich am einfachsten einrichten und administrieren lässt. Sollte später Bedarf bestehen, so kann immer noch auf eine komplexere Servervariante gewechselt werden.

Es folgen einige allgemeine Empfehlungen und Tipps, basierend auf mehrjähriger Erfahrung in der Anwenderbetreuung:

- Falls Sie für ihr Team die einfachste Servervariante suchen, dann kommen Sie mit einer Standardinstallation von **svnserve** am schnellsten ans Ziel. Beachten Sie aber, dass der Inhalt ihres Projektarchivs im Klartext über das Netzwerk übertragen wird. Wenn Sie nur innerhalb ihres Firmennetzwerks oder eines VPNs arbeiten, so ist dies kein Beinbruch. Ist ihr

Projektarchiv allerdings vom Internet aus erreichbar, so sollten Sie eventuell sicherstellen, dass darin keine sensiblen Daten vorhanden sind (z.B. nur quelloffenen Code o.ä.), oder Sie legen noch einmal Hand an und verschlüsseln mittels SASL die Netzwerkverbindung zur ihrem Projektarchiv.

- Wenn Sie bereits über Systeme zur Authentifizierung (LDAP, Active Directory, NTLM, X.509 usw.) verfügen und Subversion in diese integrieren möchten, so bleibt Ihnen die Wahl zwischen einer Apache-gestützten Variante oder eines mit SASL vermählten **svnserve**.

- Wenn Sie sich für die Verwendung von Apache oder eines Standard-**svnserve** entschieden haben, dann legen Sie auf ihrem System einen einfachen **svn**-Nutzer an und lassen den Serverprozess unter diesem Nutzer laufen. Stellen Sie zudem sicher, dass das gesamte Verzeichnis mit dem Projektarchiv nur diesem **svn**-Nutzer gehört. Damit wird der Zugriff auf ihr Projektarchiv durch das Dateisystem des Serverbetriebssystems verwaltet, und nur der Serverprozess kann noch Änderungen daran vornehmen.

- Wenn Sie bereits über eine aus SSH-Zugängen bestehende Infrastruktur verfügen, und Ihre Nutzer auf dem Subversion-Server schon lokale Zugänge haben, dann ist die Verwendung einer **svnserve**-über-SSH-Lösung sinnvoll. Wir empfehlen diese Variante allerdings nur sehr ungern. Es ist im Allgemeinen sicherer, Ihren Nutzern nur durch **svnserve** oder Apache verwaltete Zugänge den Zugriff auf Ihr Projektarchiv zu ermöglichen und eben nicht mittels vollwertiger Anwenderzugänge auf dem Serversystem. Falls der Wunsch nach einer starken Netzwerkverschlüsselung Sie auf die Verwendung des SSH gebracht hat, dann empfehlen wir Ihnen stattdessen die Verwendung von Apache und SSL, bzw. die Kombination aus **svnserve** und SASL-Verschlüsselung.

- Lassen Sie sich bitte *nicht* von der Idee verführen, allen Ihren Nutzern direkten Zugriff auf das Projektarchiv mittels der `file://`-Methode zu geben. Auch wenn der Zugriff auf das Projektarchiv durch eine Netzwerkfreigabe erfolgt, bleibt es immer noch eine schlechte Idee. Dadurch wird jeglicher Sicherheitspuffer zwischen dem Nutzer und dem Projektarchiv entfernt: Ein Anwender kann ohne (oder auch mit) Absicht die Datenbank des Projektarchivs beschädigen. Es wird zudem schwierig, das Projektarchiv offline zu nehmen um eine Inspektion oder ein Upgrade durchzuführen. Zudem kann es Ihnen eine Menge Probleme mit den Dateirechten einbringen (siehe „Unterstützung mehrerer Zugriffsmethoden auf das Projektarchiv“). Beachten Sie bitte auch, dass dies einer der Gründe ist, warum wir vor der Verwendung der `svn+ssh://`-Methode für den Projektarchiv-Zugriff warnen. Vom Standpunkt der Sicherheit ist dies effektiv dasselbe wie die Verwendung von `file://` für den Zugriff durch lokale Anwender und kann zu denselben Problemen führen, wenn der Administrator nicht alle Vorsicht walten lässt.

svnserve, ein maßgefertigter Server

Das Programm **svnserve** ist ein leichtgewichtiger Server, welcher für die Kommunikation mit den Clients ein auf TCP/IP basierendes, zustandsorientiertes Protokoll verwendet. Um sich mit dem Server zu verbinden, verwenden die Clients entweder das `svn://`- oder das `svn+ssh://`-Schema. In diesem Abschnitt behandeln wir die unterschiedlichen Möglichkeiten, **svnserve** einzusetzen, wie sich die Clients am Server authentisieren und wie die passenden Zugangsrechte zum Projektarchiv korrekt eingerichtet werden.

Der Serverstart

Es gibt mehrere Möglichkeiten, **svnserve** zu starten:

- **svnserve** als eigenständigen Dienst (engl. daemon) starten und auf Anfragen von Clients reagieren lassen.

- **svnserve** bei Bedarf mit Hilfe des Unix-Dienstes **inetd** starten, wenn auf einem festgelegten Port Anfragen eines svn-Clients ankommen.

- Einen SSH-Server verwenden, um **svnserve** fallweise über einen verschlüsselten SSH-Tunnel zu betreiben.

- **svnserve** als Microsoft-Windows-Dienst laufen lassen.

- **svnserve** als launchd-Job laufen lassen.

Die folgenden Abschnitte werden diese Einsatzoptionen für **svnserve** detailliert erörtern.

svnserve als Unix-Dienst

Die einfachste Variante ist, **svnserve** als eigenständigen (Unix-)Dienst laufen zu lassen. Verwenden Sie hierfür die `-d` Option beim Aufruf:

```
$ svnserve -d
$               # svnserve läuft nun als Dienst und lauscht auf Port 3690
```

Wird **svnserve** als Dienst betrieben, können Sie mit den Optionen `--listen-port` und `--listen-host` festlegen, auf welchem Port und unter welchem Hostnamen er lauschen soll.

Wurde **svnserve** auf diese Weise erfolgreich gestartet, stehen nun alle Projektarchive auf dem Server für Nutzer im Netzwerk zur Verfügung. Für einen Zugriff muss ein Client den *absoluten* Pfad zum Projektarchiv im URL angeben. Ist das Projektarchiv beispielsweise im Verzeichnis `/var/svn/project1` gespeichert, so sieht ein entsprechender URL für den Zugriff folgendermaßen aus: `svn://host.example.com/var/svn/project1`. Um die Sicherheit zu erhöhen, kann **svnserve** beim Start mit der Option `-r` auf ein bestimmtes Verzeichnis beschränkt werden, so dass nur noch die darin liegenden Projektarchive im Netz verfügbar sind. Ein Beispiel:

```
$ svnserve -d -r /var/svn
…
```

Mit der `-r`-Option wird festgelegt, welches Verzeichnis vom **svnserve** bei Anfragen als Wurzelverzeichnis (engl. root) verwendet wird. Ein Client muss nun in seiner URL nur noch den Pfad relativ zum neuen Wurzelverzeichnis angeben, was die URL erheblich verkürzt und die Verzeichnisstruktur etwas verschleiert:

```
$ svn checkout svn://host.example.com/project1
…
```

svnserve über inetd starten

Wenn Sie **inetd** zum Starten des Prozesses verwenden wollen, so übergeben Sie **svnserve** beim Aufruf die Option `-i` (`--inetd`). Im folgenden Beispiel sehen wir die Ausgaben beim Aufruf von `svnserve -i` auf der Kommandozeile. Beachten Sie aber, dass dies nicht der Weg ist, wie der Dienst normalerweise gestartet wird – eine genaue Beschreibung, wie **svnserve** über **inetd** gestartet wird, folgt anschließend.

```
$ svnserve -i
( success ( 2 2 ( ) ( edit-pipeline svndiff1 absent-entries commit-revprops d\
epth log-revprops partial-replay ) ) )
```

Mit der `--inetd`-Option versucht **svnserve** mit dem Subversion-Client unter Verwendung eines speziellen Protokolls via `stdin` und `stdout` zu sprechen. Dies ist der normale Weg für ein Programm, welches über **inetd** gestartet wurde. Die IANA (Internet Assigned Numbers Authority) hat für das Subversion-Protokoll den Port 3690 reserviert – auf einem Unix-ähnlichen System fügen Sie einfach folgende Zeilen (wenn noch nicht vorhanden) in die Datei `/usw/services` ein:

```
svn             3690/tcp    # Subversion
svn             3690/udp    # Subversion
```

Wenn Sie den klassischen Unix-**inetd** verwenden, können Sie die folgende Zeile in die Datei `/usw/inetd.conf` einfügen:

```
svn stream tcp nowait svnowner /usr/bin/svnserve svnserve -i
```

Stellen Sie sicher, dass „svnowner“ der Nutzer ist, welcher alle notwendigen Zugriffsrechte auf ihre Projektarchive hat. Kommt nun eine Anfrage eines Subversion-Clients auf Port 3690 herein, so wird **inetd** einen **svnserve**-Prozess starten, um die Anfrage zu bedienen. Wahrscheinlich möchten Sie noch die `-r`-Option zur oben genannten Zeile hinzufügen, um einzuschränken, welche Projektarchive exportiert werden dürfen.

svnserve über einen Tunnel

Eine weitere Möglichkeit ist, **svnserve** mittels der `-t`-Option im Tunnel-Modus aufzurufen. Bei diesem Aufruf wird vorausgesetzt, dass ein anderes Programm für den Remote-Zugriff – etwa **rsh** oder **ssh** – den Nutzer bereits erfolgreich authentisiert hat, um nun einen privaten **svnserve**-Prozess als *dieser Nutzer* zu starten. (Beachten Sie, dass für Sie als Nutzer selten bis nie die Notwendigkeit bestehen wird, **svnserve** mit der `-t`-Option von Hand auf der Kommandozeile aufzurufen – der SSH-Dienst wird dies in der Regel für Sie machen.) **svnserve** wird sich nun normal verhalten (Abwicklung der Kommunikation über `stdin` und `stdout`) und davon ausgehen, dass alle Daten mit Hilfe des Tunnels zum Client weitergeleitet werden. Wird **svnserve** wie in diesem Fall durch ein Tunnel-Programm aufgerufen, ist es notwendig, dass der aufrufende Nutzer volle Lese- und Schreibrechte auf die Dateien der Projektarchiv-Datenbank hat. Es verhält sich dabei im Grunde genommen so, als wenn der Nutzer mit einem `file://`-URL auf ein Projektarchiv zugreifen würde.

Wir werden diese Option noch genauer in diesem Kapitel behandeln, und zwar in „Tunneln über SSH“.

svnserve als ein Dienst unter Windows

Gehört ihr Windows zur NT-Familie (Windows oder neuer), so können Sie **svnserve** auch als normalen Windows-Dienst laufen lassen. Dies ist wesentlich sinnvoller, als die Option `--daemon` (`-d`) zu verwenden und ihn als selbstständigen Dienst zu betreiben. Sie müssten dann immer eine Konsole (cmd) öffnen, den passenden Befehl aufrufen und die Konsole anschließend die ganze Zeit geöffnet lassen. Ein Windows-Dienst dagegen läuft im Hintergrund, kann bereits beim Hochfahren automatisch starten und lässt sich wie jeder andere Windows-Dienst mit demselben Administrationsprogramm starten und stoppen.

Es ist notwendig, den neuen Windows-Dienst unter Verwendung des Kommandozeilenprogramms **SC.EXE** einzurichten. Ähnlich der **inetd**-Konfigurationszeile müssen Sie den genauen Aufruf für den Start von **svnserve** festlegen:

```
C:\> sc create svn
        binpath= "C:\svn\bin\svnserve.exe --service -r C:\repos"
        displayname= "Subversion Server"
        depend= Tcpip
        start= auto
```

Hiermit erzeugen Sie einen neuen Windows-Dienst mit dem Namen `svn`, welcher jedes Mal das Programm **svnserve.exe** startet (und in diesem Fall `C:\repos` als Wurzelverzeichnis verwendet). In diesem Beispiel müssen jedoch einige wichtige Punkte beachtet werden.

Als erstes ist es wichtig, dass das Programm **svnserve.exe** immer mit der Option `--service` aufgerufen wird. Alle weiteren Optionen müssen in derselben Zeile folgen, allerdings dürfen sich widersprechende Option nicht verwendet werden – wie etwa `--daemon` (`-d`), `--tunnel` oder `--inetd` (`-i`). Optionen wie `-r` oder `--listen-port` sind hingegen in Ordnung. Zweitens, seien Sie beim Aufruf von **SC.EXE** mit Leerzeichen vorsichtig: Beim Schreiben der `Schlüssel= Wert`-Zeile darf zwischen `Schlüssel` und `=` kein Leerzeichen stehen, vor `Wert` muss genau ein Leerzeichen stehen. Seien Sie zuletzt auch bei der Verwendung von Leerzeichen innerhalb ihres Kommandozeilenaufrufes vorsichtig. Sollten Verzeichnisangaben etwa Leerzeichen (oder andere zu schützende Zeichen) enthalten, so umschließen Sie sie mit zusätzlichen doppelten Anführungszeichen:

```
C:\> sc create svn
        binpath= "\"C:\program files\svn\bin\svnserve.exe\" --service -r C:\repos"
        displayname= "Subversion Server"
        depend= Tcpip
        start= auto
```

Beachten Sie bitte auch, dass das Wort `binpath` etwas irreführend ist – sein Wert ist eine *Kommandozeile* und nicht der Pfad zu einem Programm. Dies ist der Grund, warum Sie vorhandene Leerzeichen mit doppelten Anführungszeichen schützen müssen.

Ist der Dienst erst einmal eingerichtet, können Sie ihn mit Hilfe von grafischen Programmen (etwa der Microsoft Management Console) stoppen, starten oder seinen Status abfragen. Alternativ steht ihnen auch die Kommandozeile zur Verfügung:

```
C:\> net stop svn
C:\> net start svn
```

Der Dienst kann natürlich auch wieder deinstalliert werden, indem Sie den Befehl **`sc delete svn`** aufrufen. Stoppen Sie den Dienst aber vorher! Das Programm **SC.EXE** kennt noch etliche andere nützliche Optionen und Parameter, ein Aufruf von **`sc /?`** verrät ihnen, welche das sind.

svnserve als ein launchd-Job

Mac OS X (10.4 und höher) verwendet **launchd** zur Prozessverwaltung (einschließlich Dämonen) sowohl systemweit als auch pro Anwender. Ein **launchd**-Job wird durch Parameter in einer XML-Datei als Eigenschaftsliste spezifiziert. und der Befehl **launchctl** wird verwendet, um den Lebenszyklus dieser Jobs zu verwalten.

Ist es als **launchd**-Job eingerichtet, wird **svnserve** bei Bedarf automatisch gestartet, sobald eingehender Subversion-Netzverkehr mit `svn://` abgewickelt werden muss. Das ist viel einfacher als eine Konfiguration, die voraussetzt, dass **svnserve** als ein langlaufender Hintergrundprozess manuell gestartet wird.

Um **svnserve** als einen **launchd**-Job einzurichten, erstellen Sie zunächst eine Jobdefinitionsdatei namens `/Library/LaunchDaemons/org.apache.subversion.svnserve.plist`. Beispiel 6.1, „Eine Beispieldefinition für einen svnserve launchd Job“ liefert ein Beispiel für eine solche Datei.

Beispiel 6.1. Eine Beispieldefinition für einen svnserve launchd Job

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>Label</key>
        <string>org.apache.subversion.svnserve</string>
        <key>ServiceDescription</key>
        <string>Host Subversion repositories using svn:// scheme</string>
        <key>ProgramArguments</key>
        <array>
            <string>/usr/bin/svnserve</string>
            <string>--inetd</string>
            <string>--root=/var/svn</string>
        </array>
        <key>UserName</key>
        <string>svn</string>
        <key>GroupName</key>
        <string>svn</string>
        <key>inetdCompatibility</key>
```

```
        <dict>
            <key>Wait</key>
            <false/>
        </dict>
        <key>Sockets</key>
        <dict>
            <key>Listeners</key>
            <array>
                <dict>
                    <key>SockServiceName</key>
                    <string>svn</string>
                    <key>Bonjour</key>
                    <true/>
                </dict>
            </array>
        </dict>
    </dict>
</plist>
```

Das Erlernen des **launchd**-Systems kann ein wenig herausfordernd sein. Glücklicherweise gibt es Dokumentation zu den in diesem Abschnitt beschriebenen Befehlen. Rufen Sie beispielsweise **`man launchd`** von der Kommandozeile auf, um die Handbuchseite zu **launchd** selbst zu sehen, **`man launchd.plist`**, für das Format der Job-Definition, usw.

Sobald die Jobdefinitionsdatei erstellt ist, können Sie den Job mit **launchctl load** aktivieren:

```
$ sudo launchctl load \
       -w /Library/LaunchDaemons/org.apache.subversion.svnserve.plist
```

Zur Klarstellung: Diese Aktion startet **svnserve** noch nicht. Sie teilt **launchd** bloß mit, wie **svnserve** gestartet werden soll, falls Netzverkehr auf dem `svn` Netzwerk-Port aufschlägt; es wird beendet nachdem der Verkehr abgewickelt worden sein wird.

Da wir möchten, dass **svnserve** ein systemweiter Dämon-Prozess ist, müssen wir **sudo** verwenden, um diesen Job als Administrator zu verwalten. Beachten Sie ebenfalls, dass die Schlüssel `UserName` und `GroupName` in der Definitionsdatei optional sind — werden sie weggelassen, wird der Job unter dem Konto des Anwenders laufen, den ihn geladen hat.

Den Job abzustellen ist ebenso einfach — mit **launchctl unload**:

```
$ sudo launchctl unload \
       -w /Library/LaunchDaemons/org.apache.subversion.svnserve.plist
```

launchctl bietet Ihnen auch eine Möglichkeit, den Zustand von Jobs abzufragen. Falls der Job geladen ist, gibt es eine Zeile, die zum `Label` in der Jobdefinitionsdatei passt:

```
$ sudo launchctl list | grep org.apache.subversion.svnserve
-       0       org.apache.subversion.svnserve
$
```

Integrierte Authentifizierung und Autorisierung

Wenn sich ein Subversion-Client mit einem **svnserve**-Prozess verbindet, geschieht folgendes:

- Der Client wählt ein bestimmtes Projektarchiv.
- Der Server liest die zum Projektarchiv gehörende Datei `conf/svnserve.conf` und führt die darin enthaltenen Regeln für die Authentifizierung (Legitimation, Identitätsprüfung) und die Autorisierung (Berechtigungen, Befugnisse) aus.
- Je nach festgelegten Regeln und Einstellungen geht es mit einem der folgenden Punkte weiter:
  - Der Client kann seine Anfragen anonym, also ohne eine vorhergehende Authentifikationsanfrage, senden.
  - Der Client kann jederzeit eine Anmeldeaufforderung erhalten.
  - Läuft die Verbindung über einen Tunnel, so erklärt der Client, dass eine externe Anmeldung stattgefunden hat (meistens durch SSH).

Der **svnserve**-Server beherrscht als Standardeinstellung nur den CRAM-MD5-Anmeldedialog [1]. Im Kern läuft dieser wie folgt ab: Der Server sendet einen kleinen Datensatz als Anfrage an den Client. Dieser erzeugt mittels des MD5-Hash-Algorithmus einen Fingerabdruck/Hash des Passwortes zusammen mit dem Datensatz und sendet diesen Fingerabdruck als Antwort zurück an den Server. Der Server vollzieht nun dieselbe Operation mit dem Passwort und dem Datensatz und vergleicht anschließend seinen Fingerabdruck mit dem des Clients. *Während des gesamten Vorgangs wird das eigentliche Passwort nie über das Netzwerk gesendet.*

Enthält ihr **svnserve**-Server Unterstützung für SASL, so beherrscht er nicht nur die CRAM-MD5-Anmeldung, sondern noch eine Menge anderer Verfahren zur Authentifizierung. Lesen Sie „**svnserve** mit SASL verwenden“ weiter unten, um zu lernen, wie die einzelnen Möglichkeiten zur Authentifizierung und Verschlüsselung in SASL eingerichtet werden.

Es ist selbstverständlich auch möglich, dass sich der Client über ein eigenständiges Tunnel-Programm anmeldet, etwa **ssh**. In einem solchem Fall stellt der Server nur fest, unter welchem Anwenderkonto er gestartet wurde, und verwendet dieses für die weitere Anmeldung. Mehr dazu im Kapitel „Tunneln über SSH“.

Wie Sie sicher bereits bemerkt haben, ist die Datei `svnserve.conf` in jedem Projektarchiv die zentrale Anlaufstelle für alle Regeln im Rahmen der Anwenderanmeldung und Rechtevergabe. Die Datei hat dasselbe Format wie die anderen Konfigurationsdateien (siehe „Laufzeit-Konfigurationsbereich“): Die Abschnittsbezeichnungen sind von eckigen Klammern umschlossen (`[` und `]`), Kommentare werden mit Rauten (#) eingeleitet, und jeder Abschnitt enthält spezielle Variablen, denen Werte zugewiesen werden (`variable = value`). Lassen Sie uns einen Blick in diese Dateien werfen, um zu sehen wie sie verwendet werden.

Erstellen einer Passwortdatei und festlegen der Authentifikationsumgebung (Realm)

Zu Beginn enthält der Abschnitt `[general]` in der Datei `svnserve.conf` alle Einstellungen, welche für den Start notwendig sind. Lassen Sie uns anfangen und den Variablen Werte zuweisen: Wählen Sie den Namen der Datei, welche die Namen ihrer Nutzer und deren Passwörter enthält, und entscheiden Sie sich für den Namen der Authentifikationsumgebung:

```
[general]
password-db = passwortdatei
realm = Anmeldedomäne
```

Den Namen des `realm` können Sie frei wählen. Er teilt den Clients mit, an welcher „Authentifikationsumgebung“ sie sich anmelden. Der Subversion-Client zeigt diesen Namen im Anmeldedialog und verwendet ihn auch (zusammen mit dem Namen und Port des Servers) als Schlüssel, welcher als Teil des Anmeldenachweises auf der Festplatte des Nutzers gespeichert wird (siehe dazu „Zwischenspeichern von Zugangsdaten“). Die Variable `password-db` enthält den Namen der Passwortdatei, die vom Aufbau her gleich ist und die Namen der Nutzer und deren Passwörter speichert. Als Beispiel:

[1] Siehe RFC 2195.

```
[users]
harry = geheimespasswort
sally = undnocheins
```

Der Wert von `password-db` kann den absoluten oder relativen Pfad zur Anwenderdatei enthalten. In der Regel, ist es am einfachsten diese Datei ebenfalls im `conf/`-Verzeichnis des Projektarchivs zu speichern – also dort, wo auch `svnserve.conf` liegt. Andererseits möchten Sie vielleicht eine Passwortdatei für mehrere Projektarchive verwenden; in diesem Fall sollten Sie die Datei an einem zentraleren Ort ablegen. Die Projektarchive, die sich die Anwenderdatei teilen, sollten so konfiguriert sein, dass sie derselben Authentifikationsumgebung angehören, da die Anwenderliste im Wesentlichen einen Authentifikations-Bereich definiert. Wo die Datei auch liegen mag, stellen Sie sicher, die Lese- und Schreibrechte entsprechend zu setzen. Falls Sie wissen, unter welchem Konto **svnserve** laufen wird, sollten Sie den Lesezugriff zur Anwenderdatei auf das Notwendige beschränken.

Setzen von Zugriffsbeschränkungen

Es sind noch zwei weitere Variablen in der Datei `svnserve.conf` zu setzten: Sie legen fest, was nicht authentifizierten (anonymen) und authentifizierten Nutzern erlaubt ist. Die Variablen `anon-access` und `auth-access` können auf die Werte `none`, `read` oder `write` gesetzt werden. Wenn Sie den Wert auf `none` setzen, so unterbinden Sie sowohl den Lese- als auch den Schreibzugriff – `read` erlaubt den Nur-Lese-Zugriff auf das Projektarchiv und `write` gibt auf das gesamte Projektarchiv Lese- und Schreibzugriff.

```
[general]
password-db = Anwenderdatei
realm = Ihr realm

# Anonyme Anwender können nur lesend zugreifen
anon-access = read

# Authentifizierte Anwender können sowohl lesen als auch schreiben
auth-access = write
```

Tatsächlich sind die in diesem Beispiel gezeigten Einstellungen, auch die Standardwerte der Variablen, falls Sie vergessen sollten, sie zu setzten. Für den Fall, dass Sie noch zurückhaltender sein möchten, können Sie den anonymen Zugriff auch komplett unterbinden:

```
[general]
password-db = Anwenderdatei
realm = Ihr realm

# Anonyme Anwender sind nicht erlaubt
anon-access = none

# Authentifizierte Anwender können sowohl lesen als auch schreiben
auth-access = write
```

Der Serverprozess versteht nicht nur diese „pauschalen“ Zugriffseinstellungen für ein Projektarchiv, sondern auch feiner granulierte Zugriffsrechte auf einzelne Dateien und Verzeichnisse innerhalb des Projektarchive. Um diese Funktion nutzen zu können, müssen Sie eine Datei anlegen, welche die umfangreicheren Regeln enthält und anschließend die Variable `authz-db` mit folgenden Wert setzten:

```
[general]
password-db = Anwenderdatei
```

```
realm = Ihr realm

# Zum Festlegen von umfangreicheren Zugriffsregeln für bestimmte Bereiche
authz-db = Auth-Datei
```

Wir werden die Syntax der `Auth-Datei` noch später in diesem Kapitel besprechen, und zwar in „Pfadbasierte Autorisierung“. Beachten Sie, dass die `authz-db`-Variable die Verwendung der `anon-access`- und `auth-access`-Variablen nicht ausschließt – wenn alle diese Variablen gleichzeitig gesetzt sind, so müssen auch *alle* diese Regeln erfolgreich greifen, bevor ein Zugriff erlaubt wird.

svnserve mit SASL verwenden

Die meisten Teams benötigen lediglich die eingebaute CRAM-MD5 Authentifizierung von **svnserve**. Falls Ihr Server (und Ihre Subversion Clients) jedoch mit der Cyrus Simple Authentication and Security Layer (SASL) Bibliothek gebaut wurde, stehen Ihnen eine Reihe von Authentifikations- und Verschlüsselungsoptionen zur Verfügung.

Was ist SASL?

Cyrus Simple Authentication and Security Layer (einfache Cyrus Authentifikations- und Sicherheitsschicht) ist quelloffene Software, die von der Carnegie Mellon University geschrieben wurde. Sie fügt beliebigen Netzprotokollen allgemeine Authentifikations- und Verschlüsselungsfähigkeiten hinzu, und seit Subversion 1.5 kann sowohl der **svnserve**-Server als auch der **svn**-Client damit umgehen. Möglicherweise steht sie Ihnen zur Verfügung. Falls Sie Subversion selber bauen, müssen Sie mindestens Version 2.1 von SASL auf Ihrem System installiert haben und sicherstellen, dass es während des Erstellungsprozesses gefunden wird. Der Subversion Kommandozeilenclient zeigt die Verfügbarkeit von Cyrus SASL an, wenn Sie **`svn --version`** aufrufen; falls Sie irgend einen anderen Subversion-Client verwenden, sollten sie mit dem Lieferanten des Paketes Rücksprache halten, ob SASL-Unterstützung hineinkompiliert wurde.

SASL wird mit einer Anzahl ergänzbarer Module geliefert, die verschiedenartige Authentifikationssysteme repräsentieren: Kerberos (GSSAPI), NTLM, One-Time-Passwords (OTP), DIGEST-MD5, LDAP, Secure-Remote-Password (SRP) u.a. Bestimmte Mechanismen könnten für Sie verfügbar sein; überprüfen Sie, welche Module mitgeliefert werden.

Sie können Cyrus SASL (sowohl den Code als auch die Dokumentation) bei http://asg.web.cmu.edu/sasl/sasl-library.html herunterladen.

Wenn ein Subversion-Client sich mit **svnserve** verbindet, sendet der Server normalerweise eine Begrüßung, die eine Auflistung der von ihm unterstützten Fähigkeiten umfasst, woraufhin der Client mit einer ähnlichen Liste von Fähigkeiten antwortet. Falls der Server so konfiguriert wurde, dass er eine Authentifikation benötigt, sendet er eine Aufforderung, die die verfügbaren Authentifikationsmechanismen auflistet; der Client antwortet, indem er einen der Mechanismen auswählt und die Authentifizierung erfolgt dann mittels eines Nachrichtenaustausches. Selbst falls keine SASL-Fähigkeiten vorhanden sind, verstehen Client und Server von sich aus die CRAM-MD5- und ANONYMOUS-Mechanismen (siehe „Integrierte Authentifizierung und Autorisierung“). Falls Client und Server mit SASL gebaut wurden, könnten eine Anzahl weiterer Authentifikationsmechanismen verfügbar sein. Trotzdem müssen Sie serverseitig ausdrücklich SASL konfigurieren, um es anbieten zu können.

Authentifizierung mit SASL

Um bestimmte SASL-Mechanismen auf dem Server zu aktivieren, müssen Sie zwei Dinge tun. Erstellen Sie zunächst einen Abschnitt `[sasl]` in der Datei `svnserve.conf` Ihres Projektarchivs mit einem Schlüssel-Wert-Paar:

```
[sasl]
use-sasl = true
```

Erstellen Sie zweitens eine SASL-Hauptkonfigurationsdatei namens `svn.conf` dort, wo die SASL-Bibliothek sie finden kann – typischerweise in dem Verzeichnis, wo Sie müssen das Plug-in-Verzeichnis auf Ihrem System lokalisieren, etwa `/usr/lib/sasl2/` oder `/etc/sasl2/`. (Beachten Sie, dass es sich hierbei *nicht* um die Datei `svnserve.conf` handelt, die innerhalb eines Projektarchivs liegt!)

Auf einem Windows-Server müssen Sie außerdem die Systemregistratur anpassen (mit einem Werkzeug wie **regedit**), um SASL mitzuteilen, wo es Dinge finden kann. Erstellen Sie einen Registraturschlüssel namens `[HKEY_LOCAL_MACHINE\SOFTWARE\Carnegie Mellon\Project Cyrus\SASL Library]` und legen zwei weitere Schlüssel hinein: einen Schlüssel namens `SearchPath` (dessen Wert ein Pfad zum Verzeichnis bezeichnet, in dem die SASL `sasl*.dll`-Plug-in-Bibliotheken liegen) und einen Schlüssel namens `ConfFile` (dessen Wert ein Pfad zum Elternverzeichnis der von Ihnen erstellten Datei `svn.conf` ist).

Da SASL so viele unterschiedliche Arten von Authentifikationsmechanismen zur Verfügung stellt, wäre es töricht (und würde den Rahmen dieses Buches sprengen), wenn wir versuchen würden, jede mögliche Server-Konfiguration zu erläutern. Stattdessen empfehlen wir Ihnen, die Lektüre der Dokumentation aus dem Unterverzeichnis `doc/` des SASL Quelltextes. Sie beschreibt detailliert jeden Mechanismus und die entsprechende Konfiguration des Servers. Für die Erörterung an dieser Stelle zeigen wir ein einfaches Beispiel der Konfiguration des DIGEST-MD5 Mechanismus. Wenn Ihre Datei `subversion.conf` (oder `svn.conf`) beispielsweise folgenden Inhalt hat:

```
pwcheck_method: auxprop
auxprop_plugin: sasldb
sasldb_path: /etc/my_sasldb
mech_list: DIGEST-MD5
```

haben Sie SASL aufgefordert, Clients den DIGEST-MD5 Mechanismus anzubieten und Anwenderpasswörter mit einer privaten Passwort-Datenbank in `/etc/my_sasldb` abzugleichen. Ein Systemadministrator kann dann mit dem Programm **saslpasswd2** Anwendernamen und Passwörter in der Datenbank eintragen oder bearbeiten:

```
$ saslpasswd2 -c -f /etc/my_sasldb -u realm username
```

Ein paar Worte zur Warnung: Stellen Sie zunächst sicher dass das Argument „realm“ für **saslpasswd2** demselben Bereich entspricht, den Sie in der Datei `svnserve.conf` Ihres Projektarchivs definiert haben; falls diese Werte nicht übereinstimmen, wird die Authentifizierung fehlschlagen. Darüber hinaus muss aufgrund einer Unzulänglichkeit in SASL der gemeinsame Bereich aus einer Zeichenkette ohne Leerzeichen bestehen. Falls Sie sich entscheiden, die standardmäßige SASL-Passwort-Datenbank zu verwenden, sollten Sie schließlich sicherstellen, dass das Programm **svnserve** die Datei lesen (und möglicherweise auch schreiben) kann, wenn Sie einen Mechanismus wie OTP verwenden).

Dies ist lediglich eine einfache Art, SASL zu konfigurieren. Viele andere Authentifikationsmechanismen stehen zur Verfügung, und Passwörter können an anderer Stelle gespeichert werden, etwa in LDAP oder in einer SQL-Datenbank. Details hierzu finden Sie in der Dokumentation zu SASL.

Wenn Sie Ihren Server so konfigurieren, dass er nur bestimmte SASL-Authentifikationsmechanismen erlaubt, müssen Sie beachten, dass damit auch alle Clients gezwungen sind, SASL zu unterstützen. Kein Subversion-Client ohne SASL-Unterstützung (u.a. alle Clients vor Version 1.5) kann sich authentisieren. Andererseits möchten Sie vielleicht gerade diese Einschränkung („Meine Clients müssen sämtlich Kerberos verwenden!“). Wenn Sie jedoch möchten, dass sich auch Nicht-SASL-Clients authentisieren können, stellen Sie sicher, dass optional der CRAM-MD5-Mechanismus angeboten wird. Alle Clients können CRAM-MD5 verwenden, egal, ob sie SASL verstehen oder nicht.

SASL Verschlüsselung

SASL kann auch Daten verschlüsseln, sofern ein bestimmter Mechanismus das unterstützt. Der eingebaute CRAM-MD5-Mechanismus unterstützt keine Verschlüsselung, jedoch DIGEST-MD5, und Mechanismen wie SRP erfordern sogar die Verwendung der OpenSSL-Bibliothek. Um verschiedene Verschlüsselungsstufen zu aktivieren oder abzustellen, können Sie zwei Werte in der Datei `svnserve.conf` Ihres Projektarchivs einstellen:

```
[sasl]
use-sasl = true
min-encryption = 128
max-encryption = 256
```

Die Variablen `min-encryption` und `max-encryption` kontrollieren die vom Server verlangte Verschlüsselungsstufe. Um Verschlüsselung vollständig abzustellen, setzen Sie beide Werte auf 0. Um die einfache Erstellung von Prüfsummen für Daten zu ermöglichen (etwa, um Manipulationen zu verhindern und Datenintegrität ohne Verschlüsselung zu garantieren), setzen Sie beide Werte auf 1. Falls Sie Verschlüsselung erlauben, jedoch nicht voraussetzen, setzen Sie den Minimalwert auf 0 und den Maximalwert auf irgendeine Bitlänge. Um unbedingte Verschlüsselung zu verlangen,, setzen Sie beide Werte auf Zahlen größer 1. Im vorangegangenen Beispiel verlangen wir, dass Clients mindestens 128-Bit- aber höchstens 256-Bit-Verschlüsselung vornehmen.

Tunneln über SSH

Die eingebaute Authentifizierung (und die SASL-Unterstützung) von **svnserve** kann sehr praktisch sein, da es die Notwendigkeit echter Systemkonten vermeidet. Andererseits haben einige Administratoren bereits etablierte SSH-Authentifikations-Frameworks im Einsatz. In diesen Fällen haben die Anwender des Projektes bereits Systemkonten, um sich damit über SSH mit dem Server zu verbinden.

Es ist einfach, SSH in Verbindung mit **svnserve** zu verwenden. Der Client benutzt zum Verbinden einfach das `svn+ssh://` URL-Schema:

```
$ whoami
harry

$ svn list svn+ssh://host.example.com/repos/project
harryssh@host.example.com's password:  *****

foo
bar
baz
…
```

In diesem Beispiel ruft der Subversion-Client einen lokalen **ssh**-Prozess auf, der sich mit `host.example.com` verbindet, sich (gemäß der SSH-Anwenderkonfiguration) als Anwender `harryssh` authentisiert und dann auf dem entfernten Rechner einen privaten **svnserve**-Prozess unter der Anwenderkennung `harryssh` startet. Der Befehl **svnserve** wird im Tunnelmodus (`-t`) aufgerufen und dessen Netzprotokoll wird über die durch den Tunnelagenten **ssh** verschlüsselte Verbindung „getunnelt“. Falls der Client eine Übergabe macht, wird der authentifizierte Anwendername `harryssh` als Autor der neuen Revision verwendet.

An dieser Stelle ist es wichtig, zu verstehen, dass der Subversion-Client sich *nicht* mit einem laufenden **svnserve**-Dämonen verbindet. Diese Zugriffsmethode benötigt keinen Dämonen und merkt auch nicht, wenn einer vorhanden ist. Sie verlässt sich vollständig auf die Fähigkeit von **ssh**, einen temporären **svnserve**-Prozesses zu starten, der nach dem Schließen der Netzverbindung beendet wird.

Denken Sie daran, dass beim Zugriff auf ein Projektarchiv über URLs der Form `svn+ssh://` die Abfrage zur Authentifikation von **ssh** kommt und *nicht* vom **svn**-Client. Das bedeutet, dass es keine automatische Passwortspeicherung gibt (siehe „Zwischenspeichern von Zugangsdaten“). Der Subversion-Client stellt häufig mehrere Verbindungen mit dem Projektarchiv her, wenngleich Anwender das wegen der zwischengespeicherten Passwörter normalerweise gar nicht mitbekommen. Jedoch könnten Anwender bei Verwendung von `svn+ssh://`-URLs durch die wiederholten Passwortanfragen für ausgehende Verbindungen von **ssh** etwas genervt sein. Die Lösung besteht darin, ein zusätzliches Passwort-Speicherungs-Werkzeug wie etwa **ssh-agent** auf einem Unix-ähnlichen System oder **pageant** auf Windows zu verwenden.

Bei der Verwendung eines Tunnels wird die Autorisierung größtenteils durch die Betriebssystemberechtigungen auf die

Datenbankdateien des Projektarchivs gesteuert, als ob Harry direkt über einen `file://`-URL auf das Projektarchiv zugreifen würde. Falls mehrere Anwender direkt auf das Projektarchiv zugreifen sollen, möchten Sie sie vielleicht in eine gemeinsame Gruppe zusammenfassen; Sie sollten auch auf umasks achten (lesen Sie auf alle Fälle „Unterstützung mehrerer Zugriffsmethoden auf das Projektarchiv“ später in diesem Kapitel). Doch selbst beim Tunneln können Sie immer noch die Datei `svnserve.conf` zum Blockieren des Zugriffs verwenden, indem Sie einfach `auth-access = read` oder `auth-access = none` setzen.[2]

Vielleicht glauben Sie, dass die Geschichte mit dem SSH-Tunneln hier endet. Es ist aber nicht so. Subversion erlaubt es Ihnen, selbstdefinierte Verhaltensweisen für das Tunneln in Ihrer Laufzeit-Datei `config` zu definieren (siehe „Laufzeit-Konfigurationsbereich“). Nehmen wir beispielsweise an, dass Sie RSH statt SSH verwenden möchten.[3] Definieren Sie einfach im Abschnitt `[tunnels]` Ihrer Datei `config`:

```
[tunnels]
rsh = rsh
```

Ab jetzt können Sie diese neue Tunneldefinition verwenden, indem Sie ein URL-Schema benutzen, welches dem Namen Ihrer neuen Variablen entspricht: `svn+rsh://host/path`. Bei Verwendung des neuen URL-Schemas führt der Subversion-Client im Hintergrund eigentlich den Befehl **`rsh host svnserve -t`** aus. Falls Sie einen Anwendernamen im URL angeben (z.B. `svn+rsh://username@host/path`), wird der Client das auch mit in seinen Befehl übernehmen (**`rsh username@host svnserve -t`**). Sie können jedoch auch viel raffiniertere neue Tunnel-Schemata definieren:

```
[tunnels]
joessh = $JOESSH /opt/alternate/ssh -p 29934
```

Dieses Beispiel verdeutlicht einige Dinge. Erstens zeigt es, wie der Subversion-Client angewiesen wird, ein bestimmtes Tunnelprogramm (`/opt/alternate/ssh`) mit speziellen Optionen zu starten. In diesem Fall würde der Zugriff auf einen URL wie `svn+joessh://` das bestimmte SSH-Programm mit den Argumenten `-p 29934` aufrufen – was nützlich ist, wenn Sie möchten, dass sich das Tunnelprogramm mit einem Nicht-Standard-Port verbindet.

Zweitens zeigt es, wie eine eigene Umgebungsvariable definiert werden kann, die den Namen des Tunnelprogramms überschreibt. Durch das Setzen der Umgebungsvariablen `SVN_SSH` besteht eine bequeme Methode, den Standard-SSH-Tunnelagenten zu ersetzen. Falls Sie jedoch für unterschiedliche Server verschiedene Werte überschreiben müssen, wobei jeder vielleicht einen anderen Port verwendet oder unterschiedliche Optionen an SSH übergeben werden müssen, können Sie die in diesem Beispiel vorgestellte Methode verwenden. Falls wir nun die Umgebungsvariable `JOESSH` setzten, würde deren Wert den gesamten Wert der Tunnelvariablen überschreiben – statt **`/opt/alternate/ssh -p 29934`** würde **`$JOESSH`** ausgeführt.

SSH-Konfigurationstricks

Es ist möglich, nicht nur die Art und Weise zu steuern, auf die der Client **ssh** aufruft, sondern auch das Verhalten von **sshd** auf dem Server. In diesem Abschnitt zeigen wir, wie der von **sshd** ausgeführte **svnserve**-Befehl genau festgelegt wird sowie sich mehrere Anwender ein einzelnes Systemkonto teilen können.

Erstmalige Einrichtung

Ermitteln Sie zunächst das Heimatverzeichnis des Kontos, welches Sie zum Starten von **svnserve** verwenden möchten. Stellen Sie sicher, dass für das Konto ein Paar bestehend aus einem öffentlichen und einem privaten SSH-Schlüssel installiert ist und dass sich der Anwender über die Authentifikation mit einem öffentlichen Schlüssel anmelden kann. Die Authentifikation mit Passwort wird nicht funktionieren, da sich alle folgenden SSH-Tricks um die Verwendung der SSH-Datei `authorized_keys` drehen.

[2] Beachten Sie, dass die Verwendung irgendwelcher durch **svnserve** sichergestellten Zugriffskontrollen sinnlos ist, da der Anwender ohnehin direkten Zugriff auf die Projektarchiv-Datenbank hat.
[3] Wir empfehlen das natürlich nicht, da RSH deutlich unsicherer ist als SSH.

Falls noch nicht geschehen, erzeugen Sie die Datei `authorized_keys` (unter Unix üblicherweise `~/.ssh/authorized_keys`). Jede Zeile dieser Datei beschreibt einen öffentlichen Schlüssel, der sich verbinden darf. Die Zeilen sehen normalerweise so aus:

```
ssh-dsa AAAABtce9euch… user@example.com
```

Das erste Feld beschreibt die Art des Schlüssels, das zweite ist der eigentliche in Base64 codierte Schlüssel und das dritte Feld ist ein Kommentar. Es ist jedoch weniger bekannt, dass die gesamte Zeile mit einem `command`-Feld beginnen kann:

```
command="program" ssh-dsa AAAABtce9euch… user@example.com
```

Falls das Feld `command` definiert ist, wird der SSH-Daemon das angegebene Programm starten anstatt wie üblich den vom Subversion-Client geforderten Aufruf von **svnserve** im Tunnelmodus auszuführen. Das öffnet die Tür für eine Reihe von serverseitigen Tricks. In den folgenden Beispielen werden wir die Zeilen der Datei wie folgt abkürzen:

```
command="program" TYPE KEY COMMENT
```

Steuerung des aufgerufenen Befehls

Da wir den durch den Server ausgeführten Befehl angeben können, ist es leicht, ein bestimmtes **svnserve**-Programm zu benennen und ihm zusätzliche Argumente mitzugeben:

```
command="/path/to/svnserve -t -r /virtual/root" TYPE KEY COMMENT
```

In diesem Beispiel könnte es sich bei `/path/to/svnserve` um ein maßgeschneidertes Wrapper-Skript für **svnserve** handeln, das die umask setzt (siehe „Unterstützung mehrerer Zugriffsmethoden auf das Projektarchiv“). Es wird außerdem veranschaulicht, wie **svnserve** in einem virtuellen Wurzelverzeichnis verankert wird, so wie es oft gemacht wird, wenn **svnserve** als Daemon-Prozess läuft. Das könnte entweder geschehen, um den Zugriff auf Teile des Systems einzuschränken oder um dem Anwender die Bürde abzunehmen, einen absoluten Pfad im URL `svn+ssh://` angeben zu müssen.

Es besteht weiterhin die Möglichkeit, dass sich mehrere Anwender ein einzelnes Konto teilen. Statt ein gesondertes Konto für jeden Anwender zu erstellen, generieren Sie für jede Person jeweils ein Paar aus einem öffentlichen und einem privaten Schlüssel. Tragen Sie dann jeden öffentlichen Schlüssel in eine eigene Zeile der Datei `authorized_keys` ein und verwenden die Option `--tunnel-user`:

```
command="svnserve -t --tunnel-user=harry" TYPE1 KEY1 harry@example.com
command="svnserve -t --tunnel-user=sally" TYPE2 KEY2 sally@example.com
```

Dieses Beispiel erlaubt sowohl Harry als auch Sally, sich über die Authentifikation durch einen öffentlichen Schlüssel mit demselben Konto zu verbinden. Beide verfügen über einen angepassten Befehl, der ausgeführt wird. Die Option `--tunnel-user` teilt **svnserve** mit, das benannte Argument als den authentifizierten Anwender zu akzeptieren. Ohne `--tunnel-user` sähe es so aus als kämen alle Übergaben vom gemeinsamen Konto.

Ein letztes Wort zur Warnung: Die Zugangsberechtigung für einen Anwender über einen öffentlichen Schlüssel und ein gemeinsames Konto könnte noch weitere SSH-Zugänge erlauben, selbst wenn Sie einen Wert für `command` in in

`authorized_keys` angegeben haben. So kann der Anwender beispielsweise Shell-Zugang über SSH erlangen oder X11 oder allgemeine Portweiterleitung über Ihren Server einrichten. Um Ihrem Anwender die geringstmögliche Erlaubnis zu erteilen, sollten Sie unmittelbar nach `command` eine Reihe einschränkender Optionen angeben:

```
  command="svnserve -t --tunnel-user=harry",no-port-forwarding,no-agent-forw
arding,no-X11-forwarding,no-pty TYPE1 KEY1 harry@example.com
```

Beachten Sie, dass alles auf einer Zeile stehen muss – tatsächlich auf einer Zeile – da die SSH-Dateien `authorized_keys` nicht einmal das übliche Backslash-Zeichen (\) zur Zeilenfortsetzung erlauben. Der einzige Grund für den von uns eingefügten Zeilenumbruch ist die Unterbringung auf der Breite einer Buchseite.

httpd, der Apache HTTP-Server

Der Apache HTTP-Server ist ein „Hochleistungs“-Netzwerk-Server, den Subversion zu seinem Vorteil nutzen kann. Über ein angepasstes Modul macht **httpd** Subversion-Projektarchive für Clients über das WebDAV/DeltaV-Protokoll[4] verfügbar, welches eine Erweiterung von HTTP 1.1 ist. Dieses Protokoll nimmt das allgegenwärtige HTTP-Protokoll, das der Kern des World Wide Web ist, und fügt Schreibfähigkeiten – im Besonderen, versioniertes Schreiben – hinzu. Das Ergebnis ist ein standardisiertes, robustes System, das auf geeignete Weise als Teil der Software Apache 2.0 verteilt wird, die von zahlreichen Betriebssystemen und Drittanbieter-Produkten unterstützt wird und keine Netzwerk-Administratoren benötigt, um einen weiteren speziellen Port zu öffnen.[5] Während ein Apache-Subversion-Server mehr Möglichkeiten bietet als **svnserve**, ist er allerdings auch etwas schwieriger einzurichten. Flexibilität geht oft mit Komplexität einher.

Viele der folgenden Erläuterungen beziehen sich auf Konfigurationsdirektiven von Apache. Obwohl ein paar Beispiele zur Verwendung dieser Direktiven gegeben werden, würde deren erschöpfende Behandlung dieses Kapitel sprengen. Das Apache-Team verfügt über hervorragende Dokumentation, die auf deren Web-Seite http://httpd.apache.org frei verfügbar ist. So befindet sich beispielsweise eine allgemeine Referenz der Konfigurationsdirektiven unter http://httpd.apache.org/docs-2.0/mod/directives.html.

Falls Sie Änderungen an den Einstellungen von Apache vornehmen, ist es wahrscheinlich, dass sich irgendwo ein Fehler einschleicht. Wenn Sie noch nicht mit der Protokollierung von Apache vertraut sind, sollten sie sich damit vertraut machen. In der Datei `httpd.conf` befinden sich Direktiven, die angeben, wo auf der Platte sich die von Apache erzeugten Zugriffs- und Fehlerprotokollierungsdateien befinden (die Direktiven `CustomLog` bzw. `ErrorLog`). Auch **mod_dav_svn** von Subversion verwendet die Protokollierungsschnittstelle von Apache. Sie können jederzeit den Inhalt dieser Dateien nach Informationen durchforsten, die eine Problemquelle aufdecken könnten, die sonst nicht offensichtlich wäre.

Voraussetzungen

Um Ihr Projektarchiv im Netz über HTTP zur Verfügung zu stellen, brauchen Sie grundsätzlich vier Komponenten, die in zwei Paketen verfügbar sind. Sie benötigen Apache **httpd** 2.0 oder neuer, das dazu gelieferte DAV-Modul **mod_dav**, Subversion und das mitgelieferte Dateisystemmodul **mod_dav_svn**. Sobald Sie über all diese Komponenten verfügen, ist die Bereitstellung Ihres Projektarchivs über das Netz ganz einfach:

- Inbetriebnahme von httpd mit dem Modul **mod_dav**
- Installation des **mod_dav_svn**-Backends zu **mod_dav**, das die Bibliotheken von Subversion für den Zugriff auf das Projektarchiv verwendet
- Konfiguration Ihrer Datei `httpd.conf`, um das Projektarchiv zu exportieren (oder sichtbar zu machen)

Sie können die ersten beiden Punkte bewerkstelligen, indem Sie entweder **httpd** und Subversion aus den Quellen übersetzen oder als vorgefertigte Binärpakete auf Ihrem System installieren. Die aktuellsten Informationen zur Übersetzung von Subversion in Verbindung mit dem Apache HTTP-Server sowie die Übersetzung und Konfigurierung von Apache zu diesem Zweck finden Sie in der Datei `INSTALL` in der obersten Verzeichisebene des Quelltextes von Subversion.

[4] Siehe http://www.webdav.org/.
[5] Die hassen so etwas echt.

Grundlegende Konfiguration von Apache

Sobald alle notwendigen Komponenten auf Ihrem System installiert sind, bleibt nur noch die Konfiguration von Apache über seine Datei `httpd.conf`. Weisen Sie Apache mit der Direktive `LoadModule` an, das Modul **mod_dav_svn** zu laden. Diese Direktive muss vor allen Konfigurationseinträgen in Verbindung mit Subversion stehen. Falls Ihr Apache mit dem vorgegebenen Aufbau installiert wurde, sollte sich das Modul **mod_dav_svn** im Unterverzeichnis `modules` des Apache-Installationsverzeichnisses befinden (oft `/usr/local/apache2`). Die Direktive `LoadModule` hat eine einfache Syntax, wobei ein benanntes Modul auf den Ort einer Shared-Library auf der Platte abgebildet wird:

```
LoadModule dav_svn_module     modules/mod_dav_svn.so
```

Apache interpretiert den Bibliothekspfad des Konfigurationseintrags `LoadModule` als relativ zum Wurzelverzeichnis seines eigenen Servers. Falls er wie gezeigt konfiguriert ist, wird Apache in seinem eigenen `modules/` Unterverzeichnis nach der Shared-Library des Subversion DAV-Moduls suchen. Abhängig davon, wie Subversion auf Ihrem System installiert wurde, kann es sein, dass Sie einen ganz anderen Pfad für diese Bibliothek angeben müssen, vielleicht sogar einen absoluten Pfad wie im folgenden Beispiel:

```
LoadModule dav_svn_module     C:/Subversion/lib/mod_dav_svn.so
```

Falls **mod_dav** als Shared-Objekt übersetzt wurde (statt statisch direkt in die **httpd**-Binärdatei gelinkt worden zu sein), benötigen Sie hierfür ebenfalls einen ähnlichen `LoadModule`-Eintrag. Stellen Sie sicher, dass er vor der **mod_dav_svn**-Zeile steht:

```
LoadModule dav_module         modules/mod_dav.so
LoadModule dav_svn_module     modules/mod_dav_svn.so
```

Weiter unten in der Konfigurationsdatei sollten Sie Apache nun mitteilen, wo Sie Ihr Subversion-Projektarchiv (oder Ihre Projektarchive) aufbewahren. Die Direktive `Location` besitzt eine XML-ähnliche Notation, beginnend mit einem öffnenden Tag, endend mit einem schließenden Tag und verschiedener anderer Konfigurationsdirektiven dazwischen. Der Zweck der Direktive `Location` besteht darin, Apache anzuweisen, etwas Besonderes zu tun, falls Anfragen bearbeitet werden, die an einen bestimmten URL oder dessen Kinder gerichtet sind. Im Fall von Subversion möchten Sie, dass Apache die Unterstützung für URLs, die auf versionierte Ressourcen zeigen, einfach an die DAV-Schicht weiterleitet. Sie können Apache anweisen, die Bearbeitung aller URLs, deren Pfadteile (der Teil des URL, der nach dem Servernamen und der optionalen Portnummer steht) mit `/repos/` beginnen, an einen DAV-Provider zu delegieren, dessen Projektarchiv unter `/var/svn/repository` liegt, indem Sie die folgende `httpd.conf`-Syntax verwenden:

```
<Location /repos>
  DAV svn
  SVNPath /var/svn/repository
</Location>
```

Falls Sie planen, mehrere Subversion-Projektarchive zu unterstützen, die sich unterhalb eines gemeinsamen Elternverzeichnisses auf Ihrer lokalen Platte befinden, können Sie eine alternative Direktive, `SVNParentPath`, verwenden, um auf das gemeinsame Elternverzeichnis hinzuweisen. Wenn Sie beispielsweise wissen, dass Sie mehrere Subversion-Projektarchive in einem Verzeichnis `/var/svn` anlegen möchten, auf die über URLs wie `http://my.server.com/svn/repos1`, `http://my.server.com/svn/repos2` usw. zugegriffen werden soll, könnten Sie die Konfigurationssyntax von `httpd.conf` aus dem folgenden Beispiel verwenden:

```
<Location /svn>
  DAV svn

  # Automatisch irgendein "/svn/foo" URL auf Projektarchiv /var/svn/foo abbilden
  SVNParentPath /var/svn
</Location>
```

Die Verwendung dieser Syntax veranlasst Apache, die Bearbeitung aller URLs, deren Pfadteil mit `/svn/` beginnt, an den Subversion-DAV-Provider weiterzuleiten, der dann davon ausgeht, dass alle Objekte in dem durch die `SVNParentPath`-Direktive spezifizierten Verzeichnis tatsächlich Subversion-Projektarchive sind. Dies ist insofern eine besonders bequeme Syntax, da Sie im Gegensatz zur Direktive `SVNPath` Apache nicht neu starten müssen, wenn Sie Projektarchive hinzufügen oder entfernen.

Achten Sie beim Definieren Ihrer neuen `Location` darauf, dass sie sich nicht mit anderen bereits exportierten überschneidet. Wenn beispielsweise Ihre Haupt-`DocumentRoot` nach `/www` exportiert wird, sollten Sie ein Subversion-Projektarchiv nicht in `<Location /www/repos>` exportieren. Falls eine Anfrage für den URI `/www/repos/foo.c` hereinkommt, weiß Apache nicht, ob die Datei `repos/foo.c` in `DocumentRoot` gesucht wird oder ob es die Herausgabe von `foo.c` aus dem Projektarchiv an **mod_dav_svn** delegieren soll. Das Ergebnis ist oftmals ein Fehler vom Server der Form `301 Moved Permanently`.

Server-Namen und die COPY-Anfrage

Subversion verwendet den `COPY`-Anfragetyp, um serverseitige Kopien von Dateien und Verzeichnissen anzufertigen. Als Teil der von den Apache-Modulen durchgeführten Plausibilitätsprüfung wird erwartet, dass sich die Quelle der Kopie auf demselben Rechner befindet wie das Ziel. Um diese Anforderung zu erfüllen, kann es notwendig sein, dass Sie **mod_dav** den von Ihnen verwendeten Namen für den Server mitteilen müssen. Im Allgemeinen können Sie hierfür die Direktive `ServerName` in `httpd.conf` verwenden.

```
ServerName svn.example.com
```

Falls Sie Apaches Unterstützung für virtuelles Hosting über die Direktive `NameVirtualHost` nutzen, müssen Sie eventuell die Direktive `ServerAlias` verwenden, um weitere Namen anzugeben, unter denen Ihr Server bekannt ist. Wenden Sie sich auch hier für alle Details an die Dokumentation von Apache.

An dieser Stelle sollten Sie ernsthaft über Berechtigungen nachdenken. Falls Sie Apache schon eine Zeit lang als Ihren regulären Web-Server in Betrieb haben, werden Sie wahrscheinlich bereits eine Ansammlung von Inhalt haben, etwa Webseiten, Skripte usw. Diese Dinge sind bereits mit einer Menge an Berechtigungen versehen, die es ihnen erlauben, mit Apache zusammen zu arbeiten, oder passender, die es Apache erlauben, mit diesen Dateien zu arbeiten. Wenn Apache als Subversion-Server eingesetzt wird, braucht er ebenfalls die richtigen Berechtigungen zum Lesen und Schreiben Ihres Subversion-Projektarchivs.

Sie werden ein Berechtigungssystem festlegen müssen, das die Anforderungen von Subversion erfüllt, ohne dabei bestehende Webseiten oder Skriptinstallationen zu beeinträchtigen. Das kann bedeuten, dass die Berechtigungen für Ihr Projektarchiv an die anderen Dinge angepasst werden müssen, die Apache für Sie zur Verfügung stellt, oder dass Sie die Direktiven `User` und `Group` in `httpd.conf` verwenden, um Apache mit denjenigen Anwender- und Gruppenkennungen laufen zu lassen, die auch das Subversion-Projektarchiv besitzt. Es gibt keine einzig richtige Methode, um die Berechtigungen zu vergeben, und jeder Administrator wird bestimmte Gründe haben, um es auf eine bestimmte Art zu tun. Seien Sie sich lediglich bewusst, dass Probleme im Zusammenhang mit den Berechtigungen am häufigsten übersehen werden, wenn ein Subversion-Projektarchiv für die Verwendung mit Apache eingerichtet wird.

Authentifikationsoptionen

Falls Sie `httpd.conf` dergestalt konfiguriert haben, so dass sie etwa den folgenden Eintrag enthält:

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
</Location>
```

kann die Welt „anonym“ auf Ihr Projektarchiv zugreifen. Bis Sie Authentifikations- und Autorisierungsrichtlinien konfiguriert haben, sind die über die Direktive `Location` zur Verfügung gestellten Projektarchive allgemein für jedermann zugreifbar. Mit anderen Worten:

- Jeder kann mit einem Subversion-Client eine Arbeitskopie eines Projektarchiv-URLs (oder irgendeines der Unterverzeichnisse) auschecken.
- Jeder kann interaktiv die letzte Revision des Projektarchivs durchstöbern, indem der Projektarchiv-URL einfach mit einem Web-Browser geöffnet wird.
- Jeder kann an das Projektarchiv übergeben.

Natürlich kann es sein, dass Sie schon längst ein `pre-commit` Hook-Skript bereitgestellt haben, um Übergaben zu verhindern (siehe „Erstellen von Projektarchiv-Hooks“). Sie werden jedoch beim Weiterlesen feststellen, dass es möglich ist, die eingebauten Methoden von Apache zu verwenden, um den Zugriff auf bestimmte Art und Weise einzuschränken.

Die erforderliche Authentifizierung verhindert zwar, dass unerlaubte Anwender direkt auf das Projektarchiv zugreifen, schützt aber nicht die Vertraulichkeit der Netzwerkaktivitäten erlaubter Anwender. Siehe „Schutz des Netzwerkverkehrs durch SSL“ zur Konfiguration Ihres Servers mit SSL-Verschlüsselung, die eine zusätzliche Sicherheitsschicht bietet.

Einfache Authentifizierung

Die einfachste Methode, einen Client zu authentifizieren geht über den HTTP-Basic-Authentifikationsmechanismus, der einfach einen Anwendernamen und ein Passwort verwendet, um die Identität eines Anwenders sicherzustellen. Apache stellt das Dienstprogramm **htpasswd**[6] zur Verfügung, welches die Verwaltung von Dateien übernimmt, die Anwendernamen und Passwörter beinhalten.

Die einfache Authentifizierung ist *extrem* unsicher, da Passwörter fast im Klartext über das Netz geschickt werden. Siehe „Digest authentication“ für Details zur Verwendung des wesentlich sichereren Digest Mechanismus.

Legen Sie zunächst eine Passwort-Datei und erlauben Sie den Zugriff für die Anwender Harry und Sally:

```
$ ### Beim 1. Mal: -c verwenden, um die Datei anzulegen
$ ### -m für die sicherere MD5-Verschlüsselung des Passworts verwenden
$ htpasswd -c -m /etc/svn-auth.htpasswd harry
New password: *****
Re-type new password: *****
Adding password for user harry
$ htpasswd -m /etc/svn-auth.htpasswd sally
New password: *******
Re-type new password: *******
Adding password for user sally
$
```

[6] Siehe http://httpd.apache.org/docs/current/programs/htpasswd.html.

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  # Authentication: Basic
  AuthName "Subversion repository"
  AuthType Basic
  AuthUserFile /etc/svn-auth.htpasswd
</Location>
```

Diese Direktiven funktionieren wie folgt:

- `AuthName` ist ein beliebiger Name, den Sie für Ihre Authentifikationsdomäne wählen. Die meisten Browser zeigen diesen Namen im Dialog an, wenn der Browser den Anwender nach seinem Namen und dem Passwort fragt.
- `AuthType` spezifiziert den Typ der zu verwendenden Authentifikation.
- `AuthUserFile` spezifiziert den Ort der zu verwendenden Passwortdatei.

Allerdings bewirkt dieser `<Location>`-Block noch nichts sinnvolles. Er teilt Apache lediglich mit, dass es sich den Anwendernamen und das Passwort vom Subversion-Client besorgen soll, *falls* eine Autorisierung benötigt wird. (Wenn eine Autorisierung erforderlich ist, benötigt Apache auch eine Authentifikation.) Was hier jedoch noch fehlt, sind Direktiven, die Apache sagen, *welche Arten* von Client-Anfragen eine Autorisierung erfordern; momentan sind das keine. Das Einfachste ist es, anzugeben, dass *alle* Anfragen eine Autorisierung erfordern, indem dem Block die Anweisung `Require valid-user` hinzugefügt wird:

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn

  # Authentication: Basic
  AuthName "Subversion repository"
  AuthType Basic
  AuthUserFile /etc/svn-auth.htpasswd

  # Authorization: Authenticated users only
  Require valid-user
</Location>
```

Zu Details über die Direktive `Require` und anderen Möglichkeiten, Autorisierungsrichtlinien festzulegen, sehen Sie unter „Autorisierungsoptionen“ nach.

Digest authentication

Digest-Authentifizierung ist eine Verbesserung der Basic-Authentifizierung, die es dem Server ermöglicht, die Identität des Clients zu bestätigen, ohne das Passwort ungeschützt durch das Netz zu schicken. Sowohl Client als auch Server erzeugen einen nicht rückgängig zu machenden MD5-Hashwert des Anwendernamens, Passworts, verlangter URI und einer Einwegnummer, die vom Server vergeben wird und jedes Mal geändert wird, wenn eine Authentifizierung benötigt wird. Der Client sendet seinen Hash an den Server und der Server verifiziert dann, dass die Hashes zusammenpassen.

Die Konfigurierung von Apache für die Digest-Authentifizierung ist unkompliziert und nur eine kleine Abweichung von unserem vorangegangenen Beispiel:

```
<Location /svn>
  DAV svn
```

```
  SVNParentPath /var/svn

  # Authentication: Digest
  AuthName "Subversion repository"
  AuthType Digest
  AuthUserFile /etc/svn-auth.htdigest

  # Authorization: Authenticated users only
  Require valid-user
</Location>
```

Beachten Sie, dass `AuthType` nun auf `Digest` gesetzt ist, und wir einen unterschiedlichen Pfad für `AuthUserFile` angegeben haben. Digest-Authentifizierung verwendet ein unterschiedliches Dateiformat als Basic-Authentifizierung; es wird mit Apaches Dienstprogramm **htdigest** erzeugt[7] statt mit **htpasswd**. Digest-Authentifizierung besitzt auch das zusätzliche Konzept eines Bereichs, „realm“, der dem Wert der Direktive `AuthName` entsprechen muss. Die Passwortdatei kann wie folgt erzeugt werden:

```
$ ### Beim ersten Mal: -c zum Erzeugen der Datei verwenden
$ htdigest -c /etc/svn-auth.htdigest "Subversion repository" harry
Adding password for harry in realm Subversion repository.
New password: *****
Re-type new password: *****
$ htdigest /etc/svn-auth.htdigest "Subversion repository" sally
Adding user sally in realm Subversion repository
New password: *******
Re-type new password: *******
$
```

Autorisierungsoptionen

An diesem Punkt haben Sie die Authentifizierung eingerichtet, nicht jedoch die Autorisierung. Apache kann Clients auffordern und Identitäten bestätigen, aber es wurde ihm noch nicht gesagt, wie er den Zugriff von Clients mit diesen Identitäten erlauben oder einschränken soll. Dieser Abschnitt beschreibt zwei Strategien, um den Zugriff auf Ihre Projektarchive zu kontrollieren.

Pauschale Zugriffskontrolle

Die einfachste Form der Zugriffskontrolle besteht darin, bestimmten Nutzern entweder nur Lesezugriff oder Lese- und Schreibzugriff auf ein Projektarchiv zu gewähren.

Sie können den Zugriff auf alle Operationen im Projektarchiv einschränken, indem Sie `Require valid-user` direkt dem `<Location>`-Block hinzufügen. Das Beispiel von „Digest authentication“ erlaubt nur Clients, die sich erfolgreich authentifiziert haben, um irgendetwas mit dem Subversion-Projektarchiv zu machen:

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn

  # Authentifizierung: Digest
  AuthName "Subversion repository"
  AuthType Digest
  AuthUserFile /etc/svn-auth.htdigest

  # Autorisierung: Nur für authentifizierte Anwender
  Require valid-user
</Location>
```

[7] Siehe http://httpd.apache.org/docs/current/programs/htdigest.html.

Manchmal müssen Sie gar nicht so ein strenges Regiment führen. So erlaubt beispielsweise das eigene Projektarchiv von Subversion unter http://svn.collab.net/repos/svn allen auf der Welt lesende Operationen (wie etwa das Auschecken von Arbeitskopien und das Stöbern im Projektarchiv), beschränkt jedoch Schreiboperationen auf authentifizierte Nutzer. Die Direktiven `Limit` und `LimitExcept` erlauben diese Art der selektiven Einschränkung. Ähnlich der Direktive `Location` haben diese Blöcke Start- und Ende-Tags, die Sie innerhalb Ihres `<Location>`-Blocks unterbringen.

Die für die Direktiven `Limit` und `LimitExcept` verfügbaren Parameter sind HTTP-Anfrage-Typen, die von diesem Block beeinflusst werden. Falls Sie beispielsweise anonyme Nur-Lese-Zugriffe erlauben wollen, so würden Sie die Direktive `LimitExcept` (mit den Anfrage-Parametern `GET`, `PROPFIND`, `OPTIONS` und `REPORT`) verwenden und die vorher erwähnte Direktive `Require valid-user` im Block `<LimitExcept>` statt nur innerhalb des `<Location>`-Blocks einfügen.

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn

  # Authentifizierung: Digest
  AuthName "Subversion repository"
  AuthType Digest
  AuthUserFile /etc/svn-auth.htdigest

  # Autorisierung: Nur authentifizierte Anwender für nicht Nur-Lese
  #                (Schreib-) Operationen; anonymes Lesen zulassen
  <LimitExcept GET PROPFIND OPTIONS REPORT>
    Require valid-user
  </LimitExcept>
</Location>
```

Dies sind nur ein paar einfache Beispiele. Für tiefer gehende Informationen über Apaches Zugriffskontrolle und die Direktive `Require` sollten Sie im Abschnitt `Security` der Apache Lehrbuchsammlung unter http://httpd.apache.org/docs-2.0/misc/tutorials.html nachlesen.

Verzeichnisweise Zugangskontrolle

Es ist möglich, detailliertere Zugriffsrechte mithilfe von **mod_authz_svn** einzurichten. Dieses Apache-Modul schnappt sich die verschiedenen undurchsichtigen URLs, die vom Client zum Server gereicht werden, fordert **mod_dav_svn** auf, sie zu dekodieren, und unterbindet dann möglicherweise Anforderungen entsprechend in einer Konfigurationsdatei definierter Zugriffsregeln.

Falls Sie Subversion aus Quellcode gebaut haben, ist **mod_authz_svn** automatisch neben **mod_dav_svn** gebaut und installiert worden. Viele binäre Distributionen installieren es ebenfalls automatisch. Um die korrekte Installation zu überprüfen, müssen Sie sicherstellen, dass es direkt hinter der `LoadModule`-Direktive von **mod_dav_svn** in `httpd.conf` auftaucht:

```
LoadModule dav_module         modules/mod_dav.so
LoadModule dav_svn_module     modules/mod_dav_svn.so
LoadModule authz_svn_module   modules/mod_authz_svn.so
```

Zur Aktivierung dieses Moduls müssen Sie Ihren `<Location>`-Block mit der Direktive `AuthzSVNAccessFile` konfigurieren, die eine Datei mit Zugriffsrichtlinien für Pfade in Ihren Projektarchiven bezeichnet. (Gleich werden wir auf das Format dieser Datei eingehen.)

Da Apache flexibel ist, haben Sie die Wahl, Ihren Block auf eine von drei Arten zu konfigurieren. Fangen Sie mit der Auswahl eines dieser grundlegenden Konfigurationsmuster an. (Die folgenden Beispiele sind sehr einfach gehalten; sehen Sie sich die

mitgelieferte Apache-Dokumentation an, um wesentlich mehr Einzelheiten zu den Authentifikations- und Autorisierungsoptionen von Apache zu erfahren.)

Der offenste Ansatz besteht aus Zugang für jeden. Das bedeutet, dass Apache niemals Aufforderungen zur Authentifikation sendet, so dass alle Anwender als „anonymous“ behandelt werden. (Siehe Beispiel 6.2, „Eine Beispielkonfiguration für anonymen Zugang“.)

Beispiel 6.2. Eine Beispielkonfiguration für anonymen Zugang

```
<Location /repos>
  DAV svn
  SVNParentPath /var/svn

  # Authentifizierung: keine

  # Autorisierung: pfadbasierte Zugangskontrolle
  AuthzSVNAccessFile /path/to/access/file
</Location>
```

Am anderen Ende der Paranoia-Skala können Sie Apache dergestalt konfigurieren, dass er alle Clients authentifiziert. Dieser Block verlangt eine unbedingte Authentifikation durch die Direktive `Require valid-user` und definiert wie berechtigte Anwender authentifiziert werden sollen. (Siehe Beispiel 6.3, „Eine Beispielkonfiguration für authentifizierten Zugang“.)

Beispiel 6.3. Eine Beispielkonfiguration für authentifizierten Zugang

```
<Location /repos>
  DAV svn
  SVNParentPath /var/svn

  # Authentifizierung: Digest
  AuthName "Subversion repository"
  AuthType Digest
  AuthUserFile /etc/svn-auth.htdigest

  # Autorisierung: pfadbasierte Zugangskontrolle; nur für authentifizierte Anwender
  AuthzSVNAccessFile /path/to/access/file
  Require valid-user
</Location>
```

Ein drittes sehr verbreitetes Muster ist es, eine Kombination aus authentifizierten und anonymen Zugriff zu erlauben. So möchten beispielsweise viele Administratoren für anonyme Anwender den Lesezugriff auf bestimmte Verzeichnisse des Projektarchivs freigeben, während für heikle Bereichen nur authentifizierte Anwender zugelassen werden. Bei dieser Einstellung greifen alle Anwender zunächst anonym auf das Projektarchiv zu. Falls Ihre Zugangsrichtlinien an einer Stelle einen echten Anwendernamen erfordern sollte, fordert Apache den Client auf, sich zu authentisieren. Eingestellt wird dieses Verhalten mit den Direktiven `Satisfy Any` sowie `Require valid-user`. (Siehe Beispiel 6.4, „Eine Beispielkonfiguration für gemischten authentifizierten/anonymen Zugang“.)

Beispiel 6.4. Eine Beispielkonfiguration für gemischten authentifizierten/anonymen Zugang

```
<Location /repos>
  DAV svn
  SVNParentPath /var/svn
```

```
  # Authentifizierung: Digest
  AuthName "Subversion repository"
  AuthType Digest
  AuthUserFile /etc/svn-auth.htdigest

  # Autorisierung: pfadbasierte Zugangskontrolle; zunächst anonymen Zugang
  #                versuchen, doch wenn nötig authentifizieren
  AuthzSVNAccessFile /path/to/access/file
  Satisfy Any
  Require valid-user
</Location>
```

Der nächste Schritt ist, eine Autorisierungsdatei zu erstellen, die Zugangsregeln für bestimmte Pfade innerhalb des Projektarchivs enthält. Wie, beschreiben wir später in „Pfadbasierte Autorisierung“.

Abstellen pfadbasierter Prüfungen

Das Modul **mod_dav_svn** unternimmt einen hohen Arbeitsaufwand, um sicherzustellen, dass Daten, die Sie als „nicht lesbar“ markiert haben, nicht versehentlich nach draußen geraten. Das bedeutet, dass es aufmerksam alle Pfade überwachen muss, die von Befehlen wie **svn checkout** und **svn update** zurückgegeben werden. Begegnen diese Befehle einem Pfad, der aufgrund einer Autorisierungsrichtlinie nicht lesbar ist, wird dieser Pfad üblicherweise vollständig unterdrückt. Im Fall der Historien- oder Umbenennungsverfolgung, z.B. mit einem Befehl wie **`svn cat -r OLD foo.c`** auf einer Datei, die vor langer Zeit umbenannt wurde, bleibt die Umbenennungsverfolgung einfach stehen, wenn einer der früheren Namen des Objektes als lesebeschränkt erkannt wird.

All diese Pfadüberprüfungen können manchmal sehr teuer werden, besonders mit **svn log**. Wenn eine Liste mit Revisionen erstellt wird, sieht der Server bei jedem geänderten Pfad in jeder Revision nach, ob er lesbar ist. Falls ein nicht lesbarer Pfad entdeckt wird, taucht er in der Liste der geänderten Pfade dieser Revision nicht auf (normalerweise sichtbar mit der Option `--verbose` (`-v`)), und die gesamte Protokollnachricht wird unterdrückt. Es bedarf wohl keiner Erwähnung, dass dies bei Revisionen, die eine große Anzahl an Pfaden betreffen, sehr zeitaufwändig sein kann. Das ist der Preis für Sicherheit: selbst wenn Sie überhaupt kein Modul wie **mod_authz_svn** konfiguriert haben, fordert das Modul **mod_dav_svn** Apache **httpd** auf, Autorisierungsüberprüfungen für jeden Pfad vorzunehmen. Das Modul **mod_dav_svn** weiß nicht, welche Autorisierungsmodule installiert wurden, also kann es lediglich Apache auffordern, all das aufzurufen, was vorhanden sein könnte.

Auf der anderen Seite gibt es auch eine Art Notausgang, der es Ihnen erlaubt, Sicherheitsmerkmale gegen Geschwindigkeit zu tauschen. Falls Sie nicht irgendeine Art verzeichnisbasierter Autorisierung durchsetzen möchten (d.h., **mod_authz_svn** oder ähnliche Module nicht verwenden), können Sie die gesamte Pfadüberprüfung abstellen. Verwenden Sie die Direktive `SVNPathAuthz` in Ihrer Datei `httpd.conf` wie in Beispiel 6.5, „Abstellen aller Pfadüberprüfungen“ gezeigt.

Beispiel 6.5. Abstellen aller Pfadüberprüfungen

```
<Location /repos>
  DAV svn
  SVNParentPath /var/svn

  SVNPathAuthz off
</Location>
```

Standardmäßig steht die Direktive `SVNPathAuthz` auf „on“. Auf „off“ gesetzt, wird die gesamte pfadbasierte Autorisierungsüberprüfung abgestellt. **mod_dav_svn** beendet den Aufruf von Autorisierungsüberprüfungen für jeden entdeckten Pfad.

Schutz des Netzwerkverkehrs durch SSL

Die Verbindung zu einem Projektarchiv über `http://` bedeutet, dass alle Subversion-Aktivitäten im Klartext über das Netzwerk geschickt werden. Dass heißt, dass Operationen wie das Auschecken, Übergaben und Aktualisierungen potentiell

durch Unbefugte, die den Netzwerkverkehr „abschnorcheln“, abgefangen werden können. Die Verschlüsselung des Verkehrs mit SSL ist eine gute Maßnahme, um möglicherweise heikle Informationen im Netzwerk zu schützen.

Wird ein Subversion-Client für die Verwendung von OpenSSL übersetzt, erlangt er die Fähigkeit, mit einem Apache-Server über `https://`-URLs zu kommunizieren, wobei sämtlicher Verkehr durch einen Sitzungsschlüssel pro Verbindung verschlüsselt wird. Die vom Subversion-Client verwendete WebDAV-Bibliothek kann nicht nur Server-Zertifikate verifizieren, sondern nach Aufforderung auch Client-Zertifikate liefern.

Konfiguration von Subversion Server SSL Zertifikaten

Es würde den Rahmen dieses Buches sprengen, wenn beschrieben würde, wie SSL Client- und Server-Zertifikate erzeugt werden und wie Apache für ihre Verwendung konfiguriert wird. Viele andere Bücher, darunter Apaches eigene Dokumentation, erläutern diese Aufgabe.

SSL-Zertifikate von wohlbekannten Instanzen sind in der Regel kostenpflichtig, doch können Sie als Minimallösung Apache so konfigurieren, das er ein selbstgezeichnetes Zertifikat verwendet, dass durch ein Werkzeug wie etwa OpenSSL erzeugt wurde (http://openssl.org).[8]

Subversion-Client SSL Zertifikatsverwaltung

Bei einer Verbindung zu Apache über `https://` kann ein Subversion-Client zwei unterschiedliche Arten von Antworten empfangen:

- Ein Server-Zertifikat
- Eine Aufforderung zur Vorlage eines Client-Zertifikats

Server-Zertifikat

Wenn der Client ein Server-Zertifikat empfängt, muss er sicherstellen, dass der Server derjenige ist, für den er sich ausgibt. OpenSSL macht das, indem der Unterzeichner des Server-Zertifikats, die sogenannte *Certificate Authority* (CA), oder Zertifizierungsstelle, untersucht wird. Falls OpenSSL der CA nicht automatisch vertrauen kann, oder falls ein anderes Problem auftaucht (etwa ein abgelaufenes Zertifikat oder ein nicht übereinstimmender Rechnername), fragt Sie der Subversion-Kommandozeilenclient, ob Sie dem Server-Zertifikat dennoch vertrauen möchten:

```
$ svn list https://host.example.com/repos/project

Fehler bei der Validierung des Serverzertifikats für »https://host.example.com:443«:
 - Das Zertifikat ist nicht von einer vertrauenswürdigen Instanz ausgestellt
   Überprüfen Sie den Fingerabdruck, um das Zertifikat zu validieren!
Zertifikats-Informationen:
 - Hostname: host.example.com
 - Gültig: von Jan 30 19:23:56 2004 GMT bis Jan 30 19:23:56 2006 GMT
 - Aussteller: CA, example.com, Sometown, California, US
 - Fingerabdruck: 7d:e1:a9:34:33:39:ba:6a:e9:a5:c4:22:98:7b:76:5c:92:a0:9c:7b

Ve(r)werfen, (t)emporär akzeptieren oder (p)ermanent akzeptieren?
```

Dieser Dialog ist im Wesentlichen dieselbe Frage, die Sie bei Ihrem Web-Browser gesehen haben (der auch bloß ein weiterer HTTP-Client ist, so wie Subversion). Falls Sie die Option `(p)`ermanent auswählen, wird Subversion das Server-Zertifikat in Ihrem privaten Laufzeitbereich `auth/` zwischengespeichert, ebenso wie Ihr Anwendername und Passwort (siehe „Client-Zugangsdaten“), und diesem Zertifikat bei künftigen Protokollverhandlungen vertrauen.

Ihre Laufzeit-Datei `servers` ermöglicht es Ihrem Subversion-Client ebenso, automatisch bestimmten CAs zu vertrauen, entweder global oder pro Host. Setzen Sie die Variable `ssl-authority-files` auf eine durch Semikolons getrennte Liste ~~PEM-kodierter CA~~-Zertifikate:

[8] Obgleich selbstgezeichnete Zertifikate anfällig für „Man-in-the-Middle“-Angriffe sind, ist ein solcher Angriff schwieriger für einen laienhaften Beobachter durchzuführen als ungeschützte Passwörter abzuschnorcheln.

```
[global]
ssl-authority-files = /path/to/CAcert1.pem;/path/to/CAcert2.pem
```

Viele OpenSSL-Installationen besitzen auch eine vordefinierte Menge von „Standard“-CAs, denen nahezu allgemein vertraut wird. Damit der Subversion-Client diesen Standard-Zertifizierungsstellen automatisch vertraut, setzen Sie die Variable `ssl-trust-default-ca` auf `true`.

Client certificate challenge

Falls der Client eine die Aufforderung erhält, ein Client-Zertifikat vorzulegen, ersucht Apache den Client, sich zu identifizieren. Der Client muss ein Zertifikat zurückschicken, das von einer CA signiert wurde, der Apache vertraut, zusätzlich mit einer *Aufforderungserwiederung* (*Challenge Response*), die beweist, dass der Client in Besitz des zum Zertifikat gehörigen privaten Schlüssels ist. Für gewöhnlich wird der private Schlüssel und das Client-Zertifikat, durch eine lokale Passphrase geschützt, verschlüsselt auf Platte gespeichert. Wenn Subversion diese Aufforderung erhält, fragt es Sie nach dem Pfad zum Zertifikat und der Passphrase, das jenes schützt:

```
$ svn list https://host.example.com/repos/project

Anmeldebereich: https://host.example.com:443
Client Zertifikatsdatei: /path/to/my/cert.p12
Passphrase für »/path/to/my/cert.p12«:  ********
```

Beachten Sie, dass die Zugangsdaten des Clients in einer `.p12`-Datei gespeichert werden. Um ein Client-Zertifikat mit Subversion verwenden zu können, muss es im PKCS#12-Format vorliegen, was einem portablen Standard entspricht. Die meisten Web-Browser können Zertifikate in diesem Format im- und exportieren. Eine weitere Option ist es, die OpenSSL-Kommandozeilenwerkzeuge zu verwenden, um bestehende Zertifikate in PKCS#12 zu überführen.

Die Laufzeitdatei `servers` erlaubt Ihnen auch, diese Aufforderung pro Host zu automatisieren. Falls Sie die Variablen `ssl-client-cert-file` und `ssl-client-cert-password` setzen, kann Subversion automatisch auf Client-Zertifikatsanforderungen antworten, ohne bei Ihnen nachzufragen:

```
[groups]
examplehost = host.example.com

[examplehost]
ssl-client-cert-file = /path/to/my/cert.p12
ssl-client-cert-password = somepassword
```

Sicherheitsbewusstere Leute lassen möglicherweise `ssl-client-cert-password` weg, um zu vermeiden, die Passphrase im Klartext auf Platte zu speichern.

Extra Schmankerl

Die meisten Authentifikations- und Autorisierungsoptionen für Apache und **mod_dav_svn** haben wir abgehandelt. Es gibt jedoch noch ein paar weitere nette Dinge, die Apache zu bieten hat.

Stöbern im Projektarchiv

Einer der nützlichsten Vorteile eines Apache/WebDAV Aufbaus für Ihr Subversion Projektarchiv besteht darin, dass Ihre versionierten Dateien und Verzeichnisse unmittelbar mit einem gewöhnlichen Webbrowser betrachtet werden können. Da Subversion zur Identifizierung versionierter Ressourcen URLs verwendet, können diese URLs für den HTTP-basierten Zugriff direkt im Webbrowser eingetippt werden. Ihr Browser verschickt daraufhin für diesen URL eine HTTP `GET`-Anfrage; je

nachdem, ob dieser URL ein versioniertes Verzeichnis oder eine Datei repräsentiert, antwortet **mod_dav_svn** mit der Auflistung eines Verzeichnisinhalts oder mit dem Inhalt einer Datei.

URL Syntax

Falls die URLs keinerlei Informationen über die Ressourcenversion enthalten, die Sie sehen möchten, wird **mod_dav_svn** stets mit der jüngsten Version antworten. Diese Funktionalität hat den wundervollen Nebeneffekt, dass Sie Subversion-URLs als Dokumentverweise an Ihre Mitarbeiter weitergeben können, die stets auf die neuesten Ausprägungen dieser Dokumente zeigen werden. Natürlich können Sie diese URLs auch aus anderen Webseiten heraus verwenden.

Seit Subversion 1.6 unterstützt **mod_dav_svn** eine öffentliche URI Syntax zur Untersuchung älterer Revisionen sowohl von Dateien als auch Verzeichnissen. Die Syntax verwendet den Teil des Query-Strings des URL, um entweder die Peg-Revision oder die operative Revision oder beide anzugeben, die Subversion dann verwendet, um die in Ihrem Browser darzustellende Version zu ermitteln. Fügen Sie dem Query-String das Namens-Wert-Paar `p=`*`PEGREV`*, hinzu, wobei *`PEGREV`* eine Revisionsnummer ist, die die Peg-Revision festlegt, die Sie für die Abfrage anwenden möchten. Verwenden Sie `r=`*`REV`*, wobei *`REV`* eine Revisionsnummer ist, die die operative Revisionsnumer festlegt.

Wenn Sie beispielsweise die letzte Version einer Datei `README.txt` in `/trunk` Ihres Projektes sehen möchten, zeigen Sie mit Ihrem Webbrower auf die Projektarchiv-URL dieser Datei, die ähnlich der folgenden aussehen sollte:

```
http://host.example.com/repos/project/trunk/README.txt
```

Falls Sie nun eine ältere Version dieser Datei sehen wollten, fügen Sie dem Query-String der URL eine operative Revision hinzu:

```
http://host.example.com/repos/project/trunk/README.txt?r=1234
```

Was ist, falls das Objekt, das Sie sehen möchten, nicht mehr in der letzten Revision des Projektarchivs vorhanden ist? Hierbei ist eine Peg-Revision sehr hilfreich:

```
http://host.example.com/repos/project/trunk/deleted-thing.txt?p=321
```

Natürlich können Sie Peg-Revisions- und Angaben für operative Revisionen kombinieren, um genau anzugeben, was genau Sie sehen möchten:

```
http://host.example.com/repos/project/trunk/renamed-thing.txt?p=123&r=21
```

Der obige URL würde Revision 21 des Objekts anzeigen, das in Revision 123 an der Stelle `/trunk/renamed-thing.txt` im Projektarchiv lag. Siehe „Peg- und operative Revisionen“ für eine detaillierte detailed Erörterung dieser Konzepte von „Peg-Revision“ und „operativer Revision“. Sie könnten ein wenig schwerverständlich sein.

Zur Erinnerung: Diese Funktionalität von **mod_dav_svn** bietet nur ein eingeschränktes Stöbererlebnis im Projektarchiv. Sie können Verzeichnislisten und Dateiinhalte sehen, jedoch keine Revisionseigenschaften (wie etwa Protokollnachrichten) oder Datei- oder Verzeichniseigenschaften. Für Leute, die weitergehende Informationen zum Projektarchiv und seiner Geschichte benötigen gibt es hierfür mehrere Softwarepakete von Drittanbietern. Hierzu zählen beispielsweise ViewVC (http://viewvc.tigris.org), Trac (http://trac.edgewall.org) und WebSVN (http://websvn.info). Diese Werkzeuge von Drittanbietern beeinträchtigen nicht die eingebaute „stöberfähigkeit“ von **mod_dav_svn** und bieten im Allgemeinen weitergehende Funktionalität, wozu die Anzeige der eben erwähnten Mengen von Eigenschaften, die Anzeige von Unterschieden zwischen Dateirevisionen u.a. gehört.

Passender MIME-Typ

Während des Durchstöberns eines Subversion-Projektarchivs bekommt der Web-Browser Hinweise zur Darstellung des Inhalts einer Datei, indem er in den `Content-Type:`-Header von Apaches Antwort auf die HTTP `GET`-Anfrage schaut. Der Wert dieses Headers ist eine Art MIME-Typ. Standardmäßig teilt Apache den Web-Browsern mit, dass alle Dateien des Projektarchivs den „Standard“-MIME-Typen besitzen, normalerweise `text/plain`. Das kann jedoch frustrierend sein, wenn ein Anwender möchte, dass Dateien aus dem Projektarchiv etwas aussagekräftiger dargestellt werden; beispielsweise wäre es nett, wenn eine Datei `foo.html` aus dem Projektarchiv auch als HTML-Datei angezeigt würde.

Um das zu erreichen, müssen Sie nur sicherstellen, dass Ihre Dateien den passenden `svn:mime-type` gesetzt haben. Im Detail besprechen wir das in „Datei-Inhalts-Typ“. Sie können Ihren Client sogar so konfigurieren, dass er automatisch passende `svn:mime-type`-Eigenschaften an Dateien hängt, wenn sie das erste Mal in das Projektarchiv eingebracht werden (siehe „Automatisches Setzen von Eigenschaften“).

Um mit unserem Beispiel fortzufahren, wenn also jemand die Eigenschaft `svn:mime-type` mit dem Wert `text/html` an die Datei `foo.html` hänge, würde Apache Ihrem Browser wahrscheinlich mitteilen, dass die Datei als HTML darzustellen sei. Man könnte auch passende `image/*`-MIME-Type-Eigenschaften an Bilddateien hängen und somit eine komplette Webpräsenz direkt aus dem Projektarchiv heraus sichtbar machen! Solange die Webpräsenz keinen dynamisch erzeugten Inhalt hat, gibt es damit im Allgemeinen kein Problem.

Anpassung der Darstellung

Gemeinhin werden Sie mehr Nutzen aus URLs auf versionierte Dateien ziehen – hier liegt schließlich der interessante Inhalt. Gelegentlich werden Sie beim Durchstöbern eines Subversion-Verzeichnisinhalts feststellen, dass das zur Darstellung verwendete HTML sehr einfach ist und bestimmt nicht ästhetisch ansprechend (oder gar interessant). Um eine Anpassung dieser Verzeichnisdarstellungen zu ermöglichen, stellt Subversion einen XML-Index-Mechanismus zur Verfügung. Eine einzelne `SVNIndexXSLT`-Direktive im `Location`-Block des Projektarchivs in `httpd.conf` fordert **mod_dav_svn** auf, bei der Anzeige von Verzeichnisinhalten XML auszugeben und ein XSLT-Stylesheet Ihrer Wahl zu verwenden:

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  SVNIndexXSLT "/svnindex.xsl"
  …
</Location>
```

Wenn Sie die Direktive `SVNIndexXSLT` zusammen mit einem gestalterischen XSLT-Stylesheet verwenden, können Sie die Verzeichnisinhalte an das Farbschema und die bildliche Darstellung anderer Teile Ihrer Webpräsenz anpassen. Sollten Sie es vorziehen, können Sie auch die Beispiel-Stylesheets aus dem Verzeichnis `tools/xslt/` des Subversion-Quelltextpakets verwenden. Beachten Sie, dass die Pfadangabe des Verzeichnisses `SVNIndexXSLT` tatsächlich um einen URL-Pfad handelt – Browser müssen Ihre Stylesheets lesen können, um sie zu verwenden!

Anzeige von Projektarchiven

Falls Sie mit einem einzelnen URL eine Ansammlung von Projektarchiven über die Direktive `SVNParentPath` verfügbar machen, ist es auch möglich, dass Apache einem Web-Browser alle verfügbaren Projektarchive anzeigt. Sie müssen nur die Direktive `SVNListParentPath` aktivieren:

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  SVNListParentPath on
  …
</Location>
```

Falls ein Anwender nun mit dem Web-Browser auf den URL `http://host.example.com/svn/` geht, sieht er eine

Liste aller Projektarchive unterhalb von `/var/svn`. Offensichtlich kann dies ein Sicherheitsproblem sein, so dass dieser Mechanismus standardmäßig abgestellt ist.

Protokollierung von Apache

Da Apache im Grunde genommen ein HTTP-Server ist, beinhaltet er fantastisch anpassungsfähige Protokollierungsmöglichkeiten. Es würde den Rahmen dieses Buches sprengen, alle Protokollierungseinstellungen zu erörtern, doch soll darauf hingewiesen werden, dass selbst die gewöhnlichste `httpd.conf`-Datei Apache veranlasst, zwei Protokolldateien anzulegen: `error_log` und `access_log`. Diese Protokolldateien können an unterschiedlichen Orten liegen, werden normalerweise aber im Protokollbereich Ihrer Apache-Installation angelegt. (Unter Unix liegen sie oft in `/usr/local/apache2/logs/`.)

Die Datei `error_log` zeichnet sämtliche internen Fehler beim Betrieb von Apache auf. Die Datei `access_log` protokolliert jede von Apache empfangene eingehende HTTP-Abfrage. Das macht es einfach, festzustellen, von welchen IP-Adressen Subversion-Clients kommen, wie oft bestimmte Clients den Server benutzen, welche Anwender sich richtig anmelden und welche Abfragen erfolgreich sind oder fehlschlagen.

Da HTTP ein zustandsloses Protokoll ist, erzeugt selbst die einfachste Funktion eines Subversion Clients leider mehrere Netzwerkabfragen. Es ist sehr schwer, anhand der Datei `access_log` herzuleiten, was der Client tat; die meisten Funktionen sehen aus wie eine Folge kryptischer `PROPPATCH`-, `GET`-, `PUT`- und `REPORT`-Abfragen. Und, was alles noch komplizierter macht: viele Client-Funktionen schicken fast identische Anfragen, was ein Auseinanderhalten erschwert.

mod_dav_svn kann Ihnen jedoch helfen. Durch die Aktivierung einer „operativen Protokollierung“ können Sie **mod_dav_svn** veranlassen, eine gesonderte Protokolldatei anzulegen, die festhält, welche Art von Funktionen Ihre Clients auf höherer Ebene ausführen.

Um das zu bewerkstelligen, müssen Sie die Apache-Direktive `CustomLog` verwenden (die detailliert in der Dokumentation zu Apache beschrieben wird). Stellen Sie sicher, dass Sie die Direktive *außerhalb* Ihres Subversion `Location`-Blocks verwenden:

```
<Location /svn>
  DAV svn
  …
</Location>

CustomLog logs/svn_logfile "%t %u %{SVN-ACTION}e" env=SVN-ACTION
```

In diesem Beispiel veranlassen wir Apache, die spezielle Protokolldatei `svn_logfile` im standardmäßigen Verzeichnis für Apache Protokolldateien, `logs`, anzulegen. Die Variablen `%t` und `%u` werden durch die Zeit bzw. den Anwendernamen der Anfrage ersetzt. Die wirklich wichtigen Teile sind die zwei Instanzen von `SVN-ACTION`. Wenn Apache diese Variable sieht, ersetzt er den Wert der Umgebungsvariablen `SVN-ACTION`, die automatisch von **mod_dav_svn** belegt wird, wenn eine Client-Funktion auf hoher Ebene feststellt wird.

Statt also eine traditionelle `access_log`-Protokolldatei auswerten zu müssen, die etwa so aussieht:

```
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc/!svn/vcc/default HTTP/1.1" 207 398
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc/!svn/bln/59 HTTP/1.1" 207 449
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc HTTP/1.1" 207 647
[26/Jan/2007:22:25:29 -0600] "REPORT /svn/calc/!svn/vcc/default HTTP/1.1" 200 607
[26/Jan/2007:22:25:31 -0600] "OPTIONS /svn/calc HTTP/1.1" 200 188
[26/Jan/2007:22:25:31 -0600] "MKACTIVITY
/svn/calc/!svn/act/e6035ef7-5df0-4ac0-b811-4be7c823f998 HTTP/1.1" 201 227
…
```

können Sie eine weit verständlichere Datei `svn_logfile` durchgehen, die so aussieht:

```
[26/Jan/2007:22:24:20 -0600] - get-dir /tags r1729 props
[26/Jan/2007:22:24:27 -0600] - update /trunk r1729 depth=infinity
[26/Jan/2007:22:25:29 -0600] - status /trunk/foo r1729 depth=infinity
[26/Jan/2007:22:25:31 -0600] sally commit r1730
```

Zusätzlich zur Umgebungsvariablen `SVN-ACTION` besetzt **mod_dav_svn** auch die Variablen `SVN-REPOS` und `SVN-REPOS-NAME`, die den Dateisystempfad zum Projektarchiv bzw. dessen Basisnamen beinhalten. Es sei empfohlen, Referenzen auf eine oder beide dieser Variablen in Ihre `CustomLog` Formatbeschreibung einzufügen; besonders dann, falls Sie Informationen aus mehreren Projektarchiven in einer einzelnen Protokolldatei sammeln.

Eine vollständige Liste mit allen protokollierten Aktionen finden Sie unter „Protokollierung auf hohem Niveau“.

Proxy mit Weiterleitung beim Schreiben

Einer der netten Vorteile von Apache als Subversion-Server ist die Möglichkeit zur Einrichtung eines einfachen Abgleichs. Nehmen wir zum Beispiel an, dass Ihr Team über vier Standorte auf der Welt verteilt ist. Da das Subversion-Projektarchiv nur an einem davon untergebracht sein kann, ist es für die anderen drei Standorte kein Vergnügen, darauf zuzugreifen, da sie wahrscheinlich eine spürbar langsamere Verbindung und längere Antwortzeiten beim Aktualisieren und Abliefern von Code erdulden müssen. Eine leistungsfähige Lösung besteht darin, ein System aufzusetzen, das aus einem *Master*-Apache-Server und mehreren *Slave*-Apache-Servern besteht. Falls Sie an jedem Standort einen Slave-Server aufstellen, können die Anwender eine Arbeitskopie vom nächstgelegenen Slave auschecken. Alle Leseanfragen gehen an den Server vor Ort. Schreibanfragen werden automatisch an den einzigen Master-Server weitergeleitet. Wenn die Übergabe abgeschlossen ist, „schiebt“ der Master automatisch die neue Revision mithilfe des Abgleichswerkzeugs **svnsync** auf jeden Slave-Server.

Diese Konfiguration bewirkt eine riesige, für Ihre Anwender deutlich wahrnehmbare Geschwindigkeitszunahme, da der Netzverkehr von Subversion-Clients normalerweise zu 80—90% aus Leseabfragen besteht. Und wenn diese Abfragen von einem *lokalen* Server kommen, ist das ein Riesengewinn.

In diesem Abschnitt begleiten wir Sie durch eine Standard-Einrichtung dieses Ein-Master/Mehrere-Slaves-Systems. Denken Sie jedoch daran, dass auf Ihren Servern mindestens Apache 2.2.0 (mit geladenem **mod_proxy**) und Subversion 1.5 (**mod_dav_svn**) laufen muss.

Einrichtung der Server

Konfigurieren Sie zunächst die Datei `httpd.conf` des Master-Servers auf die übliche Art. Stellen Sie das Projektarchiv unter einem bestimmten URI zur Verfügung und richten Sie nach ihren Wünschen die Authentifizierung sowie Autorisierung ein. Sobald dies erledigt ist, konfigurieren Sie jeden Ihrer „Slave“-Server auf exakt dieselbe Art, fügen jedoch die besondere Direktive `SVNMasterURI` dem Block hinzu:

```
<Location /svn>
  DAV svn
  SVNPath /var/svn/repos
  SVNMasterURI http://master.example.com/svn
  …
</Location>
```

Diese neue Direktive teilt dem Slave-Server mit, alle Schreibanfragen an den Master weiterzuleiten. (Dies geschieht durch das Apache-Modul **mod_proxy** automatisch.) Gewöhnliche Leseanfragen werden jedoch immer noch von den Slaves bedient. Stellen Sie sicher, dass Ihre Master- und Slave-Server die gleichen Authentifikations- und Autorisierungs-Konfigurationen haben; falls sie nicht mehr synchron sein sollten, kann das zu heftigen Kopfschmerzen führen.

Als nächstes müssen wir uns um das Problem unendlicher Rekursion kümmern. Stellen Sie sich vor, was unter der gegenwärtigen Konfiguration passiert, wenn ein Subversion-Client eine Übergabe an den Master-Server vornimmt. Wenn die Übergabe abgeschlossen ist, benutzt der Server **svnsync**, um die neue Revision nach jedem Slave zu replizieren. Da sich aber **svnsync** wie ein gewöhnlicher Subversion-Client bei einer Übergabe verhält, wird der Slave sofort versuchen, die hereinkommende Schreibaufforderung zurück an den Master weiterzuleiten! Da kommt Freude auf.

Die Lösung des Problems besteht darin, den Master Revisionen an eine unterschiedliche `<Location>` auf den Slaves senden zu lassen. Dieser Ort ist dergestalt konfiguriert, dass Schreibanfragen *nicht* weitergeleitet werden, sondern normale Übergaben von der IP-Adresse des Masters (und nur von dort) angenommen werden:

```
<Location /svn-proxy-sync>
  DAV svn
  SVNPath /var/svn/repos
  Order deny,allow
  Deny from all
  # Nur Zugriffe auf diese Location von der IP-Adresse des Servers erlauben:
  Allow from 10.20.30.40
  …
</Location>
```

Einrichten der Replizierung

Nachdem Sie nun Ihre `Location`-Blöcke auf Mastern und Slaves konfiguriert haben, müssen Sie nun Ihren Master für die Replizierung zu den Slaves einrichten. Das geschieht auf die übliche Weise – mit **svnsync**. Wenn Sie dieses Werkzeug noch nicht kennen, können Sie Details unter „Projektarchiv Replikation“ nachlesen.

Stellen Sie zunächst sicher, dass jedes Slave-Projektarchiv ein `pre-revprop-change`-Hook-Skript hat, das Änderungen an Revisions-Eigenschaften aus der Ferne ermöglicht. (Das ist Standard, wenn von **svnsync** empfangen wird.) Melden Sie sich dann auf dem Master-Server an und konfigurieren jede der Slave-Projektarchiv-URIs, so dass sie Daten vom Master-Projektarchiv auf der lokalen Platte empfangen:

```
$ svnsync init http://slave1.example.com/svn-proxy-sync file:///var/svn/repos
Eigenschaften für Revision 0 kopiert.
$ svnsync init http://slave2.example.com/svn-proxy-sync file:///var/svn/repos
Eigenschaften für Revision 0 kopiert.
$ svnsync init http://slave3.example.com/svn-proxy-sync file:///var/svn/repos
Eigenschaften für Revision 0 kopiert.

# Die initiale Replizierung durchführe

$ svnsync sync http://slave1.example.com/svn-proxy-sync
Übertrage Daten ....
Revision 1 übertragen.
Eigenschaften für Revision 1 kopiert.
Übertrage Daten ....
Revision 2 übertragen.
Eigenschaften für Revision 2 kopiert.
…

$ svnsync sync http://slave2.example.com/svn-proxy-sync
Übertrage Daten ....
Revision 1 übertragen.
Eigenschaften für Revision 1 kopiert.
Übertrage Daten ....
Revision 2 übertragen.
Eigenschaften für Revision 2 kopiert.
…

$ svnsync sync http://slave3.example.com/svn-proxy-sync
Übertrage Daten ....
Revision 1 übertragen.
Eigenschaften für Revision 1 kopiert.
Übertrage Daten ....
Revision 2 übertragen.
Eigenschaften für Revision 2 kopiert.
…
```

Nachdem das erledigt ist, wird das `post-commit`-Hook-Skript des Master-Servers konfiguriert, damit **svnsync** auf jedem Slave-Server aufgerufen wird:

```
#!/bin/sh
# Post-Commit-Skript zum Replizieren der neu übergebenen Revision an die Slaves

svnsync sync http://slave1.example.com/svn-proxy-sync > /dev/null 2>&1 &
svnsync sync http://slave2.example.com/svn-proxy-sync > /dev/null 2>&1 &
svnsync sync http://slave3.example.com/svn-proxy-sync > /dev/null 2>&1 &
```

Die zusätzlichen Stückchen am Ende jeder Zeile sind zwar nicht notwendig, erlauben es aber den Sync-Befehlen, auf eine leise Art und Weise im Hintergrund zu laufen, so dass der Subversion-Client keine Ewigkeit auf den Abschluss der Übergabe warten muss. Zusätzlich zu diesem `post-commit`-Hook werden Sie außerdem einen `post-revprop-change`-Hook benötigen, damit, wenn ein Anwender beispielsweise eine Protokollnachricht verändert, die Slave-Server diese Änderung ebenfalls mitbekommen:

```
#!/bin/sh
# Post-revprop-Change-Skript zur Weitergabe der Änderung an den
Revisionseigenschaften an die Slaves

REV=${2}
svnsync copy-revprops http://slave1.example.com/svn-proxy-sync ${REV} > /dev/null
2>&1 &
svnsync copy-revprops http://slave2.example.com/svn-proxy-sync ${REV} > /dev/null
2>&1 &
svnsync copy-revprops http://slave3.example.com/svn-proxy-sync ${REV} > /dev/null
2>&1 &
```

Das Einzige, was wir hier ausgelassen haben, ist die Behandlung von Sperren auf Anwenderebene (der Sorte **svn lock**). Sperren werden vom Master-Server während der Übergabeoperationen durchgesetzt; alle Informationen zu Sperren werden jedoch während Leseoperationen wie **svn update** und **svn status** durch den Slave-Server verteilt. An und für sich müsste eine voll funktionsfähige Proxy-Umgebung die Sperrinformationen vom Master-zum Slave-Server perfekt replizieren. Leider sind die meisten der hierfür eingesetzten Mechanismen auf die eine oder andere Art unzureichend[9]. Viele Teams verwenden die Sperrfunktionalität von Subversion überhaupt nicht, so dass es Sie gar nicht betreffen könnte. Leider können wir den Teams, die Sperren verwenden, keine Empfehlungen aussprechen, wie diese Schwäche umgangen werden kann.

Warnungen

Nun sollte Ihr Master-Slave-Replizierungssystem einsatzbereit sein. An dieser Stelle sind einige Worte zur Warnung angebracht. Bedenken Sie, dass diese Replizierung nicht vollständig robust gegenüber Rechner- und Netzwerkausfällen ist. Wenn beispielsweise einer der automatisierten **svnsync**-Befehle aus irgendeinem Grund nicht vollständig abgeschlossen wird, beginnen die Slaves, hinterher zu hinken. Ihre entfernten Anwender werden sehen, dass sie Revision 100 übergeben haben; wenn sie allerdings **svn update** aufrufen, wird ihr lokaler Server ihnen mitteilen, dass Revision 100 noch nicht existiert! Natürlich wird das Problem automatisch mit der nächsten Übergabe behoben wenn das folgende **svnsync** erfolgreich ist – alle wartenden Revisionen werden dann repliziert. Trotzdem möchten Sie vielleicht eine zusätzliche Überwachung einrichten, um auf Synchronisierungsfehler hingewiesen zu werden, damit Sie in diesem Fall **svnsync** erneut aufrufen können.

Können wir eine Replizierung mit svnserve aufsetzen?

Falls Sie **svnserve** statt Apache als Server verwenden, können Sie sicherlich die Hook-Skripte Ihres Projektarchivs dergestalt konfigurieren, dass sie **svnsync** wie hier gezeigt aufrufen und somit eine Replizierung vom Master zu den Slaves bewirken. Leider besteht zum gegenwärtigen Zeitpunkt nicht die Möglichkeit, Slave-**svnserve**-Server dazu zu veranlassen, Schreibanfragen automatisch an den Master weiterzuleiten. Das bedeutet, dass Ihre Anwender nur Lesekopien von den Slave-Servern auschecken können. Sie müssten Ihre Slave-Server so konfigurieren, dass sie

[9] http://subversion.tigris.org/issues/show_bug.cgi?id=3457 verfolgt diese Probleme.

Schreibanfragen vollständig verbieten. Das könnte für die Bereitstellung von „Spiegeln“ mit Lesezugriff beliebter Open-Source-Projekte nützlich sein, jedoch stellt es kein transparentes Proxy-System dar.

Andere Funktionen von Apache

Einige der Funktionen, die Apache als robuster Webserver mitbringt, können auch zur Verbesserung der Funktionalität und Sicherheit in Subversion verwendet werden. Der Subversion-Client kann SSL (den bereits besprochenen Secure Sockets Layer) verwenden. Falls ihr Subversion-Client mit SSL-Unterstützung gebaut wurde, kann er auf Ihren Apache-Server mit `https://` zugreifen und sich einer verschlüsselten Netzwerksitzung von hoher Qualität erfreuen.

Gleichermaßen nützlich sind andere Funktionen der Beziehung zwischen Apache und Subversion, wie etwa die Möglichkeit, einen besonderen Port zu spezifizieren (statt des HTTP Standard-Ports 80), oder einen virtuellen Domain-Namen, unter dem das Subversion-Projektarchiv erreichbar sein soll, oder die Möglichkeit, das Projektarchiv über einen HTTP-Proxy zu erreichen.

Da **mod_dav_svn** eine Teilmenge des WebDAV/DeltaV-Protokolls spricht, ist es möglich, auf das Projektarchiv über DAV-Clients von Drittanbietern zuzugreifen. Die meisten modernen Betriebssysteme (Win32, OS X und Linux) besitzen die eingebaute Fähigkeit, einen DAV-Server als eine Standard-Netz-„Freigabe“ einzuhängen. Das ist eine komplizierte Angelegenheit, doch ebenso erstaunlich, wenn es implementiert ist. Zu Einzelheiten, siehe Anhang C, *WebDAV und Autoversionierung*.

Beachten Sie, dass es noch eine Anzahl weiterer kleiner Schräubchen gibt, an denen man bei **mod_dav_svn** drehen kann, die aber zu verworren sind, um sie hier im Kapitel aufzuführen. Eine vollständige Liste aller `httpd.conf` Direktiven, auf die **mod_dav_svn** reagiert, finden Sie unter „Anweisungen“.

Pfadbasierte Autorisierung

Sowohl Apache als auch **svnserve** können Anwendern Zugriffsechte gewähren (oder verweigern). Normalerweise geschieht das für das gesamte Projektarchiv: ein Anwender darf das Projektarchiv lesen (oder auch nicht) und ein Anwender darf in das Projektarchiv schreiben (oder auch nicht). Es ist jedoch ebenfalls möglich, feiner abgestufte Zugriffsregeln zu definieren. Ein Anwender dürfte nur in ein bestimmtes Verzeichnis des Projektarchivs schreiben, aber in kein anderes; ein weiteres Verzeichnis könnte bis auf wenige besondere Personen für niemanden lesbar sein. Da es sich auch bei Dateien um Pfade handelt, ist es sogar möglich, den Zugriff dateiabhängig einzurichten.

Beide Server verwenden ein gemeinsames Dateiformat, um diese pfadbasierten Zugriffsregeln zu beschreiben. Im Falle von Apache muss das Modul **mod_authz_svn** geladen und dann die Direktive `AuthzSVNAccessFile` (in der Datei `httpd.conf`) hinzugefügt werden, die auf Ihre eigene Zugriffsregeldatei verweist. (Eine vollständige Erklärung finden Sie unter „Verzeichnisweise Zugangskontrolle“.) Falls Sie **svnserve** verwenden, müssen Sie dafür sorgen, dass die Variable `authz-db` (in `svnserve.conf`) auf Ihre Zugriffsregeldatei zeigt.

Benötigen Sie wirklich pfadbasierte Zugriffskontrolle?

Viele Administratoren, die das erste Mal Subversion einrichten, neigen dazu, sich auf die pfadbasierte Zugangskontrolle zu stützen, ohne einen weiteren Gedanken darüber zu verschwenden. Gewöhnlicherweise ist einem Administrator bekannt, welche Entwicklerteams in den einzelnen Projekten arbeiten, so dass es einfach ist, nur bestimmten Teams Zugriff auf bestimmte Verzeichnisse zu geben. Es scheint eine normale Sache zu sein und kommt dem Wunsch des Administrators entgegen, eine strenge Kontrolle über das Projektarchiv aufrechtzuerhalten.

Bedenken Sie jedoch, dass mit dieser Funktionalität oft sichtbare (und unsichtbare) Kosten einhergehen. Zur sichtbaren Kategorie gehört die erhebliche Mehrarbeit, die ein Server leisten muss, um zu gewährleisten, dass der Anwender das entsprechende Lese- oder Schreibrecht für jeden Pfad besitzt; in bestimmten Situationen führt das zu spürbaren Leistungseinbußen. Zur unsichtbaren Kategorie zählt die Kultur, die Sie erzeugen. In den meisten Fällen, in denen bestimmte Anwender Änderungen in bestimmte Teile des Projektarchivs *nicht* übergeben sollten, bedarf diese soziale Übereinkunft keiner technischen Erzwingung. Teams können manchmal spontan miteinander arbeiten, jemand könnte jemand anderen aushelfen, indem eine Übergabe in einem Bereich erfolgt, in dem ersterer normalerweise nicht arbeitet. Wenn solche Dinge serverseitig unterbunden werden, entstehen Hindernisse für eine ungeplante Zusammenarbeit. Sie erzeugen außerdem eine Menge Regeln, die gepflegt werden müssen, da Projekte sich weiterentwickeln, neue Anwender hinzukommen usw. Die Verwaltung bedeutet jede Menge Mehrarbeit.

Vergessen Sie nicht, dass es sich hier um ein Versionskontrollsystem handelt! Selbst falls jemand versehentlich eine Änderung dorthin übergibt, wo sie eigentlich nicht hin sollte, ist es einfach, die Änderung rückgängig zu machen. Und sollte ein Anwender mit böser Absicht an eine falsche Stelle übergeben, ist es sowieso ein sozialen Problem und muss außerhalb von Subversion gelöst werden.

Bevor Sie also anfangen, die Zugriffsrechte von Anwendern einzuschränken, sollten Sie sich fragen, ob es tatsächlich einen triftigen Grund dafür gibt oder ob es sich für einen Administrator nur „gut anhört“. Entscheiden Sie, ob es sich lohnt, Servergeschwindigkeit zu opfern, und vergessen Sie nicht, dass es hier um sehr wenig Risiko geht: es ist schlecht, wenn man als Krücke für soziale Probleme von Technik abhängig wird.[10]

Als erwägenswertes Beispiel sollten Sie das Subversion-Projekt betrachten, das stets eine Vorstellung davon hatte, wer etwas wohin übergeben darf, dies jedoch immer auf der sozialen Ebene durchsetzte. Dies ist ein passendes Modell für Vertrauen in einer Gemeinschaft, besonders bei Open-Source-Projekten. Selbstverständlich existiert manchmal *tatsächlich* ein legitimes Erfordernis für pfadbasierte Zugriffskontrolle. Im Geschäftsumfeld können bestimmte Daten wirklich vertraulich sein, so dass der Zugriff darauf auf eine kleine Gruppe beschränkt werden muss.

Sobald Ihr Server weiß, wo sich Ihre Regeldateien befinden, ist es an der Zeit, die Regeln zu definieren.

Die Syntax der Datei ist dieselbe wie bei `svnserve.conf` und den Laufzeit-Konfigurationsdateien. Zeilen, die mit einer Raute (#) beginnen, werden ignoriert. In der einfachsten Form benennt jeder Abschnitt ein Projektarchiv und einen Pfad darin. Die authentifizierten Anwendernamen sind die Optionen und der entsprechende Wert beschreibt das Zugriffsrecht des Anwenders auf den Pfad im Projektarchiv, entweder `r` (nur lesend) oder `rw` (lesend und schreibend). Wird der Anwender gar nicht erwähnt, ist kein Zugriff erlaubt.

Genauer gesagt: Der Wert des Abschnittnamens hat entweder das Format `[projektarchiv-name:pfad]` oder `[pfad]`.

Für Zwecke der Zugriffskontrolle beachtet Subversion bei Namen und Pfaden von Projektarchiven nicht die Groß- oder Kleinschreibung, indem sie vor dem Vergleich mit dem Inhalt der Zugriffsdatei in Kleinbuchstaben umgewandelt werden. Verwenden Sie Kleinbuchstaben für die Inhalte der Abschnittsüberschriften Ihrer Zugriffsdatei.

Falls Sie die Direktive `SVNParentPath` verwenden, ist es wichtig, die Namen der Projektarchive in den Abschnitten anzugeben. Falls Sie sie weglassen, wird ein Abschnitt wie etwa `[/some/dir]` auf den Pfad `/some/dir` *jedes* Projektarchivs zutreffen. Falls Sie jedoch die Direktive `SVNPath` verwenden, ist es in Ordnung, in den Abschnitten nur Pfade zu definieren, da es ja schließlich nur ein einziges Projektarchiv gibt.

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r
```

Im ersten Beispiel hat der Anwender `harry` vollständigen Lese- und Schreibzugriff auf das Verzeichnis `/branches/calc/bug-142` im Projektarchiv `calc`, die Anwenderin `sally` hat jedoch nur Lesezugriff. Allen anderen Anwendern ist der Zugriff auf dieses Verzeichnis nicht gestattet.

mod_dav_svn bietet eine Direktive `SVNReposName` an, die es Administratoren ermöglicht, einen etwas menschenlesbareren Namen für ein Projektarchiv festzulegen:

```
<Location /svn/calc>
  SVNPath /var/svn/calc
  SVNReposName "Calculator Application"
…
```

[10] Ein in diesem Buch häufiges Thema!

Das gestattet **mod_dav_svn**, das Projektarchiv durch etwas anderes als den Basisnamen des Serververzeichnisses, im vorangegangenen Beispiel `calc`, zu identifizieren, wenn Verzeichnislisten zum Inhalt des Projektarchivs ausgegeben werden. Denken Sie jedoch daran, dass beim Suchen von Autorisierungsregeln in der Zugriffsdatei Subversion diesen Projektarchiv-Basisnamen verwendet und *nicht* irgendeinen konfigurierten menschenlesbaren Namen.

Selbstverständlich werden Berechtigungen von den Eltern- auf die Kindverzeichnisse vererbt. Das bedeutet, dass wir ein Unterverzeichnis mit unterschiedlichen Berechtigungen für Sally angeben können:

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r

# Sally bekommt nur für das Unterverzeichnis 'testing' Schreibrecht
[calc:/branches/calc/bug-142/testing]
sally = rw
```

Nun kann Sally im Verzeichnis `testing` des Zweigs zwar schreiben, an anderen Stellen allerdings immer noch nur lesen. Andererseits besitzt Harry weiterhin vollständigen Lese- und Schreibzugriff auf den gesamten Zweig.

Es ist ebenfalls möglich, einer Person über die Vererbungsregeln explizit Berechtigungen zu entziehen, indem die Variable mit dem Anwendernamen auf den leeren Wert gesetzt wird:

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r

[calc:/branches/calc/bug-142/secret]
harry =
```

In diesem Beispiel hat Harry Lese- und Schreibzugriff auf den gesamten Baum `bug-142`, jedoch überhaupt keinen Zugriff auf das darin befindliche Unterverzeichnis `secret`.

Man muss sich nur merken, dass der am genauesten angegebene Pfad stets am besten passt. Der Server versucht zunächst, den Pfad selbst abzugleichen, dann das Elternverzeichnis hiervon, dann dessen Elternverzeichnis usw. Unter dem Strich läuft es darauf hinaus, dass die Erwähnung eines bestimmten Pfades in der Zugriffsdatei stets die von Elternverzeichnissen ererbten Berechtigungen überdeckt.

Standardmäßig hat niemand irgendeine Zugriffsberechtigung auf das Projektarchiv. Das bedeutet, wenn mit einer leeren Datei begonnen wird, sollte mindestens für alle Anwender die Leseberechtigung für die Wurzel des Projektarchivs gewährt werden. Das kann mit der Stern-Variablen (`*`) erreicht werden, die für „alle Anwender“ steht:

```
[/]
* = r
```

Dies ist eine verbreitete Einstellung. Beachten Sie, dass im Abschnittsnamen kein Projektarchiv erwähnt wird. Das führt dazu, dass alle Projektarchive für alle Anwender der Welt lesbar sind. Sobald alle Anwender Lesezugriff auf die Projektarchive haben, können Sie bestimmten Anwendern für ausgewählte Verzeichnisse bestimmter Projektarchive die Berechtigung `rw` geben.

Die Zugriffsdatei erlaubt es Ihnen auch, ganze Anwendergruppen zu definieren, ähnlich der Unix-Datei `/etc/group`:

```
[groups]
calc-developers = harry, sally, joe
paint-developers = frank, sally, jane
everyone = harry, sally, joe, frank, jane
```

Gruppen können ebenso wie Anwendern Zugriffsberechtigungen erteilt werden. Sie werden von Anwendern durch einen „At“-Präfix (`@`) unterschieden:

```
[calc:/projects/calc]
@calc-developers = rw

[paint:/projects/paint]
jane = r
@paint-developers = rw
```

Ein weiterer wichtiger Punkt ist, dass Gruppenberechtigungen nicht durch die Berechtigungen individueller Anwender überschrieben werden. Vielmehr wird die *Kombination* aller passender Berechtigungen zugesichert. Im vorangehenden Beispiel ist Jane Mitglied der Gruppe `paint-developers`, die über Lese- und Schreibzugriff verfügt. Kombiniert mit der Regel `jane = r` ergibt das immer noch Lese- und Schreibzugriff für Jane. Zugriffsrechte für Gruppenmitglieder können allenfalls über die Gruppenberechtigungen hinaus erweitert werden. Die Einschränkung von Anwendern, die Gruppenmitglieder sind auf geringere Berechtigungen als deren Gruppenberechtigung ist nicht möglich.

Gruppen können auch definiert werden, indem sie andere Gruppen beinhalten:

```
[groups]
calc-developers = harry, sally, joe
paint-developers = frank, sally, jane
everyone = @calc-developers, @paint-developers
```

Subversion 1.5 brachte einige nützliche Erweiterungen für die Syntax der Zugriffsdatei: Anwendernamen-Aliase, Authentifizierungsklassen-Symbol und einen neuen Regel-Ausschlussmechanismus. Dieses alles vereinfacht die Wartung der Zugriffsdatei. Zunächst beschreiben wir die Funktionalität des Anwendernamen-Alias.

Einige Authentifikationssysteme erwarten und verwenden relativ kurze Anwendernamen, wie wir sie hier bereits beschrieben haben: `harry`, `sally`, `joe` usw. Andere Authentifikationssysteme jedoch, wie solche, die LDAP oder SSL-Client-Zertifikate verwenden, könnten wesentlich komplexere Anwendernamen verwenden. So könnte beispielsweise Harrys Anwendername in einem durch LDAP geschützten System `CN=Harold Hacker,OU=Engineers,DC=red-bean,DC=com` lauten. Mit derartigen Anwendernamen könnte die Zugriffsdatei mit langen oder undurchsichtigen Anwendernamen zugemüllt werden, was auch leicht zu Tippfehlern führen kann. Glücklicherweise ermöglichen es Anwendernamen-Aliase, den komplizierten Anwendernamen nur einmal in einer Anweisung einzutippen, die ihm ein einfacheres Alias zuteilt.

```
[aliases]
harry = CN=Harold Hacker,OU=Engineers,DC=red-bean,DC=com
sally = CN=Sally Swatterbug,OU=Engineers,DC=red-bean,DC=com
joe = CN=Gerald I. Joseph,OU=Engineers,DC=red-bean,DC=com
…
```

Sobald Sie eine Menge an Aliasen definiert haben, können Sie sich an allen Stellen der Zugriffsdatei über die Aliase auf die Anwender beziehen, an denen Sie sonst die eigentlichen Anwendernamen benutzt hätten. Setzen Sie einfach ein kaufmännisches Und vor den Alias, um ihn von einem normalen Anwendernamen zu unterscheiden:

```
[groups]
calc-developers = &harry, &sally, &joe
paint-developers = &frank, &sally, &jane
everyone = @calc-developers, @paint-developers
```

Sie könnten sich auch dazu entscheiden, Aliase zu verwenden, falls sich die Anwendernamen Ihrer Anwender häufig ändern. In diesem Fall müssen Sie bei Änderungen der Anwendernamen lediglich die Aliastabelle aktualisieren anstatt eine globale Such- und Ersetzungsoperation über die gesamte Zugriffsdatei vornehmen zu müssen.

Ferner unterstützt Subversion einige „magische“ Symbole, die Ihnen dabei helfen sollen, Regeln abhängig von der Authentifizierungsklasse des Anwenders zu vergeben. Ein solches Symbol ist `$authenticated`. Verwenden Sie dieses Symbol dort, wo Sie ansonsten einen Anwendernamen, einen Alias oder einen Gruppennamen in Ihren Autorisierungsregeln angeben würden, um die Zugriffsrechte zu deklarieren, die ein Anwender erteilt bekommt, der sich schon einmal mit einem Anwendernamen anmeldet. Ähnlich wird das Symbol `$anonymous` verwendet, mit der Ausnahme, dass es auf jeden anwendbar ist, der sich *nicht* mit einem Anwendernamen authentifiziert hat.

```
[calendar:/projects/calendar]
$anonymous = r
$authenticated = rw
```

Ein weiteres praktisches Stück magischer Zugriffsdatei-Syntax ist die Verwendung der Tilde (~) als eine Ausschlussmarkierung. Wenn Sie in Ihren Autorisierungsregeln einem Anwendernamen, einem Alias, einen Gruppennamen oder einem Authentifizierungsklassen-Symbol eine Tilde voranstellen, gilt diese Regel für Anwender, die *nicht* durch diese Regel erfasst werden. Obwohl es unnötigerweise etwas verwirrend erscheint, ist der folgende Block äquivalent zu dem aus dem vorangegangenen Beispiel:

```
[calendar:/projects/calendar]
~$authenticated = r
~$anonymous = rw
```

Ein weniger offensichtliches Beispiel könnte wie folgt aussehen:

```
[groups]
calc-developers = &harry, &sally, &joe
calc-owners = &hewlett, &packard
calc = @calc-developers, @calc-owners

# jeder calc-Teilnehmer hat Lese- und Schreibzugriff...
[calc:/projects/calc]
@calc = rw

# ...doch nur die Eigentümer dürfen Release-Tags erstellen und ändern.
[calc:/projects/calc/tags]
~@calc-owners = r
```

Alle obigen Beispiele verwenden Verzeichnisse, da es sich bei der Definition von Zugriffsrechten hierbei um den verbreitetsten Anwendungsfall handelt. Es ist jedoch ebenfalls möglich, die Zugriffsrechte auf Dateipfade zu beschränken.

```
[calendar:/projects/calendar/manager.ics]
harry = rw
sally = r
```

Teilweise Lesbarkeit und Checkouts

Falls Sie Apache als Ihren Subversion-Server verwenden und bestimmte Unterverzeichnisse Ihres Projektarchivs für bestimmte Anwender unlesbar gemacht haben, müssen Sie über ein mögliches suboptimales Verhalten von **svn checkout** Bescheid wissen.

Wenn der Client einen Checkout oder eine Aktualisierung über HTTP verlangt, macht er eine einzige Anfrage beim Server und erhält eine einzelne (oftmals umfangreiche) Antwort vom Server. Wenn der Server die Anfrage erhält, ist das die *einzige* Gelegenheit für Apache, die Authentifizierung des Anwenders einzufordern. Das hat einige merkwürdige Seiteneffekte. Wenn beispielsweise ein Unterverzeichnis des Projektarchivs nur für die Anwenderin Sally lesbar ist und der Anwender Harry ein Elternverzeichnis auscheckt, wird sein Client auf die initiale Aufforderung zur Authentifizierung als Harry antworten. Während der Server die umfangreiche Antwort erzeugt, besteht keine Möglichkeit beim Erreichen des besonderen Verzeichnisses eine erneute Aufforderung zu senden; das Verzeichnis wird somit einfach übergangen, anstatt den Anwender im passenden Moment aufzufordern, sich als Sally zu authentifizieren. Auf ähnliche Weise wird der komplette Checkout ohne Authentifizierung vollzogen, falls das Wurzelverzeichnis des Projektarchivs anonym für jeden lesbar ist; auch hier werden nicht lesbare Verzeichnisse übergangen, anstatt zwischendurch zur Authentifizierung aufzufordern.

Protokollierung auf hohem Niveau

Sowohl der Apache **httpd**- als auch der Subversion **svnserve**-Server bieten die Protokollierung von Subversion-Operationen auf hohem Niveau. Die Einstellung jeder dieser Server-Optionen zur Bereitstellung dieser Protokollstufe erfolgt natürlich auf unterschiedliche Weise; gleichwohl folgt jedes Ausgabeformat einer einheitlichen Syntax.

Um dieses hohe Protokollierungsniveau in **svnserve** zu ermöglichen, brauchen Sie nur beim Start des Servers die Kommandozeilenoption `--log-file` zu verwenden, deren Wert der Name der Datei ist, in die **svnserve** seine Protokollausgaben schreibt.

```
$ svnserve -d -r /path/to/repositories --log-file /var/log/svn.log
```

Um das Gleiche in Apache einzustellen, ist etwas mehr Arbeit notwendig, jedoch handelt es sich um eine Erweiterung von Apaches Standardkonfigurationsmechanismus für die Protokollausgabe (Näheres unter „Protokollierung von Apache“).

Es folgt eine eine Liste mit Protokollnachrichten von Subversion-Aktionen, die von seinem Protokollierungsmechanismus auf hohem Niveau erzeugt werden, gefolgt durch ein oder mehrere Beispiele einer Protokollnachricht wie sie in der Ausgabe erscheint.

Checkout oder Export

```
checkout-or-export /path r62 depth=infinity
```

Übergabe

```
commit harry r100
```

Diffs

```
diff /path r15:20 depth=infinity ignore-ancestry
diff /path1@15 /path2@20 depth=infinity ignore-ancestry
```

Holen eines Verzeichnisses

```
get-dir /trunk r17 text
```

Holen einer Datei

```
get-file /path r20 props
```

Holen einer Dateirevision

```
get-file-revs /path r12:15 include-merged-revisions
```

Holen von Informationen über eine Zusammenführung

```
get-mergeinfo (/path1 /path2)
```

Sperre

```
lock /path steal
```

Protokoll

```
log (/path1,/path2,/path3) r20:90 discover-changed-paths revprops=()
```

Wiederholen von Revisionen (svnsync)

```
replay /path r19
```

Änderung einer Revisions-Eigenschaft

```
change-rev-prop r50 propertyname
```

Liste von Revisions-Eigenschaften

```
rev-proplist r34
```

Status

```
status /path r62 depth=infinity
```

Umschalten

```
switch /pathA /pathB@50 depth=infinity
```

Entsperren

```
unlock /path break
```

Aktualisieren

```
update /path r17 send-copyfrom-args
```

Zur Erleichterung der Arbeit von Administratoren, die ihre Subversion Protokollausgaben nachbearbeiten möchten (vielleicht für Berichte oder zur Analyse), ist den Quelltextpaketen für Subversion ein Python-Modul unter `tools/server-side/svn_server_log_parse.py` beigelegt, das verwendet werden kann, um die Protokollausgaben von Subversion zu parsen.

Unterstützung mehrerer Zugriffsmethoden auf das Projektarchiv

Sie haben gesehen, wie auf viele verschiedene Weisen auf ein Projektarchiv zugegriffen werden kann. Ist es aber möglich, oder sicher, wenn auf Ihr Projektarchiv gleichzeitig mit mehreren Methoden zugegriffen wird? Die Antwort lautet: ja, vorausgesetzt, sie handeln ein bisschen vorausschauend.

Diese Prozesse benötigen jederzeit Lese- und Schreibzugriff auf Ihr Projektarchiv:

- Gewöhnliche Systemanwender, die einen Subversion-Client mit ihrer Kennung verwenden, um direkt über `file://` URLs auf das Projektarchiv zuzugreifen
- Gewöhnliche Systemanwender, die sich über durch SSH gestartete private **svnserve**-Prozesse verbinden, die unter ihrer Kennung auf das Projektarchiv zugreifen
- Ein **svnserve**-Prozess, entweder als Daemon oder von **inetd** gestartet, der unter einer bestimmten festen Kennung läuft
- Ein Apache **httpd**-Prozess, der unter einer bestimmten festen Kennung läuft

Das häufigste Problem, in das Administratoren laufen, besteht im Eigentumsverhältnis und den Zugriffsrechten des Projektarchivs. Hat jeder Prozess (oder Anwender) aus der obigen Liste die Rechte, die dem Projektarchiv zugrunde liegenden Dateien zu lesen und zu schreiben? Unter der Annahme, dass Sie ein Unix-ähnliches Betriebssystem haben, mag ein einfacher Ansatz darin bestehen, jeden möglichen Anwender des Projektarchivs in eine neue Gruppe `svn` aufzunehmen und dieser Gruppe die Eigentumsrechte über das Projektarchiv zu geben. Aber auch das reicht nicht, da ein Prozess die Datenbankdateien unter Verwendung einer widrigen umask schreiben könnte, so dass anderen Anwendern der Zugriff verwehrt würde.

Nach dem Einrichten einer gemeinsamen Gruppe für Projektarchiv-Anwender besteht der nächste Schritt darin, jeden Prozess, der auf das Projektarchiv zugreift, zu zwingen, eine passende umask zu verwenden. Für Anwender, die direkt auf das Projektarchiv zugreifen, können Sie das Programm **svn** in ein Skript umwandeln, das zunächst **`umask 002`** aufruft und dann das eigentliche Client-Programm **svn** startet. Ein ähnliches Skript können Sie für das Programm **svnserve** schreiben, und in das Startskript von Apache, `apachectl`, fügen Sie das Kommando **`umask 002`** ein. Zum Beispiel:

```
$ cat /usr/bin/svn
#!/bin/sh
```

```
umask 002
/usr/bin/svn-real "$@"
```

Ein weiteres bekanntes Problem tritt häufig auf Unix-ähnlichen Systemen auf. Wenn Ihr Projektarchiv beispielsweise auf Berkeley-DB aufsetzt, werden gelegentlich neue Dateien angelegt, um die Aktivitäten zu protokollieren. Selbst wenn die Gruppe **svn** vollständiger Eigentümer des Projektarchivs ist, müssen diese neu erstellten Protokolldateien nicht notwendigerweise derselben Gruppe gehören, was weitere Zugriffsprobleme für Ihre Anwender nach sich zieht. Ein guter Behelf besteht darin, das Gruppen-SUID-Bit für das Verzeichnis `db` des Projektarchivs zu setzen, was dazu führt, das alle neu erstellten Protokolldateien derselben Gruppe gehören wie das Elternverzeichnis.

Sobald Sie diese Hürden genommen haben, sollten alle notwendigen Prozesse auf Ihr Projektarchiv zugreifen können. Es scheint vielleicht etwas chaotisch und kompliziert, doch die Probleme, die bei gemeinsamen Zugriff mehrerer Anwender auf gemeinsame Dateien entstehen, sind Klassiker, die sich oft nicht elegant lösen lassen.

Glücklicherweise werden die meisten Administratoren von Projektarchiven niemals eine solch komplexe Konfiguration *benötigen*. Anwender, die auf Projektarchive des Rechners zugreifen möchten, an dem sie sich angemeldet haben, sind nicht auf URLs der Form `file://` beschränkt, sondern können den Apache-Server oder **svnserve** mit `localhost` als Server-Namen in deren `http://` oder `svn://` URL verwenden. Darüber hinaus wird das Betreiben mehrfacher Server-Prozesse für Ihr Projektarchiv Ihnen mehr Kopfschmerzen bereiten als nötig ist. Wir empfehlen, einen einzigen Server zu wählen, der Ihren Bedürfnissen am nächsten kommt und dabei zu bleiben!

Die Checkliste für svn+ssh://-Server

Es kann recht verzwickt sein, es hinzubekommen, dass sich ein Haufen Anwender mit bestehenden SSH-Konten ohne Zugriffsprobleme ein Projektarchiv teilt. Falls Sie von all den Dingen, die Sie (als Administrator) auf einem Unix-ähnlichen System erledigen müssen, verwirrt sind: Hier ist eine kurze Checkliste die noch einmal einige der in diesem Abschnitt besprochenen Themen zusammenfasst:

- Alle Ihrer SSH-Anwender müssen in der Lage sein, das Projektarchiv zu lesen und zu schreiben. Fassen Sie also alle SSH-Anwender in einer einzelnen Gruppe zusammen.
- Machen Sie das Projektarchiv vollständig zum Eigentum dieser Gruppe.
- Setzen Sie die Gruppenrechte auf lesen/schreiben.
- Ihre Anwender benötigen eine sinnvolle umask, wenn sie auf das Projektarchiv zugreifen. Also sollten sie sicherstellen, dass **svnserve** (`/usr/bin/svnserve` oder wo es sonst im `$PATH` liegt) tatsächlich ein Wrapper-Skript ist, dass **`umask 002`** aufruft und dann das eigentliche Programm **svnserve** startet.
- Ergreifen Sie ähnliche Maßnahmen bei Verwendung von **svnlook** und **svnadmin**. Starten Sie sie entweder mit einer sinnvollen umask oder verpacken sie wie eben beschrieben.

Kapitel 7. Subversion an Ihre Bedürfnisse anpassen

Versionskontrolle kann eine komplizierte Angelegenheit sein, ebenso sehr Kunst wie auch Wissenschaft, die unzählige Methoden anbietet, Dinge zu erledigen. Sie haben in diesem Buch über die verschiedenen Unterbefehle des Subversion-Kommandozeilen-Clients gelesen und die Optionen, die deren Verhalten verändern. In diesem Kapitel werden wir uns weitere Möglichkeiten ansehen, mit denen Sie Subversion an Ihre Arbeit anpassen können – die Einrichtung der Subversion-Laufzeitkonfiguration, die Verwendung externer Hilfsprogramme, wie Subversion mit der Locale Ihres Betriebssystems zusammenarbeitet, usw.

Laufzeit-Konfigurationsbereich

Subversion bietet viele optionale Verhaltensweisen, die der Benutzer kontrollieren kann. Viele dieser Optionen möchte ein Benutzer für alle Subversion-Operationen auswählen. Anstatt die Benutzer jedoch dazu zu zwingen, sich die Kommandozeilenargumente für diese Optionen zu merken, und sie bei jeder Operation anzugeben, verwendet Subversion Konfigurationsdateien, die in einen Subversion-Konfigurationsbereich ausgelagert sind.

Der *Konfigurationsbereich* von Subversion ist eine zweistufige Hierarchie aus Optionsnamen und deren Werten. Üblicherweise läuft es darauf hinaus, dass es ein Verzeichnis mit *Konfigurationsdateien* gibt (die erste Stufe), die einfache Textdateien im normalen INI-Format sind, wobei „Abschnitte“ die zweite Stufe liefern. Sie können diese Dateien einfach mit Ihrem Lieblingseditor (etwa Emacs oder vi) bearbeiten. Sie beinhalten Anweisungen, die vom Client gelesen werden, um festzustellen, welche der verschiedenen optionalen Verhaltensweisen die Benutzer bevorzugen.

Aufbau des Konfigurationsbereichs

Beim ersten Aufruf des **svn**-Kommandozeilen-Clients legt er einen benutzereigenen Konfigurationsbereich an. Auf Unix-Systemen erscheint dieser Bereich als ein Verzeichnis namens `.subversion` im Heimatverzeichnis des Benutzers. Auf Win32-Systemen erzeugt Subversion einen Ordner namens `Subversion` normalerweise innerhalb des Bereichs `Anwendungsdaten` im Verzeichnis für das Anwenderprofil (welches, nebenbei bemerkt, gewöhnlich ein verstecktes Verzeichnis ist). Jedoch ist dieser Ablageort auf dieser Plattform von System zu System verschieden und wird durch die Windows-Registrierungsdatenbank bestimmt.[1] Wir werden uns auf den benutzereigenen Konfigurationsbereich beziehen, indem wir dessen Unix-Namen `.subversion` verwenden.

Neben dem benutzereigenen Konfigurationsbereich berücksichtigt Subversion das Vorhandensein eines systemweiten Konfigurationsbereichs. Dieser erlaubt Administratoren, Standardeinstellungen für alle Benutzer auf einer Maschine vorzunehmen. Beachten Sie, dass der systemweite Konfigurationsbereich allein keine zwingende Verfahrensweise bestimmt – die Einstellungen im benutzereigenen Konfigurationsbereich heben die systemweiten auf, und letztendlich bestimmen an das Programm **svn** übergebene Kommandozeilenargumente das Verhalten. Auf Unix-ähnlichen Plattformen wird erwartet, dass der systemweite Konfigurationsbereich das Verzeichnis `/etc/subversion` ist; auf Windows-Maschinen wird nach dem Verzeichnis `Subversion` innerhalb des gemeinsamen `Anwendungsdaten`-Bereichs gesucht (auch hier, wie in der Windows-Registrierungsdatenbank angegeben). Anders als beim benutzereigenen Bereich versucht **svn** nicht, den systemweiten Konfigurationsbereich anzulegen.

Der benutzereigene Konfigurationsbereich enthält gegenwärtig drei Dateien – zwei Konfigurationsdateien (`config` und `servers`) und eine Datei `README.txt`, die das INI-Format beschreibt. Beim Anlegen enthalten die Dateien Standardwerte für alle unterstützten Optionen von Subversion, die meisten davon auskommentiert und in Gruppen zusammengefasst, mit Beschreibungen, wie die Werte das Verhalten von Subversion beeinflussen. Um ein bestimmtes Verhalten zu ändern, brauchen Sie nur die entsprechende Konfigurationsdatei in einen Editor zu laden und den Wert der passenden Option zu ändern. Falls Sie irgendwann auf die Standardeinstellungen zurückschalten wollen, können Sie einfach das Konfigurationsverzeichnis löschen (oder umbenennen) und dann irgendeinen harmlosen **svn**-Befehl ausführen, wie etwa **`svn --version`**. Ein neues Konfigurationsverzeichnis mit dem Standardinhalt wird dann angelegt.

Subversion erlaubt es Ihnen auch, individuelle Konfigurationsoptionen über die Kommandozeile zu überschreiben, wenn Sie die Option `--config-option` verwenden, die besonders nützlich ist, falls Sie eine vorübergehende Änderung des Verhaltens herbeiführen möchten. Weiteren zur richtigen Verwendung dieser Option unter „svn-Optionen“.

[1] Die Umgebungsvariable `APPDATA` zeigt auf den Bereich `Anwendungsdaten`, so dass Sie stets über `%APPDATA%\Subversion` auf diesen Ordner zugreifen können.

Der benutzereigene Konfigurationsbereich enthält auch einen Zwischenspeicher mit Authentisierungsdaten. Das Verzeichnis `auth` beinhaltet eine Reihe Unterverzeichnisse, die Teile zwischengespeicherter Informationen enthalten, welche von den verschiedenen durch Subversion unterstützten Authentifizierungsmethoden benutzt werden. Dieses Verzeichnis wird so erzeugt, dass nur der Benutzer selbst den Inhalt lesen kann.

Konfiguration und die Windows-Registrierungsdatenbank

Neben dem üblichen INI-basierten Konfigurationsbereich können Subversion-Clients auf Windows-Plattformen auch die Windows-Registrierungsdatenbank für die Konfigurationsdaten verwenden. Die Namen der Optionen und deren Werte sind die gleichen wie in den INI-Dateien. Die „Datei/Abschnitt-Hierarchie“ wird hierbei beibehalten, jedoch in einer etwas anderen Weise – bei diesem Schema sind Dateien und Abschnitte lediglich Ebenen im Schlüsselbaum der Registrierungsdatenbank.

Für systemweite Konfigurationswerte sucht Subversion unter dem Schlüssel `HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion`. Beispielsweise wird die Option `global-ignores`, die sich im Abschnitt `miscellany` der Datei `config` befindet, unter `HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Config\Miscellany\global-ignores` gefunden. Benutzereigene Konfigurationswerte sollten unter `HKEY_CURRENT_USER\Software\Tigris.org\Subversion` gespeichert werden.

Konfigurationsoptionen aus der Registrierungsdatenbank werden *vor* den entsprechenden dateibasierten Optionen gelesen, so dass die in den Konfigurationsdateien gefundenen Werte die entsprechenden Werte aus der Registrierungsdatenbank überschreiben. Mit anderen Worten: Subversion sucht auf Windows-Systemen nach Konfigurationsinformationen an den folgenden Stellen (niedrigere Nummern haben Vorrang vor höheren Nummern):

1. Kommandozeilenoptionen
2. Die benutzereigenen INI-Dateien
3. Die benutzereigenen Werte in der Registrierungsdatenbank
4. Die systemweiten INI-Dateien
5. Die systemweiten Werte in der Registrierungsdatenbank

Darüber hinaus unterstützt die Windows-Registrierungsdatenbank keine „Auskommentierungen“. Trotzdem ignoriert Subversion alle Optionsschlüssel, die mit einem Doppelkreuz (#) beginnen. Das erlaubt Ihnen, eine Subversion-Option auszukommentieren, ohne gleich den gesamten Schlüssel aus der Registrierungsdatenbank zu löschen; hierdurch wird die Wiederherstellung der Option wesentlich erleichtert.

Der **svn**-Kommandozeilen-Client versucht niemals, in die Windows-Registrierungsdatenbank zu schreiben oder dort einen Standard-Konfigurationsbereich anzulegen. Sie können benötigte Schlüssel mit dem Programm **REGEDIT** erzeugen. Alternativ können Sie auch eine `.reg`-Datei anlegen (so wie die in Beispiel 7.1, „Beispieldatei mit Einträgen für die Registrierungsdatenbank (.reg)“) und dann im Windows-Explorer auf das Icon dieser Datei doppelklicken, was bewirkt, dass die Daten mit der Registrierungsdatenbank zusammengeführt werden.

Beispiel 7.1. Beispieldatei mit Einträgen für die Registrierungsdatenbank (.reg)

```
REGEDIT4

[HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Servers\groups]

[HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Servers\global]
"#http-auth-types"="basic;digest;negotiate"
"#http-compression"="yes"
"#http-library"=""
"#http-proxy-exceptions"=""
"#http-proxy-host"=""
"#http-proxy-password"=""
"#http-proxy-port"=""
```

```
"#http-proxy-username"=""
"#http-timeout"="0"
"#neon-debug-mask"=""
"#ssl-authority-files"=""
"#ssl-client-cert-file"=""
"#ssl-client-cert-password"=""
"#ssl-pkcs11-provider"=""
"#ssl-trust-default-ca"=""
"#store-auth-creds"="yes"
"#store-passwords"="yes"
"#store-plaintext-passwords"="ask"
"#store-ssl-client-cert-pp"="yes"
"#store-ssl-client-cert-pp-plaintext"="ask"
"#username"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\auth]
"#password-stores"="windows-cryptoapi"

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\helpers]
"#diff-cmd"=""
"#diff3-cmd"=""
"#diff3-has-program-arg"=""
"#editor-cmd"="notepad"
"#merge-tool-cmd"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\tunnels]

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\miscellany]
"#enable-auto-props"="no"
"#global-ignores"="*.o *.lo *.la *.al .libs *.so *.so.[0-9]* *.a *.pyc *.pyo *.rej *~
#*# .#* .*.swp .DS_Store"
"#interactive-conflicts"="yes"
"#log-encoding"=""
"#mime-types-file"=""
"#no-unlock"="no"
"#preserved-conflict-file-exts"="doc ppt xls od?"
"#use-commit-times"="no"

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\auto-props]
```

Beispiel 7.1, „Beispieldatei mit Einträgen für die Registrierungsdatenbank (.reg)“ zeigt den Inhalt einer `.reg`-Datei: einige der am meisten benutzten Konfigurationsoptionen mit deren Standardwerten. Beachten Sie das Auftreten von systemweiten (für Netz-Proxy-Optionen) und benutzereigenen (u.a. Editoren und Passwortspeicherung) Einstellungen. Beachten Sie weiterhin, dass alle Optionen auskommentiert sind. Sie brauchen nur das Doppelkreuz (#) vom Anfang des Optionsnamens zu entfernen und den Wert nach Ihren Wünschen zu setzen.

Konfigurationsoptionen

In diesem Abschnitt werden wir die besonderen Laufzeit-Konfigurationsoptionen erörtern, die Subversion momentan unterstützt.

Servers

Die Datei `servers` enthält Subversion-Konfigurationsoptionen, die mit Netzschichten zu tun haben. In dieser Datei gibt es zwei besondere Abschnitte – `[groups]` und `[global]`. Der Abschnitt `[groups]` ist hauptsächlich eine Querverweistabelle. Die Schlüssel in diesem Abschnitt sind die Namen anderer Abschnitte dieser Datei; deren Werte sind *Globs* – Textsymbole, die möglicherweise Platzhalter enthalten – die mit dem Namen der Maschine verglichen werden, an die Subversion-Anfragen geschickt werden.

```
[groups]
beanie-babies = *.red-bean.com
collabnet = svn.collab.net
```

```
[beanie-babies]
…

[collabnet]
…
```

Bei Verwendung im Netz probiert Subversion den Namen des zu erreichenden Servers mit einem Gruppennamen im Abschnitt `[groups]` abzugleichen. Falls das gelingt, sucht Subversion nach einem Abschnitt in der Datei `servers`, dessen Name dem der abgeglichenen Gruppe entspricht. Aus diesem Abschnitt wird dann die eigentliche Netzkonfiguration gelesen.

Der Abschnitt `[global]` enthält die Einstellungen für all die Server, die nicht mit einem der Globs im Abschnitt `[groups]` abgeglichen werden konnten. Die in diesem Abschnitt verfügbaren Optionen sind exakt die gleichen, die auch in den anderen Server-Abschnitten dieser Datei erlaubt sind (außer natürlich im besonderen Abschnitt `[groups]`):

`http-auth-types`
: Dies ist eine durch Semikolon getrennte Liste von HTTP-Authentifizierungstypen, die der Client als akzeptabel erachten wird. Gültige Typen sind `basic`, `digest` und `negotiate`, mit dem standardmäßigen Verhalten, das irgendein dieser Authentifizierungstypen akzeptiert wird. Ein Client, der darauf besteht, keine Authentifizierungsdaten im Klartext zu übertragen, könnte beispielsweise dergestalt konfiguriert werden, dass der Wert dieser Option `digest;negotiate` ist, wobei `basic` in dieser Liste nicht auftaucht. (Beachten Sie, dass diese Einstellung nur von Subversions Neon-basierten HTTP-Modul berücksichtigt wird.)

`http-compression`
: Gibt an, ob Subversion versuchen soll, Netzanfragen an DAV-fähige Server zu komprimieren. Der Standardwert ist `yes` (obwohl Kompression nur dann stattfindet, wenn diese Fähigkeit in die Netzschicht hinein kompiliert worden ist). Setzen Sie diesen Wert auf `no`, um die Kompression abzuschalten, etwa wenn Sie Fehler bei Netzübertragungen suchen.

`http-library`
: Subversion stellt ein Paar aus Modulen für den Projektarchiv-Zugriff zur Verfügung, die das WebDAV-Netzprotokoll verstehen. Das mit Subversion 1.0 ausgelieferte Original ist `libsvn_ra_neon` (obwohl es damals `libsvn_ra_dav` hieß). Neuere Subversion-Versionen liefern auch `libsvn_ra_serf` mit, das eine unterschiedliche Implementierung verwendet und darauf zielt, einige der neueren HTTP-Konzepte zu unterstützen.

  Zum gegenwärtigen Zeitpunkt wird `libsvn_ra_serf` noch als experimentell betrachtet, obwohl es in gewöhnlichen Fällen ziemlich gut funktioniert. Um das Experimentieren zu fördern, stellt Subversion die Laufzeit-Konfigurationsoption `http-library` zur Verfügung, die es Benutzern ermöglicht, festzulegen (generell oder pro Server-Gruppe), welche WebDAV-Zugriffsmodule sie bevorzugen – `neon` oder `serf`.

`http-proxy-exceptions`
: Eine durch Komma getrennte Liste aus Namensmustern für Projektarchiv-Server, die, ohne einen Proxy-Rechner zu benutzen, direkt angesprochen werden sollen. Die Syntax für die Muster ist die gleiche wie bei Dateinamen in der Unix-Shell. Ein Projektarchiv-Server mit einem zu irgendeinem dieser Muster passenden Namen wird nicht über einen Proxy angesprochen.

`http-proxy-host`
: Der Name des Proxy-Rechners, über den Ihre HTTP-basierten Subversion-Anfragen geleitet werden müssen. Der Standardwert ist leer, was bedeutet, das Subversion HTTP-Anfragen nicht über einen Proxy-Rechner leitet und die Zielmaschine direkt anspricht.

`http-proxy-password`
: Das Passwort, das an den Proxy-Rechner übermittelt werden soll. Der Standardwert ist leer.

`http-proxy-port`
: Die Portnummer, die auf dem Proxy-Rechner verwendet werden soll. Der Standardwert ist leer.

`http-proxy-username`
: Der Anwendername, der an den Proxy-Rechner übermittelt werden soll. Der Standardwert ist leer.

`http-timeout`

Die Zeitspanne in Sekunden, in der auf eine Antwort vom Server gewartet werden soll. Falls Sie Probleme mit einer langsamen Netzverbindung haben, die zu Auszeiten bei Subversion-Operationen führen, sollten Sie den Wert dieser Option erhöhen. Der Standardwert ist `0`, der der zugrunde liegenden HTTP-Bibliothek, Neon, auffordert, deren Standardwert zu benutzen.

`neon-debug-mask`
Eine Ganzzahl, die der Neon HTTP-Bibliothek als Maske angibt, welche Ausgaben zur Fehlersuche sie ausgeben soll. Der Standardwert ist `0`, was alle Ausgaben zur Fehlersuche unterbindet. Für weitergehende Informationen über Subversions Verwendung von Neon siehe Kapitel 8, *Subversion integrieren*.

`ssl-authority-files`
Eine Liste aus durch Semikolon getrennten Pfaden zu Dateien, die Zertifikate der Zertifizierungsinstanzen (oder ZI) beinhalten, die vom Subversion-Client bei HTTPS-Zugriffen akzeptiert werden.

`ssl-client-cert-file`
Falls ein Wirtsrechner (oder eine Gruppe von Wirtsrechnern) ein SSL-Client-Zertifikat benötigt, werden Sie normalerweise nach dem Pfad zu Ihrem Zertifikat gefragt. Indem Sie diese Variable auf diesen Pfad setzen, kann Subversion automatisch Ihr Client-Zertifikat finden, ohne Sie zu fragen. Es gibt keinen Standard-Ablageort für Zertifikate; Subversion holt es über jeden Pfad, den Sie angeben.

`ssl-client-cert-password`
Falls Ihre SSL-Client-Zertifikatsdatei mit einem Kennwort verschlüsselt ist, wird Sie Subversion bei jeder Benutzung des Zertifikats danach fragen. Sollten Sie das lästig finden (und Sie nichts dagegen haben, das Passwort in der Datei `servers` zu speichern), können Sie diese Variable mit dem Kennwort des Zertifikats belegen. Sie werden dann nicht mehr gefragt.

`ssl-pkcs11-provider`
Der Wert dieser Option ist der Name des PKCS#11-Lieferanten, von dem ein SSL-Client-Zertifikat gezogen wird (falls der Server danach fragt). Diese Einstellung wird nur von Subversions Neon-basierten HTTP-Modul berücksichtigt.

`ssl-trust-default-ca`
Setzen Sie diese Variable auf `yes`, falls Sie möchten, dass Subversion automatisch den mit OpenSSL mitgelieferten ZI vertraut.

`store-auth-creds`
Diese Einstellung ist dieselbe wie `store-passwords`, mit der Ausnahme, dass die Zwischenspeicherung der *gesamten* Authentifizierungsinformationen unterbunden oder erlaubt wird: Anwendernamen, Passwörter, Server-Zertifikate und andere Typen von Zugangsdaten, die zwischengespeichert werden könnten.

`store-passwords`
Das weist Subversion an, vom Anwender nach Authentifizierungsaufforderung eingegebene Passwörter zwischenzuspeichern oder die Zwischenspeicherung zu unterbinden. Der Standardwert ist `yes`. Setzen Sie den Wert auf `no`, um die Zwischenspeicherung auf Platte zu unterbinden. Sie können diese Option für einen einmaligen Aufruf des Befehls **svn** überschreiben, wenn Sie den Kommandozeilenparameter `--no-auth-cache` verwenden (bei den Unterbefehlen, die ihn unterstützen). Weitere Informationen hierzu unter „Zwischenspeichern von Zugangsdaten“. Beachten Sie, dass unabhängig von dieser Option Subversion keine Passwörter im Klartext speichert, es sei denn, die Option `store-plaintext-passwords` ist auch auf `yes` gesetzt.

`store-plaintext-passwords`
Diese Variable ist nur auf UNIX-ähnlichen Systemen wichtig. Sie kontrolliert, was der Subversion-Client macht, falls das Passwort für den aktuelle Authentifizierungs-Bereich nur unverschlüsselt auf Platte in `~/.subversion/auth/` zwischengespeichert werden kann. Sie können sie auf `yes` oder `no` setzen, um unverschlüsselte Passwörter zu erlauben bzw. zu unterbinden. Der Standardwert ist `ask`, was den Subversion-Client dazu veranlasst, Sie jedes Mal zu fragen, wenn ein *neues* Passwort dem Zwischenspeicherungsbereich unter `~/.subversion/auth/` hinzugefügt werden soll.

`store-ssl-client-cert-pp`
Diese Option kontrolliert, ob Subversion vom Anwender bereitgestellte Passphrasen für SSL Client-Zertifikate zwischenspeichern soll. Der Standardwert ist `yes`. Setzen Sie ihn auf `no`, um die Zwischenspeicherung der Passphrasen zu verhindern.

`store-ssl-client-cert-pp-plaintext`
Diese Option kontrolliert, ob Subversion beim Zwischenspeichern einer Passphrase für ein SSL Client-Zertifikat die Speicherung auf Platte im Klartext vornehmen darf. Der Standardwert dieser Option ist `ask`, was den Subversion-Client dazu veranlasst, bei Ihnen nachzufragen, wenn eine *neue* Client-Zertifikat-Passphrase dem Speicherbereich in

`~/.subversion/auth/` hinzugefügt werden soll. Setzen Sie den Wert dieser Option auf `yes` oder `no` um Ihre Präferenz anzugeben und entsprechende Nachfragen zu vermeiden.

Config

Die Datei `config` enthält den Rest der momentan verfügbaren Laufzeitoptionen von Subversion – diejenigen, die nichts mit dem Netz zu tun haben. Zum gegenwärtigen Zeitpunkt werden nur wenige Optionen benutzt, die, in Erwartung künftiger Erweiterungen, jedoch ebenfalls in Abschnitten zusammengefasst sind.

Der Abschnitt `[auth]` enthält Einstellungen zur Authentifizierung und Autorisierung gegenüber Subversion-Projektarchiven. Er enthält das Folgende:

`password-stores`
: Diese Liste aus durch Komma getrennten Elementen bestimmt, welche der (wenn überhaupt) vom System bereitgestellten Passwortspeicher von Subversion zur Zwischenspeicherung von Authentifizierungsdaten in Erwägung gezogen werden sollen. Der Standardwert ist `gnome-keyring, kwallet, keychain, windows-crypto-api`, was für GNOME Keyring, KDE Wallet, Mac OS X Keychain bzw. Microsoft Windows Cryptography API steht. Aufgeführte Speicher, die auf dem System nicht zur Verfügung stehen, werden ignoriert.

`store-passwords`
: Der Eintrag dieser Option in der Datei `config` ist überholt. Sie steht nun als Konfigurationseintrag für jeden Server im Konfigurationsbereich `servers`. Details unter „Servers“

`store-auth-creds`
: Der Eintrag dieser Option in der Datei `config` ist überholt. Sie steht nun als Konfigurationseintrag für jeden Server im Konfigurationsbereich `servers`. Details unter „Servers“

Der Abschnitt `helpers` kontrolliert, welche externen Anwendungen Subversion zur Erledigung seiner Aufgaben benutzen soll. Gültige Optionen in diesem Abschnitt sind:

`diff-cmd`
: Der absolute Pfad zu einem Vergleichsprogramm, das verwendet wird, wenn Subversion „diff“-Ausgaben erzeugt (wie beim Aufruf des Befehls **svn diff**). Standardmäßig benutzt Subversion eine interne Vergleichsbibliothek – wenn diese Option gesetzt wird, benutzt es für diese Aufgabe ein externes Programm. Siehe „Verwenden externer Vergleichs- und Zusammenführungsprogramme“ für Details zur Verwendung solcher Programme.

`diff3-cmd`
: Der absolute Pfad zu einem Dreiwege-Vergleichsprogramm. Subversion verwendet dieses Programm, um Änderungen des Benutzers mit denjenigen aus dem Projektarchiv zusammenzuführen. Standardmäßig benutzt Subversion eine interne Vergleichsbibliothek – wenn diese Option gesetzt wird, benutzt es für diese Aufgabe ein externes Programm. Siehe „Verwenden externer Vergleichs- und Zusammenführungsprogramme“ für Details zur Verwendung solcher Programme.

`diff3-has-program-arg`
: Dieser Schalter sollte auf `true` gesetzt werden, falls das mit der Option `diff3-cmd` angegebene Programm den Parameter `--diff-program` zulässt.

`editor-cmd`
: Bestimmt das Programm, das Subversion verwendet, wenn den Benutzer nach bestimmten Arten textueller Metadaten gefragt wird sowie zur interaktiven Auflösung von Konflikten. Siehe „Verwendung externer Editoren“ für weitere Informationen zur Verwendung von externen Editoren mit Subversion.

`merge-tool-cmd`
: Gibt ein Programm an, das Subversion zur Durchführung von Dreiwege-Zusammenführungs-Operationen Ihrer versionierten Dateien benutzt. Siehe „Verwenden externer Vergleichs- und Zusammenführungsprogramme“ für Details zur Verwendung solcher Programme.

Der Abschnitt `[tunnels]` erlaubt Ihnen, neue Tunnel-Schemas für **svnserve** und `svn://` Client-Verbindungen zu definieren. Für weitere Details, siehe „Tunneln über SSH“.

Der Abschnitt `miscellany` ist für alles, was nicht woanders hingehört. In diesem Abschnitt finden Sie:

`enable-auto-props`
: Fordert Subversion auf, Eigenschaften von neu hinzugefügten oder importierten Dateien zu setzen. Der Standardwert ist `no` also sollte der Wert auf `yes` gesetzt werden, um diese Funktion zu ermöglichen. Der Abschnitt `[auto-props]` in dieser Datei gibt an, welche Eigenschaften bei welchen Dateien gesetzt werden sollen.

`global-ignores`
: Wenn Sie den Befehl **svn status** aufrufen, zeigt Subversion unversionierte Dateien und Verzeichnisse zusammen mit versionierten an, wobei erstere mit einem `?` markiert werden (siehe „Verschaffen Sie sich einen Überblick über Ihre Änderungen“). In dieser Anzeige können uninteressante unversionierte Objekte manchmal lästig sein – beispielsweise Objektdateien aus einer Programmübersetzung. Die Option `global-ignores` ist eine durch Leerzeichen getrennte Liste aus Globs, die die Namen der Dateien und Verzeichnisse beschreiben, die Subversion nicht anzeigen soll, sofern sie unversioniert sind. Der Standardwert ist `*.o *.lo *.la *.al .libs *.so *.so.[0-9]* *.a *.pyc *.pyo *.rej *~ #*# .#* .*.swp .DS_Store`.

  Ebenso wie **svn status** ignorieren auch die Befehle **svn add** und **svn import** beim Durchsuchen eines Verzeichnisses Dateien, deren Namen auf die Muster der Liste passen. Sie können dieses Verhalten für einen einzelnen Aufruf einer dieser Befehle aufheben, indem Sie entweder den Dateinamen ausdrücklich aufführen oder die Option `--no-ignore` angeben.

  Für Informationen über eine feinere Auswahl zu ignorierender Objekte, siehe „Ignorieren unversionierter Objekte“.

`interactive-conflicts`
: Diese boolesche Option bestimmt, ob Subversion versuchen soll, Konflikte interaktiv aufzulösen. Wenn der Wert `yes` ist (entspricht dem Standardwert), fordert Subversion den Benutzer auf, anzugeben, wie die Konflikte zu behandeln sind, wie in „Lösen Sie etwaige Konflikte auf“ beschrieben. Anderenfalls wird es den Konflikt einfach markieren und mit der Bearbeitung weitermachen, wobei die Konfliktauflösung auf einen späteren Zeitpunkt verschoben wird.

`log-encoding`
: Diese Variable bestimmt die Zeichensatzkodierung für Protokollnachrichten. Sie ist eine dauerhafte Form der Option `--encoding` (siehe „svn-Optionen“). Das Subversion-Projektarchiv speichert Protokollnachrichten in UTF-8 und geht davon aus, dass Ihre Protokollnachricht unter Verwendung der natürlichen Locale Ihres Betriebssystems verfasst wird. Sie sollten eine unterschiedliche Kodierung angeben, falls Ihre Protokollnachrichten in irgendeiner anderen Kodierung geschrieben werden.

`mime-types-file`
: Diese, in Subversion 1.5 neu hinzugekommene, Option gibt den Pfad einer MIME-Typ-Zuordnungs-Datei an, etwa die Datei `mime.types`, die vom Apache HTTP Server zur Verfügung gestellt wird. Subversion verwendet diese Datei, um neu hinzugefügten oder importierten Dateien MIME-Typen zuzuordnen. Siehe „Automatisches Setzen von Eigenschaften“ und „Datei-Inhalts-Typ“ für mehr Informationen zur Erkennung und Behandlung von Dateiinhaltstypen durch Subversion.

`no-unlock`
: Diese boolesche Option entspricht der Option `--no-unlock` des Befehls **svn commit**, die Subversion dazu auffordert, Sperren auf gerade übergebene Dateien nicht aufzuheben. Wird diese Laufzeitoption auf `yes` gesetzt, wird Subversion niemals automatisch Sperren aufheben, und es Ihnen überlassen, ausdrücklich **svn unlock** aufzurufen. Der Standardwert ist `no`.

`preserved-conflict-file-exts`
: Der Wert dieser Option ist eine durch Leerzeichen getrennte Liste von Dateiendungen, die Subversion bei der Erstellung von Konfliktdatei-Namen beibehalten soll. Standardmäßig ist die Liste leer. Diese Option ist neu in Subversion 1.5.

  Wenn Subversion Konflikte bei Änderungen an Dateiinhalten feststellt, delegiert es deren Auflösung an den Benutzer. Um bei der Auflösung dieser Konflikte behilflich zu sein, hält Subversion unveränderte Kopien der verschiedenen in Konflikt stehenden Dateiversionen in der Arbeitskopie vor. Standardmäßig haben diese Konfliktdateien Namen, die aus dem Originalnamen und einer speziellen Endung bestehen, etwa `.mine` oder `.REV` (wobei *REV* eine Revisionsnummer ist). Unter Betriebssystemen, die das Standardprogramm zum Öffnen und Bearbeiten einer Datei aus deren Dateiendung herleiten, ist es ein kleines Ärgernis, dass es durch das Anhängen dieser besonderen Endungen nicht mehr so einfach wird, die Datei mit dem passenden Standardprogramm zu bearbeiten. Wenn beispielsweise ein Konflikt bei der Datei `ReleaseNotes.pdf` auftaucht, könnte es sein, dass die Konfliktdateien `ReleaseNotes.pdf.mine` oder `ReleaseNotes.pdf.r4231` genannt werden. Obwohl Ihr System vielleicht so konfiguriert ist, dass es Dateien mit der Endung `.pdf` mit Adobes Acrobat Reader öffnet, gibt es wahrscheinlich keine Einstellung für ein Programm, das alle

Dateien mit der Endung `.r4231` öffnet.

Mithilfe dieser Konfigurationsoption können Sie dieses Ärgernis allerdings beseitigen. Bei allen Dateien mit den angegebenen Endungen wird dann nach wie vor eine besondere Endung angehängt, jedoch wird dann zusätzlich die Originalendung angefügt. Angenommen, `pdf` wäre eine der Endungen, die in dieser Liste vorkämen, so würden die für `ReleaseNotes.pdf` erzeugten Konfliktdateien im obigen Beispiel `ReleaseNotes.pdf.mine.pdf` und `ReleaseNotes.pdf.r4231.pdf` genannt werden. Da jede Datei auf `.pdf` endet, wird das korrekte Standardprogramm zum Öffnen verwendet.

`use-commit-times`
: Normalerweise besitzen die Dateien Ihrer Arbeitskopie Zeitstempel, die dem Zeitpunkt entsprechen, zu dem sie zuletzt von einem Prozess benutzt wurden, ob von einem Editor oder einem **svn**-Unterbefehl. Im Allgemeinen passt das für Softwareentwickler, da Buildsysteme oft auf Zeitstempel sehen, um festzustellen, welche Dateien neu übersetzt werden sollen.

  In anderen Situationen wäre es manchmal schön, wenn die Dateien der Arbeitskopie Zeitstempel hätten, die dem Zeitpunkt entsprächen, zu dem sie das letzte Mal im Projektarchiv geändert wurden. Der Befehl **svn export** versieht die von ihm erzeugten Bäume stets mit diesen „Zeitstempeln der letzten Übergabe“. Wenn diese Variable auf `yes` gesetzt wird, vergeben auch die Befehle **svn checkout**, **svn update**, **svn switch** und **svn revert** die Zeitstempel der letzten Übergabe an die von Ihnen berührten Dateien.

Der Abschnitt `auto-props` kontrolliert die Fähigkeit des Subversion-Clients, automatisch Eigenschaften auf Dateien zu setzen, wenn sie hinzugefügt oder importiert werden. Er enthält eine beliebige Anzahl von Schlüssel-Wert-Paaren der Form *`PATTERN = PROPNAME=VALUE`*`[;`*`PROPNAME=VALUE ...`*`]`, wobei `PATTERN` ein Muster ist, welches auf einen oder mehrere Dateinamen passt, und der Rest der Zeile eine durch Semikolon getrennte Liste von Eigenschafts-Zuweisungen ist. Wenn die Muster bei einer Datei mehrfach passen, führt das dazu, dass auch die jeweiligen Eigenschaften gesetzt werden; jedoch ist die Reihenfolge der Eigenschafts-Vergabe nicht garantiert, so dass Sie eine Regel nicht durch eine andere „aufheben“ können. Mehrere Beispiele dieser automatischen Eigenschafts-Vergabe finden Sie in der Datei `config`. Vergessen Sie schließlich nicht, `enable-auto-props` im Abschnitt `miscellany` auf `yes` zu setzen, falls Sie die automatische Vergabe aktivieren möchten.

Lokalisierung

Lokalisierung ist der Vorgang, Programme zu veranlassen, sich auf eine regionsspezifische Weise zu verhalten. Wenn ein Programm Nummern oder Daten auf eine Art und Weise formatiert, die typisch für Ihren Teil der Welt ist oder Meldungen in Ihrer Muttersprache ausgibt (oder in dieser Sprache Eingaben akzeptiert), heißt es, dass das Programm *lokalisiert* ist. Dieser Abschnitt beschreibt die von Subversion ergriffenen Schritte bezüglich Lokalisierung.

Locales verstehen

Die meisten modernen Betriebssysteme kennen den Begriff „aktuelle Locale“ – d.h., die Region oder das Land, deren bzw. dessen Lokalisierungskonventionen berücksichtigt werden. Diese Konventionen – typischerweise ausgewählt durch irgendeinen Laufzeit-Konfigurationsmechanismus des Rechners – beeinflussen die Art und Weise, in der Programme sowohl dem Benutzer Daten präsentieren, als auch Eingaben des Benutzers akzeptieren.

Auf den meisten Unix-ähnlichen Systemen können Sie die Werte der Locale-bezogenen Konfigurationsoptionen überprüfen, indem Sie den Befehl **locale** aufrufen:

```
$ locale
LANG=
LC_COLLATE="C"
LC_CTYPE="C"
LC_MESSAGES="C"
LC_MONETARY="C"
LC_NUMERIC="C"
LC_TIME="C"
LC_ALL="C"
$
```

Die Ausgabe ist eine Liste Locale-bezogener Umgebungsvariablen mitsamt deren aktuellen Werten. In diesem Beispiel sind alle Variablen auf die Standard-Locale `C` gesetzt; Benutzer können diese Variablen jedoch auf bestimmte Länder-/Sprachcode-Kombinationen setzen. Wenn jemand beispielsweise die Variable `LC_TIME` auf `fr_CA` setzt, wüssten Programme, dass sie Zeit- und Datumsinformationen so präsentieren sollen, wie es französischsprachige Kanadier erwarten. Und wenn jemand die Variable `LC_MESSAGES` auf `zh_TW` setzte, wüssten Programme, dass sie menschenlesbare Meldungen in traditionellem Chinesisch ausgeben sollen. Das Setzen der Variablen `LC_ALL` hat zur Folge, das jede Locale-Variable auf den selben Wert gesetzt wird. Der Wert von `LANG` wird als Standardwert für jede nicht gesetzte Locale-Variable verwendet. Rufen Sie den Befehl **`locale -a`** auf, um die Liste der verfügbaren Locales auf einem Unix-System anzeigen zu lassen.

Unter Windows wird die Einstellung der Locale über die „Regions- und Sprachoptionen“ der Systemsteuerung vorgenommen. Dort können Sie die Werte individueller Einstellungen aus den verfügbaren Locales ansehen, auswählen und sogar etliche der Anzeigeformatkonventionen anpassen (mit einem schwindelerregenden Detaillierungsgrad).

Wie Subversion Locales verwendet

Der Subversion-Client **svn** berücksichtigt die aktuelle Konfiguration der Locale auf zwei Weisen. Zunächst beachtet er den Wert der Variablen `LC_MESSAGES` und versucht, alle Meldungen in der angegebenen Sprache auszugeben. Zum Beispiel:

```
$ export LC_MESSAGES=de_DE
$ svn help cat
cat: Gibt den Inhalt der angegebenen Dateien oder URLs aus.
Aufruf: cat ZIEL[@REV]...
…
```

Dieses Verhalten funktioniert unter Unix und Windows auf dieselbe Weise. Beachten Sie jedoch, dass es sein kann, dass der Subversion-Client eine bestimmte Sprache nicht spricht, obwohl Ihr Betriebssystem die entsprechende Locale unterstützt. Um lokalisierte Meldungen zu erzeugen, müssen Freiwillige Übersetzungen für jede Sprache zur Verfügung stellen. Die Übersetzungen werden mit dem GNU gettext-Paket geschrieben, wodurch Übersetzungsmodule erzeugt werden, die die Dateiendung `.mo` haben. So heißt beispielsweise die deutsche Übersetzungsdatei `de.mo`. Diese Übersetzungsdateien werden irgendwo in Ihrem System installiert. Unter Unix liegen Sie typischerweise unter `/usr/share/locale/`, während sie unter Windows oft im Ordner `share\locale\` des Subversion-Installationsbereichs zu finden sind. Sobald es installiert ist, wird das Modul nach dem Programm benannt, für das es eine Übersetzung liefert. Die Datei `de.mo` könnte zum Beispiel schließlich als `/usr/share/locale/de/LC_MESSAGES/subversion.mo` installiert werden. Sie können feststellen, welche Sprachen der Subversion-Client spricht, indem Sie nachsehen, welche `.mo`-Dateien installiert sind.

Die zweite Art, auf der die Locale berücksichtigt wird, bezieht sich darauf, wie **svn** Ihre Eingaben interpretiert. Das Projektarchiv speichert alle Pfade, Dateinamen und Protokollnachrichten in UTF-8-kodiertem Unicode. In diesem Sinne ist das Projektarchiv *internationalisiert* – d.h., das Projektarchiv kann Eingaben in jeder menschlichen Sprache entgegennehmen. Das heißt aber auch, dass der Subversion-Client dafür verantwortlich ist, nur Dateinamen und Protokollnachrichten in UTF-8 an das Projektarchiv zu schicken. Um das zu bewerkstelligen, muss er die Daten aus der aktuellen Locale in UTF-8 umwandeln.

Nehmen wir zum Beispiel an, Sie erzeugen eine Datei namens `caffè.txt` und schreiben bei der Übergabe die Protokollnachricht „Adesso il caffè è più forte“. Sowohl der Dateiname als auch die Protokollnachricht enthalten Nicht-ASCII-Zeichen, doch da Ihre Locale auf `it_IT` gesetzt ist, weiß der Subversion-Client, dass er sie als italienisch interpretieren muss. Er verwendet einen italienischen Zeichensatz, um die Daten in UTF-8 umzuwandeln, bevor sie an das Projektarchiv gesendet werden.

Beachten Sie, dass das Projektarchiv sich *nicht* um den Inhalt von Dateien kümmert, obwohl es Dateinamen und Protokollnachrichten in UTF-8 verlangt. Subversion betrachtet Dateiinhalte als undurchsichtige Bytefolgen, und weder Client noch Server versuchen, den Zeichensatz oder die Kodierung der Inhalte zu verstehen.

Fehler bei der Zeichensatzumwandlung

Während Sie Subversion benutzen, könnten Sie über eine Fehlermeldung bezüglich der Zeichensatzumwandlung stolpern:

```
svn: Kann Zeichenkette nicht von der eigenen Codierung nach »UTF-8«
konvertieren:
…
svn: Kann Zeichenkette nicht von »UTF-8« in die eigene Codierung konvertieren:
…
```

Fehler wie diese treten typischerweise dann auf, wenn der Client eine Zeichenkette in UTF-8 aus dem Projektarchiv empfangen hat, aber nicht alle Zeichen dieser Zeichenkette in der Kodierung der aktuellen Locale wiedergegeben werden können. Wenn zum Beispiel Ihre aktuelle Locale `en_US` ist, aber ein Mitarbeiter einen japanischen Dateinamen übergeben hat, bekommen Sie wahrscheinlich diesen Fehler, wenn Sie bei einem **svn update** diese Datei empfangen.

Die Lösung besteht entweder darin, Ihre Locale auf irgendetwas zu setzen, das die empfangenen Daten in UTF-8 repräsentieren *kann* oder den Dateinamen oder die Protokollnachricht im Projektarchiv zu ändern. (Und vergessen Sie nicht, Ihrem Kollegen auf die Finger zu klopfen – Projekte sollten sich frühzeitig auf gemeinsame Sprachen einigen, so dass alle Mitarbeiter dieselbe Locale verwenden.)

Verwendung externer Editoren

Die offensichtlichste Art und Weise, Daten in Subversion zu bekommen, ist es, Dateien unter Versionskontrolle zu stellen, Änderungen an diesen Dateien zu übergeben usw. Aber neben versionierten Dateidaten leben auch andere Informationen in Ihrem Subversion-Projektarchiv. Einige dieser Informationen – Protokollnachrichten, Kommentare zu Sperren und einige Eigenschafts-Werte – sind textueller Natur und werden ausdrücklich von Benutzern geliefert. Die meisten dieser Informationen können dem Subversion-Kommandozeilen-Client mit den Optionen `--message` (`-m`) und `--file` (`-F`) bei den entsprechenden Unterbefehlen mitgegeben werden.

Jede dieser Optionen hat ihre Vor- und Nachteile. Wenn Sie beispielsweise eine Übergabe machen, funktioniert `--file` (`-F`) prima, falls Sie bereits eine Datei mit der Protokollnachricht vorbereitet haben. Wenn Sie es jedoch nicht gemacht haben, können Sie `--message` (`-m`) verwenden, um eine Protokollnachricht auf der Kommandozeile mitzugeben. Leider kann es knifflig werden, mehr als einen Einzeiler auf der Kommandozeile anzugeben. Benutzer verlangen mehr Flexibilität – auf Wunsch das Verfassen mehrzeiliger, formfreier Protokollnachrichten.

Subversion unterstützt das, indem es Ihnen erlaubt, einen externen Texteditor anzugeben, den es bei Bedarf startet, um Ihnen einen leistungsfähigeren Mechanismus zur Eingabe dieser textuellen Metadaten zu geben. Es gibt mehrere Möglichkeiten, Subversion mitzuteilen, welchen Editor Sie benutzen möchten. Subversion überprüft die folgenden Dinge in der angegebenen Reihenfolge, wenn es solch einen Editor starten möchte:

1. `--editor-cmd` Kommandozeilenoption
2. `SVN_EDITOR` Umgebungsvariable
3. `editor-cmd` Option der Laufzeitkonfiguration
4. `VISUAL` Umgebungsvariable
5. `EDITOR` Umgebungsvariable
6. Möglicherweise ein Standardwert, der in die Subversion-Bibliotheken eingebaut wurde (nicht in offiziell gebauten Versionen)

Der Wert aller dieser Optionen oder Variablen ist der Anfang einer Kommandozeile, die von der Shell ausgeführt werden soll. Subversion hängt an diese Kommandozeile ein Leerzeichen und den Pfadnamen einer temporären Datei, die editiert werden soll. Um einen Editor also mit Subversion verwenden zu können, muss er so aufgerufen werden können, dass dessen letzter Kommandozeilenparameter eine zu editierende Datei ist, die er beim Sichern überschreibt; und im Erfolgsfall ist ein Rückgabewert von Null zu liefern.

Wie beschrieben, können externe Editoren bei allen übergebenden Unterbefehlen (wie **svn commit** oder **import**, **svn mkdir** oder **delete** wenn ein URL-Ziel angegeben wird usw.) zur Eingabe von Protokollnachrichten verwendet werden, und Subversion versucht, den Editor automatisch zu starten, sofern Sie nicht eine der Optionen `--message` (`-m`) oder `--file` (`-F`) angeben. Der Befehl **svn propedit** ist fast ausschließlich für die Verwendung eines externen Editors gebaut worden. Seit Version 1.5, benutzt Subversion auch den konfigurierten externen Editor, wenn der Benutzer es auffordert, einen Editor zur interaktiven Konfliktauflösung zu starten. Seltsamerweise scheint keine Möglichkeit zu bestehen, einen externen Editor zur Eingabe von Kommentaren für Sperren zu verwenden.

Verwenden externer Vergleichs- und Zusammenführungsprogramme

Die Schnittstelle zwischen Subversion und externen Zwei- und Dreiwege-Vergleichsprogrammen geht zurück bis in eine Zeit, als sich die kontextabhängigen Vergleichsfähigkeiten von Subversion allein auf Aufrufe der GNU-diffutils-Werkzeuge stützten, insbesondere **diff** und **diff3**. Um das von Subversion benötigte Verhalten zu bekommen, wurden diese Werkzeuge mit mehr als einer handvoll Optionen und Parametern aufgerufen, von denen die meisten sehr werkzeugspezifisch waren. Einige Zeit später entwickelte Subversion seine eigene interne Vergleichsbibliothek, und als Ausfallsicherung wurden dem Subversion-Kommandozeilen-Client die Optionen `--diff-cmd` und `--diff3-cmd` hinzugefügt, so dass Benutzer auf einfache Art mitteilen konnten, dass sie die GNU-Werkzeuge diff und diff3 gegenüber der neumodischen internen Vergleichsbibliothek bevorzugen. Wenn diese Optionen verwendet wurden, ignorierte Subversion einfach die interne Vergleichsbibliothek und benutzte die externen Programmen mit den langen Argumentlisten und dem ganzen Drumherum. Uns so ist es noch heute.

Es dauerte nicht lange, bis einige Leute feststellten, das diese einfachen Konfigurationsmechanismen zur Festlegung der Benutzung der externen GNU-Werkzeuge diff und diff3, die an einem bestimmten Ort im System liegen, auch für andere Vergleichswerkzeuge verwendet werden können. Schließlich hat Subversion nicht überprüft, ob die Werkzeuge zur Werkzeugkette der GNU diffutils gehören. Der einzige konfigurierbare Aspekt bei der Verwendung dieser externen Werkzeuge ist allerdings der Speicherort im System – weder die Menge der Optionen noch die Reihenfolge der Parameter usw. Subversion übergibt all diese GNU-Werkzeug-Optionen an Ihr externes Vergleichswerkzeug, ohne zu berücksichtigen, ob das Programm sie überhaupt versteht. Und hier hört es für die meisten Benutzer auf, intuitiv zu sein.

Die Entscheidung, wann ein kontextabhängiges Zwei- oder Dreiwege-Vergleichsprogramm als Teil einer größeren Operation von Subversion gestartet wird, obliegt allein Subversion und wird unter anderem dadurch beeinflusst, ob die Dateien nach Maßgabe der Eigenschaft `svn:mime-type` menschenlesbar sind. Das bedeutet beispielsweise, selbst falls Sie über das raffinierteste Vergleichs- oder Zusammenführungsprogramm des Universums verfügten, welches Microsoft Word versteht, würde es niemals von Subversion aufgerufen, solange Ihre versionierten Word-Dokumente einen MIME-Typen hätten, der sie als nicht-menschenlesbar kennzeichnet (so wie `application/msword`). Mehr über MIME-Type-Einstellungen unter „Datei-Inhalts-Typ“

Viel später hat Subversion 1.5 die interaktive Auflösung von Konflikten eingeführt (beschrieben in „Lösen Sie etwaige Konflikte auf“). Eine den Benutzern durch diese Funktionalität angebotene Option ist die Fähigkeit, interaktiv ein Zusammenführungsprogramm eines Drittanbieters starten zu können. Wenn dieses Vorgehen gewählt wird, prüft Subversion, ob der Anwender ein solches Werkzeug für diesen Einsatzzweck bestimmt hat. Subversion prüft zunächst die Umgebungsvariable `SVN_MERGE` auf den Namen eines externen Zusammenführungswerkzeugs. Sollte diese Variable nicht gesetzt sein, wird im Wert der Laufzeitoption `merge-tool-cmd` nach derselben Information gesucht. Wird das eingestellte externe Zusammenführungsprogramm gefunden, wird es gestartet.

Während der allgemeine Zweck des Dreiwege-Vergleichs- und des Zusammenführungsprogramms weitestgehend derselbe ist (einen Weg zu finden, um getrennte, sich jedoch überlappende, Dateiänderungen zu harmonisieren), führt Subversion jede dieser Optionen zu unterschiedlichen Zeitpunkten aus unterschiedlichen Gründen aus. Die interne Dreiwege-Vergleichsmaschine und ihr optionaler externer Ersatz werden verwendet, wenn die Zusammenarbeit mit dem Anwender *nicht* erwartet wird. Tatsächlich kann eine durch ein solches Werkzeug herbeigeführte spürbare Verzögerung letztendlich zum Fehlschlagen einer zeitkritischen Subversion-Operation führen. Für den interaktiven Aufruf ist das externe Zusammenführungsprogramm vorgesehen.

Obwohl die Schnittstelle zwischen Subversion und einem externen Zusammenführungsprogramm wesentlich gradliniger ist als die zwischen Subversion und den diff und diff3 Werkzeugen, ist die Wahrscheinlichkeit doch ziemlich gering, ein solches Tool zu finden, dessen Aufrufkonventionen exakt den Erwartungen von Subversion entspricht. Der Schlüssel in der Benutzung

externer Vergleichs- und Zusammenführungswerkzeuge liegt in der Verwendung von Wrapper-Skripten, die die Eingabe von Subversion in irgendetwas umwandeln, das Ihr besonderes Vergleichsprogramm versteht, und dann die Ausgabe ihres Programms in ein Format zurück überführt, das Subversion erwartet. Die folgenden Abschnitte behandeln die Details solcher Erwartungen.

Externes diff

Subversion ruft externe diff-Programme mit Parametern auf, die für GNU diff passen und erwartet lediglich, dass das externe Programm mit einem, nach GNU diff Definition, Erfolg signalisierenden Rückgabewert zurückkommt. Für die meisten alternativen diff-Programme sind nur die Argumente an sechster und siebter Stelle interessant – die Pfade der Dateien, die die linke bzw. rechte Seite des Vergleichs repräsentieren. Beachten Sie, dass Subversion das diff-Programm jeweils einmal pro modifizierter Datei aufruft, die die Subversion-Operation berührt, falls Ihr Programm also asynchron läuft (oder als „Hintergrundprozess“), könnte es sein, dass mehrere Instanzen gleichzeitig ausgeführt werden. Schließlich erwartet Subversion, dass Ihr Programm den Rückgabewert 1 liefert, falls es Unterschiede entdeckt hat, oder 0, falls nicht – jeder andere Rückgabewert wird als fataler Fehler angesehen.[2]

Beispiel 7.2, „diffwrap.py“ und Beispiel 7.3, „diffwrap.bat“ sind „Verpackungs“-Vorlagen für externe diff-Werkzeuge in den Skriptsprachen Python bzw. Windows-Batch.

Beispiel 7.2. diffwrap.py

```
#!/usr/bin/env python
import sys
import os

# Geben Sie hier Ihr bevorzugtes diff-Programm an.
DIFF = "/usr/local/bin/my-diff-tool"

# Subversion liefert die benötigten Pfade als die letzten beiden Parameter.
LEFT  = sys.argv[-2]
RIGHT = sys.argv[-1]

# Aufruf des diff-Befehls (ändern Sie die folgende Zeile passend für
# Ihr diff-Programm).
cmd = [DIFF, '--left', LEFT, '--right', RIGHT]
os.execv(cmd[0], cmd)

# Rückgabewert 0 falls keine Unterschiede, 1 falls doch.
# Jeder andere Rückgabewert wird als fatal betrachtet.
```

Beispiel 7.3. diffwrap.bat

```
@ECHO OFF

REM  Geben Sie hier Ihr bevorzugtes diff-Programm an.
SET DIFF="C:\Program Files\Funky Stuff\My Diff Tool.exe"

REM Subversion liefert die benötigten Pfade als die letzten beiden Parameter.
REM Das sind die Parameter 6 und 7 (außer Sie benutzen svn diff -x, dann
REM ist alles möglich).
SET LEFT=%6
SET RIGHT=%7

REM Aufruf des diff-Befehls (ändern Sie die folgende Zeile passend für
REM Ihr diff-Programm).
```

[2] Das Handbuch zu GNU diff beschreibt es so: „Ein Rückgabewert 0 bedeutet, dass keine Unterschiede gefunden wurden, 1 bedeutet, dass einige Unterschiede gefunden wurden und 2 bedeutet Ärger.“

```
%DIFF% --left %LEFT% --right %RIGHT%

REM Rückgabewert 0 falls keine Unterschiede, 1 falls doch.
REM Jeder andere Rückgabewert wird als fatal betrachtet.
```

Externes diff3

Subversion ruft für die Ausführung nicht-interaktiver Zusammenführungen Dreiwege-Vergleichsprogramme auf. Parametern auf, die für das GNU diff3-Werkzeug passen und erwartet, dass das externe Programm einen Erfolg signalisierenden Rückgabewert liefert und der vollständige Inhalt als Ergebnis der beendeten Zusammenführung in den Standardausgabestrom geschrieben wird (damit Subversion diesen in die entsprechende Datei unter Versionskontrolle umleiten kann). Für die meisten alternativen Zusammenführungsprogramme sind nur die Argumente an neunter, zehnter und elfter Stelle interessant, die den Pfaden der Dateien entsprechen, die die Eingaben „eigene“, „ältere“ bzw. „fremde“ repräsentieren. Beachten Sie, dass Ihr Skript nicht beendet werden darf, bevor die Ausgabe an Subversion abgeliefert wurde, da Subversion auf die Ausgabe Ihres Zusammenführungsprogramms angewiesen ist. Wenn es schließlich beendet wird, sollte es einen Rückgabewert 0 im Erfolgsfall und 1 bei verbleibenden Konflikten zurückgeben – jeder andere Rückgabewert wird als fataler Fehler angesehen.

Beispiel 7.4, „diff3wrap.py“ und Beispiel 7.5, „diff3wrap.bat“ sind „Verpackungs“-Vorlagen für externe Dreiwege-Vergleichswerkzeuge in den Skriptsprachen Python bzw. Windows-Batch.

Beispiel 7.4. diff3wrap.py

```
#!/usr/bin/env python
import sys
import os


# Konfigurieren Sie hier Ihr bevorzugtes Dreiwege-Vergleichsprogramm
DIFF3 = "/usr/local/bin/my-diff3-tool"


# Subversion liefert die von uns benötigten Pfade als die letzten drei Parameter
MINE  = sys.argv[-3]
OLDER = sys.argv[-2]
YOURS = sys.argv[-1]


# Aufruf des Dreiwege-Vergleichs-Befehls (ändern Sie die folgende
# Zeile, damit es für Ihr Dreiwege-Vergleichsprogramm einen Sinn
# ergibt).
cmd = [DIFF3, '--older', OLDER, '--mine', MINE, '--yours', YOURS]
os.execv(cmd[0], cmd)


# Nach der Zusammenführung muss das Script den Inhalt der
# zusammengeführten Datei an die Standardausgabe schreiben. Machen Sie
# das wie Sie möchten.
# Nach erfolgreicher Zusammenführung wird 0 zurückgegeben; 1, falls
# offene Konflikte im Ergebnis zurückbleiben. Jeder andere Fehlercode
# wird als fatal behandelt.
```

Beispiel 7.5. diff3wrap.bat

```
@ECHO OFF
```

```
REM Konfigurieren Sie hier Ihr bevorzugtes Dreiwege-Vergleichsprogramm
SET DIFF3="C:\Program Files\Funky Stuff\My Diff3 Tool.exe"


REM Subversion liefert die von uns benötigten Pfade als die letzten
REM drei Parameter
REM Dies sind die Parameter 9, 10 und 11. Allerdings haben wir
REM gleichzeitig nur Zugriff auf neun Parameter, also verschieben wir
REM das Neun-Parameter-Fenster zweimal, damit wir das bekommen, was
REM wir benötigen
SHIFT
SHIFT
SET MINE=%7
SET OLDER=%8
SET YOURS=%9


REM Aufruf des Dreiwege-Vergleichs-Befehls (ändern Sie die folgende
REM Zeile, damit es für Ihr Dreiwege-Vergleichsprogramm einen Sinn
REM ergibt).
%DIFF3% - -older %OLDER% - -mine %MINE% - -yours %YOURS%


REM Nach der Zusammenführung muss das Script den Inhalt der
REM zusammengeführten Datei an die Standardausgabe schreiben. Machen
REM Sie das wie Sie möchten.
REM Nach erfolgreicher Zusammenführung wird 0 zurückgegeben; 1, falls
REM offene Konflikte im Ergebnis zurückbleiben. Jeder andere Fehlercode
REM wird als fatal behandelt.
```

External merge

Optional ruft Subversion ein externes Zusammenführungsprogramm als Teil der Unterstützung für interaktive Konfliktauflösung auf. Als Argumente für das Zusammenführungsprogramm werden die folgenden geliefert: der Pfad der unmodifizierten Basisdatei, der Pfad der „theirs“-Datei (die Änderungen der Autoren enthält), der Pfad der „mine“-Datei (die lokale Änderungen enthält), der Pfad der Datei, in die das Zusammenführungsprogramm letztendlich den zusammengeführte Inhalt schreiben soll und der Pfad der Arbeitskopie der in Konflikt stehenden Dateien (relativ zum ursprünglichen Ziel der Zusammenführungsoperation). Vom Zusammenführungsprogramm wird erwartet, dass es im Erfolgsfall 0 und im Fehlerfall 1 zurückgibt.

Beispiel 7.6, „mergewrap.py“ und Beispiel 7.7, „mergewrap.bat“ sind Vorlagen für externe Wrapper für das Zusammenführungsprogramm in der Python- bzw. Windows-Batch-Programmiersprache.

Beispiel 7.6. mergewrap.py

```
#!/usr/bin/env python
import sys
import os


# Geben Sie hier Ihr bevorzugtes diff-Programm an.
MERGE = "/usr/local/bin/my-merge-tool"


# Holen Sie sich die durch Subversion gelieferten Pfade
BASE   = sys.argv[1]
THEIRS = sys.argv[2]
MINE   = sys.argv[3]
MERGED = sys.argv[4]
WCPATH = sys.argv[5]
```

```
# Aufruf des merge-Befehls (ändern Sie die folgende Zeile passend für
# Ihr merge-Programm).
cmd = [DIFF3, '--base', BASE, '--mine', MINE, '--theirs', THEIRS,
               '--outfile', MERGED]
os.execv(cmd[0], cmd)


# Rückgabewert 0 bei erfolgreicher Zusammenführung, 1 bei noch
# unaufgelösten Konflikten. Alles andere wird als fatal behandelt.
```

Beispiel 7.7. mergewrap.bat

```
@ECHO OFF


REM Geben Sie hier Ihr bevorzugtes diff-Programm an.
SET DIFF3="C:\Program Files\Funky Stuff\My Merge Tool.exe"


REM Holen Sie sich die durch Subversion gelieferten Pfade
SET BASE=%1
SET THEIRS=%2
SET MINE=%3
SET MERGED=%4
SET WCPATH=%5


REM Aufruf des merge-Befehls (ändern Sie die folgende Zeile passend
REM für Ihr merge-Programm).
%DIFF3% --base %BASE% --mine %MINE% --theirs %THEIRS% --outfile %MERGED%


REM Rückgabewert 0 bei erfolgreicher Zusammenführung, 1 bei noch
REM unaufgelösten Konflikten. Alles andere wird als fatal betrachtet,
```

Zusammenfassung

Manchmal gibt es nur einen Weg, um Dinge richtig zu machen, manchmal gibt es mehrere. Den Entwicklern von Subversion ist bewusst, dass zwar der größte Teil seines bestimmten Verhaltens für die meisten Benutzer annehmbar ist, gleichwohl aber einige Nischen in seinen Funktionen vorhanden sind, für die ein allgemein glücklich machender Ansatz nicht existiert. In diesen Fällen bietet Subversion den Benutzern die Möglichkeit, ihm mitzuteilen, wie *sie* wollen, dass es sich verhält.

In diesem Kapitel betrachteten wir das Laufzeitkonfigurationssystem von Subversion sowie andere Mechanismen mit denen Benutzer das konfigurierbare Verhalten kontrollieren können. Wenn Sie jedoch ein Entwickler sind, wird Sie das nächste Kapitel noch einen Schritt weiter bringen. Es beschreibt, wie Sie Subversion noch weiter anpassen können, indem Sie Ihre eigene Software unter Zuhilfenahme der Bibliotheken von Subversion schreiben.

Kapitel 8. Subversion integrieren

Subversion ist modular entworfen: Es ist implementiert als eine Sammlung in C geschriebener Bibliotheken. Jede Bibliothek hat einen wohldefinierten Zweck und eine Programmierschnittstelle (API), die nicht nur für Subversion selbst sondern auch für andere Software zur Verfügung steht, die Subversion integrieren oder anderweitig programmseitig kontrollieren möchte. Außerdem ist die API von Subversion nicht nur für andere C-Programme verfügbar sondern auch für Programme, die in höheren Sprachen wie Python, Perl, Java und Ruby geschrieben sind.

Dieses Kapitel ist für diejenigen, die mit Subversion über sein API oder seine Schnittstellen in verschiedenen Programmiersprachen zusammenarbeiten wollen. Falls Sie robuste Skripte um Subversion herum schreiben möchten, um Ihr Leben einfacher zu machen, komplexere Integrationen zwischen Subversion und anderer Software entwickeln wollen oder sich einfach für die zahlreichen Bibliotheken von Subversion und deren Angebote interessieren, ist dies das Kapitel für Sie. Falls Sie jedoch nicht glauben, dass Sie sich auf dieser Ebene mit Subversion beschäftigen müssen, können Sie dieses Kapitel ruhig überspringen, ohne dass dadurch Ihre Erfahrung mit Subversion beeinträchtigt wird.

Schichtenmodell der Bibliotheken

Jede der Kernbibliotheken von Subversion gehört zu einer von drei Schichten – der Projektarchiv-Schicht, der Projektarchiv-Zugriffs-Schicht oder der Client-Schicht (siehe Abbildung 1, „Die Architektur von Subversion“ im Vorwort). Wir werden diese Schichten gleich untersuchen, doch zunächst wollen wir eine kurze Zusammenfassung der verschiedenen Bibliotheken präsentieren. Aus Konsistenzgründen beziehen wir uns auf die Bibliotheken mit ihren Unix-Namen ohne Dateiendung (`libsvn_fs`, `libsvn_wc`, `mod_dav_svn`, usw.).

libsvn_client
: Hauptschnittstelle für Client-Programme

libsvn_delta
: Prozeduren zum Vergleichen von Bäumen und Byte-Strömen

libsvn_diff
: Prozeduren für kontextuelle Vergleiche und Zusammenführungen

libsvn_fs
: Gemeinsame Dateisystemprozeduren und Modullader

libsvn_fs_base
: Das Berkeley-DB-Dateisystem

libsvn_fs_fs
: Das FSFS-Dateisystem

libsvn_ra
: Gemeinsame Prozeduren für den Projektarchiv-Zugriff und Modullader

libsvn_ra_local
: Modul für den lokalen Projektarchiv-Zugriff

libsvn_ra_neon
: Modul für den WebDAV-Projektarchiv-Zugriff

libsvn_ra_serf
: Ein weiteres (experimentelles) Modul für den WebDAV-Projektarchiv-Zugriff

libsvn_ra_svn
: Projektarchiv-Zugriff über das spezielle Protokoll

libsvn_repos
: Projektarchiv-Schnittstelle

libsvn_subr

Verschiedene nützliche Prozeduren

libsvn_wc
: Die Bibliothek zur Verwaltung der Arbeitskopie

mod_authz_svn
: Apache-Autorisierung-Modul für den Subversion-Zugriff über WebDAV

mod_dav_svn
: Apache-Modul, zur Abbildung von WebDAV-Operationen auf solche von Subversion

Die Tatsache, dass das Wort „verschiedene“ nur einmal in der obigen Liste auftaucht, ist ein gutes Zeichen. Das Entwicklerteam von Subversion nimmt es ernst damit, dass Funktionen in den richtigen Schichten und Bibliotheken untergebracht werden. Der vielleicht größte Vorteil des modularen Entwurfs liegt aus Sicht eines Entwicklers in der Abwesenheit von Komplexität. Als Entwickler können Sie sich schnell das „große Bild“ vorstellen, das es Ihnen erlaubt, bestimmte Funktionsbereiche relativ einfach zu lokalisieren.

Ein weiterer Vorteil der Modularität ist die Möglichkeit, ein gegebenes Modul durch eine völlig neue Bibliothek zu ersetzen, die dieselbe Programmierschnittstelle implementiert, ohne den Rest der Code-Basis zu beeinflussen. Eigentlich passiert das bereits mit Subversion. Die Bibliotheken `libsvn_ra_local`, `libsvn_ra_neon`, `libsvn_ra_serf` und `libsvn_ra_svn` implementieren alle die gleiche Schnittstelle und funktionieren als Plug-In für `libsvn_ra`. Alle vier kommunizieren mir der Projektarchiv-Schicht – `libsvn_ra_local` verbindet sich direkt mit dem Projektarchiv; die drei anderen machen das über das Netz. Die Bibliotheken `libsvn_fs_base` und `libsvn_fs_fs` sind noch ein Paar, das die gleiche Funktion auf verschiedene Art implementiert – beide sind Plug-Ins der gemeinsamen Bibliothek `libsvn_fs`.

Auch der Client selber unterstreicht die Vorteile der Modularität beim Entwurf von Subversion. Die meisten Funktionen, die für den Entwurf eines Subversion-Clients benötigt werden, sind in der Bibliothek `libsvn_client` vorhanden (siehe „Client-Schicht“). Obwohl das Subversion-Paket lediglich das Kommandozeilenprogramm **svn** mitbringt, stellen mehrere Programme von Drittanbietern verschiedene Spielarten graphischer Benutzeroberflächen zur Verfügung, die die gleichen APIs wie der Standard-Kommandozeilen-Client verwenden. Diese Art der Modularität hat eine große Rolle bei der Verbreitung verfügbarer Subversion-Clients sowie Integrationen in Entwicklungsumgebungen gespielt und durch Erweiterungen zur enormen Akzeptanzrate von Subversion selbst beigetragen.

Projektarchiv-Schicht

Wenn wir uns auf die Projektarchiv-Schicht von Subversion beziehen, reden wir üblicherweise über zwei grundlegende Konzepte – die Implementierung des versionierten Dateisystems (auf das mittels `libsvn_fs` zugegriffen wird, unterstützt durch dessen Plug-Ins `libsvn_fs_base` und `libsvn_fs_fs`) und die Logik des Projektarchivs, die es umgibt (implementiert in `libsvn_repos`). Diese Bibliotheken liefern die Speicher- und Auswertungsmechanismen für die verschiedenen Revisionen Ihrer versionskontrollierten Daten. Diese Schicht ist mit der Client-Schicht über die Projektarchiv-Zugriffs-Schicht verbunden und stellt, aus der Sicht des Benutzers von Subversion, das „andere Ende der Leitung“ dar.

Das Dateisystem von Subversion ist kein Dateisystem auf Kernel-Ebene, das im Betriebssystem installiert würde (so wie Linux ext3 oder NTFS), sondern ein virtuelles Dateisystem. Anstatt „Dateien“ und „Verzeichnisse“ als echte Dateien und Verzeichnisse abzulegen (in denen Sie mit Ihrem bevorzugten Shell-Programm navigieren können), verwendet es eins von zwei verfügbaren abstrakten Speicherverfahren – entweder eine Berkeley-DB-Datenbankumgebung oder eine Repräsentation über einfache Dateien. (Um mehr über die zwei Verfahren kennenzulernen, siehe „Auswahl der Datenspeicherung“.) Es hat seitens der Entwicklergemeinschaft ein beträchtliches Interesse bestanden, künftigen Versionen von Subversion die Fähigkeit zu verleihen, andere Datenbanksysteme zu verwenden, etwa über einen Mechanismus wie Open Database Connectivity (ODBC). Tatsächlich hat Google etwas ähnliches gemacht, bevor der Dienst Google Code Project Hosting gestartet wurde: Mitte 2006 wurde angekündigt, dass Mitarbeiter des Open-Source-Teams ein neues proprietäres Dateisystem-Plug-In für Subversion geschrieben hätten, welches Googles höchstskalierbare Bigtable-Datenbank zum Speichern verwendet.

Die von `libsvn_fs` exportierte Dateisystem-Programmierschnittstelle enthält die Funktionen, die Sie auch von jeder anderen Programmierschnittstelle für ein Dateisystem erwarten würden – Sie können Dateien und Verzeichnisse anlegen und löschen, kopieren und verschieben, den Inhalt verändern usw. Sie besitzt auch Funktionen, die nicht so verbreitet sind, wie etwa die Fähigkeit, Metadaten („Eigenschaften“) an jede Datei oder jedes Verzeichnis anzufügen, zu verändern oder zu entfernen. Zudem ist das Dateisystem von Subversion ein versioniertes Dateisystem, d.h., während Sie Ihren Verzeichnisbaum ändern, merkt sich Subversion, wie er vor Ihren Änderungen ausgesehen hat. Und vor den vorhergehenden Änderungen. Und davor. Und so weiter durch die gesamte versionierte Zeitachse bis zu dem Moment (und kurz davor), an dem Sie das erste Mal etwas dem Dateisystem hinzugefügt hatten.

Alle Änderungen, die Sie an Ihrem Baum vornehmen, geschehen im Kontext einer Subversion-Übergabe-Transaktion. Das Folgende ist eine vereinfachte allgemeine Vorgehensweise beim Ändern Ihres Dateisystems:

1. Eine Subversion-Übergabe-Transaktion starten.
2. Nehmen Sie Ihre Änderungen vor (Ergänzungen, Löschungen, Änderungen an Eigenschaften usw.).
3. Schließen Sie Ihre Transaktion ab.

Sobald Sie Ihre Transaktion abgeschlossen haben, werden Ihre Änderungen am Dateisystem dauerhaft als historische Artefakte gespeichert. Jeder dieser Zyklen erzeugt eine einzelne neue Revision Ihres Baums, und jede Revision ist für immer verfügbar als unveränderliche Momentaufnahme „der Dinge, wie sie mal waren“.

Die Transaktionsablenkung

Der Begriff einer Subversion-Transaktion kann leicht mit der Transaktionsunterstützung der darunter liegenden Datenbank verwechselt werden, besonders bei der großen Nähe der ersteren zum Berkeley-DB-Datenbank-Code in `libsvn_fs_base`. Beide Transaktionstypen dienen der Atomizität und Isolation. Mit anderen Worten: Transaktionen ermöglichen Ihnen, eine Serie von Aktionen durchzuführen, bei denen entweder alle oder keine ausgeführt wird – entweder werden alle Aktionen erfolgreich abgeschlossen oder *keine* wird überhaupt ausgeführt – und das auf eine Weise, dass andere Prozesse nicht behindert werden, die auf den Daten arbeiten.

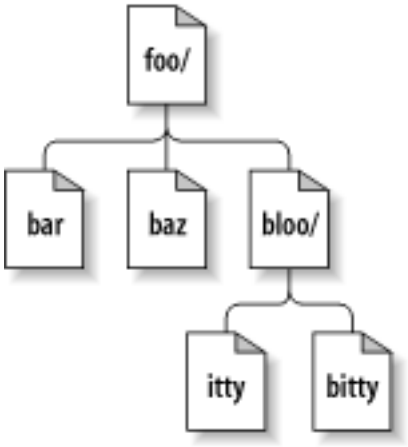
Datenbanktransaktionen umfassen im Allgemeinen kleinere Operationen, die in Zusammenhang mit der Änderung von Daten in der Datenbank selbst stehen (etwa die Änderung einer Tabellenzeile). Transaktionen von Subversion haben einen größeren Umfang und umfassen Operationen auf einer höheren Ebene, so etwa Änderungen an einer Menge aus Dateien und Verzeichnissen, die als nächste Revision im Dateisystembaum gespeichert werden sollen. Sollte das nicht bereits genug Verwirrung gestiftet haben, betrachten Sie die Tatsache, dass Subversion eine Datenbanktransaktion während der Erstellung einer Subversion-Transaktion benutzt (so dass beim Misslingen der Erstellung der Subversion-Transaktion die Datenbank in dem Zustand verbleibt, als hätten wir niemals versucht, eine Subversion-Transaktion zu starten)!

Glücklicherweise bleibt den Benutzern der Dateisystem-API die Transaktionsunterstützung des Datenbanksystems fast vollständig verborgen (wie es von einem ordentlich modularisierten Bibliotheksschema auch erwartet werden kann). Nur wenn Sie anfangen, in der Implementierung des Dateisystems herumzuwühlen, werden diese Dinge sichtbar (oder interessant).

Die meisten von der Dateisystemschnittstellen angebotenen Funktionen drehen sich um Aktionen, die auf einzelnen Dateisystempfaden stattfinden. Von außerhalb des Dateisystems betrachtet heißt das, dass der Hauptmechanismus zur Beschreibung und Handhabung einzelner Datei- und Verzeichnisrevisionen über Pfadzeichenketten wie `/foo/bar` erfolgt, genauso, als ob Sie Dateien und Verzeichnisse über Ihr bevorzugtes Shell-Programm ansprechen würden. Sie fügen neue Dateien und Verzeichnisse hinzu, indem Sie die künftigen Pfade an die passenden API-Funktionen übergeben. Sie können Informationen über den gleichen Mechanismus abrufen.

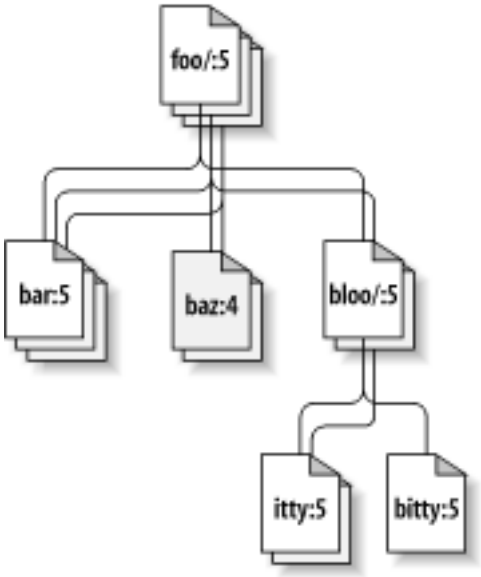
Im Gegensatz zu den meisten Dateisystemen reicht jedoch ein Pfad alleine nicht aus, um in Subversion eine Datei oder ein Verzeichnis zu identifizieren. Stellen Sie sich einen Verzeichnisbaum als ein zweidimensionales System vor, in dem Nachbarknoten eine Art horizontale und die Unterverzeichnisse eines Knotens eine vertikale Bewegung repräsentieren. Abbildung 8.1, „Dateien und Verzeichnisse in zwei Dimensionen“ zeigt genau das als typische Repräsentation eines Baums.

Abbildung 8.1. Dateien und Verzeichnisse in zwei Dimensionen



Der Unterschied ist hier, dass das Dateisystem von Subversion eine raffinierte dritte Dimension hat, die den meisten Dateisystemen fehlt – Zeit![1] Fast jede Funktion der Dateisystemschnittstelle, die ein *`path`*-Argument erwartet, benötigt auch ein *`root`*-Argument. Dieses Argument vom Typ `svn_fs_root_t` beschreibt entweder eine Revision oder eine Subversion-Transaktion (welche einfach eine Revision in Arbeit ist) und stellt die dritte Dimension des Kontexts zur Verfügung, die benötigt wird, um den Unterschied zwischen `/foo/bar` in Revision 32 und demselben Pfad in Revision 98 zu verstehen. Abbildung 8.2, „Versionierung der Zeit – die dritte Dimension!“ zeigt die Revisionsgeschichte als eine zusätzliche Dimension im Subversion-Dateisystem-Universum.

Abbildung 8.2. Versionierung der Zeit – die dritte Dimension!



Wie bereits erwähnt, mutet die `libsvn_fs`-API wie jedes andere Dateisystem an, außer dass es diese wundervolle Versionierungsmöglichkeit hat. Sie wurde entworfen, um für jedes Programm nutzbar zu sein, das an einem versionierten Dateisystem interessiert ist. Nicht nur zufällig hat Subversion selbst Interesse an dieser Funktion. Doch obwohl die Unterstützung der Dateisystemschnittstelle ausreichend für die einfache Versionierung von Dateien und Verzeichnissen ist, braucht Subversion mehr – und hier hat `libsvn_repos` seinen Auftritt.

Die Subversion-Projektarchiv-Bibliothek (`libsvn_repos`) sitzt (logisch) oberhalb der `libsvn_fs`-API und stellt zusätzliche Funktionen zur Verfügung, die über die grundlegende Logik eines versionierten Dateisystem hinausgehen. Sie umhüllt nicht alle Dateisystemfunktionen vollständig – lediglich bestimmte größere Schritte im allgemeinen Zyklus der Dateisystemaktivität. Einige dieser Schritte umfassen die Erzeugung und den Abschluss von Subversion-Transaktionen und die Änderung von Revision-Eigenschaften. Diese besonderen Ereignisse werden durch die Projektarchiv-Schicht gekapselt, da mit

[1]Wir sind uns bewusst, dass das für Science-Fiction-Fans, die lange Zeit davon ausgegangen sind, dass Zeit eigentlich die *vierte* Dimension ist, ein Schock sein kann, und wir bitten um Entschuldigung, falls die Geltendmachung einer unterschiedlichen Theorie unsererseits zu einem seelischen Schock führen sollte.

ihnen Hooks verknüpft sind. Ein System mit Projektarchiv-Hooks hat strenggenommen nichts mit der Implementierung eines versionierten Dateisystems zu tun, so dass es in der Projektarchiv-Bibliothek untergebracht ist.

Der Hook-Mechanismus ist aber nur ein Grund für die Abstraktion einer eigenständigen Projektarchiv-Bibliothek vom Rest des Dateisystemcodes. Die API `libsvn_repos` stellt mehrere andere wichtige Werkzeuge für Subversion zur Verfügung. Darunter fallen Fähigkeiten, um

- ein Subversion-Projektarchiv und das darin enthaltene Dateisystem zu erzeugen, zu öffnen, zu zerstören und hierauf Schritte zur Wiederherstellung auszuführen.
- die Unterschiede zwischen zwei Dateisystem-Bäumen zu beschreiben.
- die Übergabe-Protokollnachrichten aller (oder einiger) Revisionen abzurufen, in denen eine Menge aus Dateien im Dateisystem verändert wurde.
- einen menschenlesbaren „Auszug“ des Dateisystems zu erzeugen — eine vollständige Repräsentation der Revisionen im Dateisystem.
- dieses Auszugsformat zu lesen und die Revisionen in ein anderes Subversion-Projektarchiv zu laden.

Während sich Subversion weiterentwickelt, wird die Projektarchiv-Bibliothek gemeinsam mit der Dateisystem-Bibliothek wachsen und erweiterte Funktionen und konfigurierbare Optionen unterstützen.

Projektarchiv-Zugriffs-Schicht

Wenn die Subversion-Projektarchiv-Schicht das „andere Ende der Leitung“ repräsentiert, stellt die Projektarchiv-Zugriffs-Schicht (RA) die Leitung selbst dar. Ihre Aufgabe ist das Umherschaufeln von Daten zwischen den Client-Bibliotheken und dem Projektarchiv. Diese Schicht umfasst die Bibliothek `libsvn_ra` zum Laden von Modulen, die eigentlichen RA-Module (momentan `libsvn_ra_neon`, `libsvn_ra_local`, `libsvn_ra_serf` und `libsvn_ra_svn`) und alle zusätzlichen Bibliotheken, die von einer oder mehreren dieser RA-Module benötigt werden (so wie das Apache-Modul `mod_dav_svn` oder **svnserve**, der Server von `libsvn_ra_svn`).

Da Subversion URLs zum Identifizieren seiner Projektarchiv-Quellen benutzt, wird der Protokollteil des URL-Schemas (normalerweise `file://`, `http://`, `https://`, `svn://` oder `svn+ssh://`) verwendet, um festzustellen, welches RA-Modul die Kommunikation abwickelt. Jedes Modul hinterlegt eine Liste von Protokollen, die es „versteht“, so dass der RA-Lader zur Laufzeit bestimmen kann, welches Modul für die aktuelle Aufgabe benutzt werden kann. Sie können feststellen, welche RA-Module für der Kommandozeilen-Client zur Verfügung stehen und welche Protokolle sie zu verstehen vorgeben, indem Sie **`svn --version`** aufrufen:

```
$ svn --version
svn, Version 1.6.0
   übersetzt Mar 21 2009, 17:27:36

Copyright (C) 2000-2009 CollabNet.
Subversion ist Open-Source-Software, siehe http://subversion.tigris.org/
Dieses Produkt enthält Software, die von CollabNet (http://www.Collab.Net/)
entwickelt wurde.

Die folgenden ZugriffsModule (ZM) für Projektarchive stehen zur Verfügung:

* ra_neon : Modul zum Zugriff auf ein Projektarchiv über das Protokoll WebDAV mittels
Neon.
  - behandelt Schema »http«
  - behandelt Schema »https«
* ra_svn : Modul zum Zugriff auf ein Projektarchiv über das svn-Netzwerkprotokoll.
  - mit Cyrus-SASL-Authentifizierung
  - behandelt Schema »svn«
* ra_local : Modul zum Zugriff auf ein Projektarchiv auf der lokalen Festplatte
  - behandelt Schema »file«
* ra_serf : Modul zum Zugriff auf ein Projektarchiv über das Protokoll WebDAV mittels
serf.
  - behandelt Schema »http«
  - behandelt Schema »https«
```

$

Die von der RA-Schicht exportierte API beinhaltet Funktionen, die zum Senden und Empfangen versionierter Daten zum und vom Projektarchiv notwendig sind. Jedes der verfügbaren RA-Plug-Ins kann diese Aufgabe mithilfe eines besonderen Protokolls erledigen – `libsvn_ra_neon` sowie `libsvn_ra_serf` kommunizieren über HTTP/WebDAV (optional mit SSL-Verschlüsselung) mit einem Apache-HTTP-Server auf dem das Subversion-Server-Modul `mod_dav_svn` läuft; `libsvn_ra_svn` kommuniziert über ein maßgeschneidertes Netzprotokoll mit dem **svnserve** Programm usw.

Für diejenigen, die über ein völlig anderes Protokoll auf das Projektarchiv zugreifen möchten, sei gesagt, dass genau das der Grund für die Modularisierung der Projektarchiv-Zugriffsschicht ist. Entwickler können einfach eine neue Bibliothek schreiben, die auf der einen Seite die RA-Schnittstelle implementiert und auf der anderen Seite mit dem Projektarchiv kommuniziert. Ihre neue Bibliothek kann bestehende Netzprotokolle verwenden, oder Sie können Ihr eigenes erfinden. Sie könnten Aufrufe über Interprozess-Kommunikation (IPC) machen oder – mal etwas verrücktes – sogar ein auf E-Mail basiertes Protokoll implementieren. Subversion liefert die APIs, Sie sorgen für die Kreativität.

Client-Schicht

Auf der Client-Seite finden alle Aktionen in der Subversion-Arbeitskopie statt. Der größte Teil der in den Client-Bibliotheken implementierten Funktionen dient dem alleinigen Zweck, die Arbeitskopien zu verwalten – Verzeichnisse voller Dateien und anderer Unterverzeichnisse, die als eine Art lokaler, editierbarer „Spiegelung“ einer oder mehrere Orte im Projektarchiv dienen – und Änderungen an die RA-Schicht weiterzugeben oder von ihr zu empfangen.

Die Bibliothek für die Arbeitskopie von Subversion, `libsvn_wc`, ist direkt dafür verantwortlich, die Daten in den Arbeitskopien zu verwalten. Hierzu speichert die Bibliothek Verwaltungsinformationen zu jedem Verzeichnis der Arbeitskopie in einem besonderen Unterverzeichnis. Dieses Unterverzeichnis namens `.svn` kommt in jedem Unterverzeichnis der Arbeitskopie vor und beinhaltet zahlreiche weitere Dateien und Verzeichnisse, in denen der Zustand aufgezeichnet wird und die einen privaten Arbeitsbereich für Verwaltungsaufgaben liefern. Für diejenigen, die CVS kennen, ist der Zweck des Unterverzeichnisses `.svn` ähnlich den in CVS-Arbeitskopien zu findenden Verwaltungsverzeichnissen `CVS`. Weiter gehende Informationen zum `.svn`-Verwaltungsbereich finden sie später in diesem Kapitel unter „Innerhalb des Verwaltungsbereichs für Arbeitskopien“.

Die Subversion-Client-Bibliothek, `libsvn_client`, besitzt die weitestgehende Verantwortlichkeit; ihre Aufgabe ist es, die Funktionen der Arbeitskopie-Bibliothek mit denen der RA-Schicht zu vermischen und eine API auf höchster Ebene für Anwendungen zur Verfügung zu stellen, die allgemeine Versionskontrollaktionen durchführen wollen. Beispielsweise erwartet die Funktion `svn_client_checkout()` einen URL als Argument. Sie leitet diesen URL an die RA-Schicht weiter und startet eine authentifizierte Sitzung mit einem bestimmten Projektarchiv. Dann fragt sie das Projektarchiv nach einem bestimmten Baum und schickt diesen Baum an die Arbeitskopie-Bibliothek, die dann die gesamte Arbeitskopie auf die Platte schreibt (samt `.svn`-Verzeichnissen und allem Drum und Dran).

Die Client-Bibliothek ist so aufgebaut, dass sie von jeder Anwendung verwendet werden kann. Obwohl der Quelltext von Subversion einen Standard-Kommandozeilen-Client enthält, sollte es sehr einfach sein, eine beliebige Anzahl von graphischen Clients zu schreiben, die auf die Client-Bibliothek aufsetzen. Neue graphische Oberflächen (oder eigentlich jeder neue Client) für Subversion brauchen keine sperrigen Hüllen um den enthaltenen Kommandozeilen-Client zu sein – sie haben über die API von `libsvn_client` vollen Zugriff auf die Funktionen, Daten und Rückrufmechanismen die der Kommandozeilen-Client benutzt. Tatsächlich enthält der Quelltext von Subversion ein kleines C-Programm (das Sie unter `tools/examples/minimal_client.c` finden) welches beispielhaft zeigt, wie die Subversion API verwendet wird, um ein einfaches Client-Programm zu erzeugen.

Direkte Bindung – Ein Wort zur Genauigkeit

Warum sollte Ihr graphisches Programm direkt mit `libsvn_client` gebunden werden, statt als Umhüllung eines Kommandozeilenprogramms zu fungieren? Außer dass es einfach effizienter ist, kann es auch genauer sein. Ein Kommandozeilenprogramm (wie das von Subversion mitgelieferte), das mit der Bibliothek gebunden ist, muss gewissermaßen Rückmeldungen und abgefragte Daten aus C-Datentypen in menschenlesbare Ausgaben übersetzen. Diese Art der Übersetzung kann verlustbehaftet sein. Das heißt, es kann sein, dass das Programm nicht unbedingt alle Informationen anzeigt, die es von der API erhält, oder es könnte Teile von Informationen der kompakten Wiedergabe halber kombinieren.

Falls Sie ein solches Kommandozeilenprogramm mit noch einem anderen Programm umhüllen, hat das zweite

Programm nur Zugriff auf bereits aufbereitete (und, wie wir bemerkten, wahrscheinlich unvollständige) Informationen, die es *noch* einmal in *sein* Format umwandeln muss. Mit jeder neuen Umhüllung wird die Integrität der Originaldaten potentiell mehr und mehr verfälscht, ähnlich wie das Ergebnis einer Kopie einer Kopie (einer Kopie…) einer beliebten Audio- oder Videokassette.

Das überzeugendste Argument für das direkte Binden im Gegensatz zum Einhüllen anderer Programme ist jedoch, dass das Subversion-Projekt Kompatibilitätsversprechen bezüglich seiner APIs macht. Über Zwischenversionen dieser APIs hinweg (etwa zwischen 1.3 und 1.4) wird sich kein Prototyp einer Funktion ändern. Mit anderen Worten: Sie sind nicht gezwungen, den Quelltext Ihres Programms einfach aus dem Grund zu aktualisieren, weil Sie auf eine neue Version von Subversion umgestiegen sind. Von der Benutzung bestimmter Funktionen mag abgeraten worden sein, jedoch funktionieren sie noch, so dass Sie einen Zeitpuffer erhalten, um schließlich neuere APIs zu integrieren. Diese Art des Kompatibilitätsversprechens gilt nicht für die Ausgaben der Kommandozeilenprogramme von Subversion, die sich von Version zu Version ändern kann.

Innerhalb des Verwaltungsbereichs für Arbeitskopien

Wie bereits erwähnt, besitzt jedes Verzeichnis einer Subversion-Arbeitskopie ein besonderes Unterverzeichnis namens `.svn`, das Verwaltungsdaten zum Arbeitskopieverzeichnis beherbergt. Subversion verwendet die Informationen in `.svn`, um Dinge wie diese zu verfolgen:

- welche Orte des Projektarchivs durch die Dateien und Unterverzeichnisse der Arbeitskopie repräsentiert werden
- welche Revision jeder dieser Dateien und Verzeichnisse momentan in der Arbeitskopie vorhanden ist
- irgendwelche benutzerdefinierten Eigenschaften, die diesen Dateien und Verzeichnissen zugewiesen sein könnten
- ursprüngliche (unbearbeitete) Kopien der Dateien in der Arbeitskopie

Die Struktur sowie der Inhalt des Verwaltungsbereichs einer Subversion-Arbeitskopie werden als Implementierungsdetails betrachtet, die nicht für menschliche Bearbeitung vorgesehen sind. Entwicklern wird nahegelegt, die öffentlichen APIs oder die von Subversion zur Verfügung gestellten Werkzeuge zu benutzen, um die Daten der Arbeitskopie zu bearbeiten, statt diese Dateien direkt zu lesen und zu verändern. Die von der Arbeitskopie-Bibliothek für ihre Verwaltungsdaten verwendeten Dateiformate ändern sich hin und wieder – eine Tatsache, die dem Durchschnittsanwender dank der öffentlichen APIs verborgen bleibt. Nur um Ihre unbändige Neugier zu stillen, werden wir in diesem Abschnitt einige dieser Implementierungsdetails offenlegen.

Die Datei entries

Die vielleicht wichtigste Datei im Verzeichnis `.svn` ist die Datei `entries`. Sie enthält den größten Teil der Verwaltungsinformationen zu den versionierten Elementen in einem Verzeichnis der Arbeitskopie. Diese eine Datei verfolgt die Projektarchiv-URLs, die ursprüngliche Revision, Prüfsummen von Dateien, ursprünglichen Text und Zeitstempel von Eigenschaften, Informationen zur Ablaufkoordination und zu Konfliktzuständen, Informationen zur letzten Übergabe (Autor, Revision, Zeitstempel), die Geschichte der lokalen Kopie – praktisch alles, was ein Subversion-Client über eine versionierte (oder noch zu versionierende) Ressource wissen möchte!

Kenner der Verwaltungsverzeichnisse von CVS werden zu diesem Zeitpunkt bemerkt haben, dass die Datei `.svn/entries` von Subversion neben anderen Dingen denselben Zweck verfolgt wie eine Vereinigung der CVS-Dateien `CVS/Entries`, `CVS/Root` und `CVS/Repository`.

Das Format der Datei `.svn/entries` hat sich im Laufe der Zeit geändert. Als ursprüngliche XML-Datei verwendet sie nun ein angepasstes – doch immer noch menschenlesbares – Dateiformat. Obwohl XML eine gute Wahl für die ersten Entwickler von Subversion war, die regelmäßig den Inhalt der Datei (und abhängig davon, Subversions Verhalten) debuggen mussten, ist mittlerweile die Notwendigkeit der Fehlersuche aus der frühen Entwicklungsphase dem Wunsch der Benutzer nach einer flotteren Ausführung gewichen. Ihnen sollte bewusst sein, dass die Arbeitskopie-Bibliothek automatisch Arbeitskopien auf ein neueres Format bringt – sie liest das alte Format und schreibt das neue – was Ihnen einerseits die Bereitstellung einer neuen

Arbeitskopie erspart, andererseits allerdings zu Komplikationen führen kann, falls verschiedene Versionen von Subversion dieselbe Arbeitskopie verwenden wollen.

Unveränderte Kopien und Eigenschafts-Dateien

Wie bereits erwähnt, enthält das Verzeichnis `.svn` auch die unveränderten Versionen „textbasierter“ Dateien. Sie finden diese in `.svn/text-base`. Die Vorteile dieser unveränderten Kopien sind vielfältig – Überprüfung lokaler Änderungen und Vergleiche ohne Netzzugriff, Wiederherstellung veränderter oder verlorengegangener Dateien ohne Netzzugriff, effizientere Übertragung von Änderungen an den Server — jedoch zu dem Preis, dass jede versionierte Datei mindestens zweimal auf der Platte gespeichert wird. Heutzutage scheint das jedoch für die meisten Dateien ein vernachlässigbarer Nachteil zu sein. Jedoch sieht es nicht mehr so schön aus, wenn die Größe Ihrer Dateien anwächst. Es wird daher überlegt, die Anwesenheit der Datei „text-base“ optional zu machen. Ironischerweise jedoch wird das Vorhandensein von „text-base“ mit dem Anwachsen der Größe Ihrer versionierten Dateien immer ausschlaggebender — wer möchte schon eine riesige Datei über das Netz versenden, obwohl nur eine winzige Änderung übergeben werden soll?

Einen ähnlichen Zweck wie die „text-base“-Dateien verfolgen die Eigenschafts-Dateien und deren unveränderte Kopien „prop-base“, die in `.svn/props` bzw. `.svn/prop-base` untergebracht sind. Da selbst Verzeichnisse Eigenschaften haben können, gibt es auch die Dateien `.svn/dir-props` und `.svn/dir-prop-base`.

Benutzung der APIs

Es ist ziemlich unkompliziert, Anwendungen mit den APIs der Bibliotheken von Subversion zu entwickeln. Subversion ist vor allem eine Menge aus C-Bibliotheken mit Header-Dateien (`.h`), die im Verzeichnis `subversion/include` des Quelltextbaums liegen. Diese Header werden in Ihre Systemverzeichnisse (z.B. `/usr/local/include`) kopiert, wenn Sie Subversion aus den Quellen bauen und installieren. Diese Header repräsentieren die Gesamtheit der Funktionen und Typen, die den Benutzern der Subversion-Bibliotheken zugänglich gemacht werden sollen. Die Entwicklergemeinde von Subversion achtet peinlich genau darauf, dass die öffentliche API gut dokumentiert ist – diese Dokumentation finden Sie direkt in den Header-Dateien.

Beim Untersuchen der öffentlichen Header-Dateien wird Ihnen zunächst auffallen, dass die Datentypen und Funktionen von Subversion durch Namensräume geschützt sind. Das heißt, jeder öffentliche Symbolname beginnt mit `svn_`, gefolgt von einem Kürzel der Bibliothek, in der das Symbol definiert ist (etwa `wc`, `client`, `fs` usw.), gefolgt von einem einzelnen Unterstrich (_) und dem Rest des Symbolnamens. Halböffentliche Funktionen (die zwischen Quelldateien einer Bibliothek, jedoch nicht außerhalb davon verwendet werden und innerhalb der Bibliotheksverzeichnisse zu finden sind) weichen von diesem Namensschema ab, indem statt eines einzelnen Unterstrichs nach dem Bibliothekskürzel zwei Unterstriche stehen (__). Private Funktionen in einer Quelldatei haben keinen besonderen Präfix und werden `static` deklariert. Einem Compiler sind diese Konventionen natürlich egal, doch sie helfen, den Gültigkeitsbereich einer gegebenen Funktion oder eines Datentypen deutlich zu machen.

Eine weitere gute Informationsquelle zur Programmierung mit den Subversion-APIs sind die Programmierrichtlinien des Projektes, die Sie unter http://subversion.apache.org/docs/community-guide/ finden können. Dieses Dokument enthält nützliche Informationen, die, obwohl sie für Entwickler und angehende Entwickler von Subversion selbst gedacht sind, genauso für Leute geeignet sind, die mit Subversion als ein Satz von Bibliotheken eines Drittanbieters entwickeln. [2]

Die Bibliothek Apache Portable Runtime

Neben den eigenen Datentypen von Subversion werden Sie viele Verweise auf Datentypen entdecken, die mit `apr_` beginnen – Symbole aus der Bibliothek Apache Portable Runtime (APR). APR ist die Portabilitätsbibliothek von Apache, ursprünglich aus dem Server-Code herausgelöst, als ein Versuch, die betriebssystemspezifischen Teile von den betriebssystemabhängigen Bereichen des Codes zu trennen. Das Ergebnis war eine Bibliothek, die ein generisches API für das Ausführen von Operationen bietet, die sich je nach Betriebssystem mehr oder weniger stark unterscheiden. Obwohl der Apache HTTP-Server offensichtlich die APR-Bibliothek als erster verwendete, erkannten die Subversion-Entwickler sofort die Vorteile, die durch die Benutzung von APR entstehen. Das bedeutet, dass es praktisch keinen betriebssystemspezifischen Code in Subversion gibt. Es bedeutet auch, dass der Subversion-Client sich auf jedem System übersetzen und betreiben lässt, auf dem das auch für den Apache HTTP-Server gilt. Momentan umfasst diese Liste alle Dialekte von Unix, Win32, BeOS, OS/2 und Mac OS X.

Neben der Bereitstellung konsistenter Implementierungen von Systemaufrufen, die sich zwischen Betriebssystemen unterscheiden,[3] bietet APR Subversion unmittelbaren Zugriff auf viele maßgeschneiderte Datentypen, wie etwa dynamische Arrays und Hashtabellen. Subversion macht von diesen Typen regen Gebrauch. Der vielleicht am meisten verbreitete APR-

[2] Schließlich verwendet auch Subversion die APIs von Subversion.
[3] Subversion verwendet so weit wie möglich ANSI-Systemaufrufe und Datentypen.

Datentyp, der sich in beinahe jedem Subversion-API-Prototypen wiederfindet, ist `apr_pool_t` — der APR-Speicherpool. Subversion verwendet Pools intern zum Zuteilen all seines Speichers (außer wenn eine externe Bibliothek eine unterschiedliche Speicherverwaltung für die über ihre API ausgetauschten Daten voraussetzt),[4] und obwohl jemand, der mit den Subversion-APIs programmiert nicht gezwungen ist, dasselbe zu tun, *muss* er Pools für die API-Funktionen zur Verfügung stellen, die sie benötigen. Das heißt, dass Benutzer der Subversion-API auch gegen die APR linken, `apr_initialize()` zum Initialisieren des APR-Subsystems aufrufen und Pools zur Verwendung mit Subversion-API-Aufrufen erzeugen sowie verwalten müssen, typischerweise unter Benutzung von `svn_pool_create()`, `svn_pool_clear()` und `svn_pool_destroy()`.

Programmierung mit Speicher-Pools

Fast jeder C-Programmierer hat irgendwann mal gestöhnt, wenn es um die abschreckende Aufgabe der Speicherverwaltung ging. Genügend Speicher anzufordern, über den Speicher Buch zu führen, den Speicher nach Benutzung wieder freizugeben – diese Aufgaben können ziemlich kompliziert sein. Und wenn es nicht richtig gemacht wird, kann es natürlich dazu führen, dass sich das Programm oder, noch schlimmer, der Rechner aufhängt.

Andererseits nehmen Ihnen höhere Sprachen die Aufgaben der Speicherverwaltung vollständig ab oder überlassen sie Ihnen nur, falls Sie besonders starke Programmoptimierungen vornehmen wollen. Sprachen wie Java und Python verwenden *Garbage Collection*, indem sie Speicher bei Bedarf zuweisen und diesen Speicher automatisch wieder freigeben, wenn er nicht mehr benötigt wird.

APR bietet einen Mittelweg namens *pool-basierte Speicherverwaltung*. Es ermöglicht dem Entwickler, den Speicherbedarf auf einer niedrigeren Ebene zu kontrollieren – pro Speicherbrocken (oder „Pool“) statt pro zugeteilten Objekt. Statt `malloc()` und Konsorten zu verwenden, um ausreichend Speicher für ein Objekt anzufordern, fordern Sie APR auf, Speicher aus einem Pool zuzuteilen. Wenn Sie mit der Benutzung der im Pool erzeugten Objekte fertig sind, löschen Sie den gesamten Pool und geben somit den Speicher *aller* Objekte frei, die Sie daraus erzeugt haben. Anstatt sich also um einzelne freizugebende Objekte kümmern zu müssen, betrachtet Ihr Programm nur die allgemeine Lebenszeit dieser Objekte und erzeugt diese Objekte in einem Pool, dessen Lebenszeit (zwischen seiner Erzeugung und seiner Zerstörung) dem Bedarf der Objekte entspricht.

Funktionen und Batons

Um ein „datenstromähnliches“ (asynchrones) Verhalten zu ermöglichen und Benutzern der Subversion C-Programmierschnittstelle Anknüpfpunkte für eine flexible Informationsverarbeitung zur Verfügung zu stellen, akzeptieren viele Funktionen der Schnittstelle Parameterpaare: ein Zeiger auf eine Rückruffunktion und einen Zeiger auf einen Speicherbereich, genannt *Baton* (Staffelstab), der Kontextinformationen für die Rückruffunktion beinhaltet. Bei Batons handelt es sich üblicherweise um C-Strukturen mit zusätzlichen Informationen, die die Rückruffungktion benötigt, ihr jedoch nicht direkt durch die steuernde Schnittstellenfunktion übergeben werden.

URL- und Pfadanforderungen

Da das Betreiben entfernte Versionskontrolle der Grund für das Vorhandensein von Subversion ist, ergibt es einen Sinn, dass der Internationalisierung (i18n) etwas Aufmerksamkeit gewidmet wurde. Schließlich kann „entfernt“ sowohl „am anderen Ende vom Büro“ als auch „am anderen Ende der Welt“ bedeuten. Um das zu ermöglichen, erwarten alle öffentlichen Schnittstellen von Subversion, die Pfadargumente annehmen, dass diese Pfade im kanonischen Format vorliegen – was am besten gelingt, wenn sie durch die Funktion `svn_path_canonicalize()` geschickt werden – und in UTF-8 kodiert sind. Das bedeutet beispielsweise, dass ein neuer Client, der die Schnittstelle `libsvn_client` benutzt, zunächst Pfade aus der sprachabhängigen Kodierung nach UTF-8 konvertieren muss, bevor sie an die Subversion-Bibliotheken übergeben werden und anschließend umgekehrt etwaige Ergebnispfade von Subversion zurück in die sprachabhängige Kodierung umgewandelt werden müssen, bevor die Pfade für Zwecke verwendet werden, die nichts mit Subversion zu tun haben. Glücklicherweise stellt Subversion eine Reihe von Funktionen zur Verfügung (siehe `subversion/include/svn_utf.h`) die jedes Programm zum Konvertieren benutzen kann.

Ferner ist es für die Subversion APIs erforderlich, dass alle URL-Parameter richtig URI-kodiert sind. So sollten Sie statt `file:///home/username/My File.txt` als URL einer Datei namens `My File.txt` `file:///home/username/My%20File.txt` übergeben. Auch hierfür stellt Subversion Hilfsfunktionen für Ihre Anwendung zur Verfügung – `svn_path_uri_encode()` und `svn_path_uri_decode()` zum Kodieren bzw. Dekodieren von URIs.

[4] Neon und Berkeley DB sind Beispiele solcher Bibliotheken.

Verwendung anderer Sprachen als C und C++

Falls Sie daran interessiert sein sollten, die Subversion-Bibliotheken in Verbindung mit etwas anderem als ein C-Programm zu benutzen – etwa ein Python- oder ein Perl-Script – bietet Subversion etwas Unterstützung über den *Simplified Wrapper and Interface Generator* (SWIG). Die SWIG-Bindungen für Subversion liegen in `subversion/bindings/swig`. Sie reifen zwar noch, können aber verwendet werden. Diese Bindungen erlauben Ihnen, Subversion-API-Funktionen indirekt aufzurufen, indem eine Zwischenschicht verwendet wird, die die Datentypen Ihrer Skriptsprache in die Datentypen umwandeln, die von Subversions C-Bibliotheken benötigt werden.

Es wurden bedeutende Anstrengungen unternommen, um funktionierende SWIG-erzeugte Bindungen für Python, Perl und Ruby zur Verfügung zu stellen. Bis zu einem gewissen Grad kann die Arbeit zur Vorbereitung der SWIG-Schnittstellen für diese Sprachen wiederverwendet werden, wenn es darum geht, Bindungen für andere von SWIG unterstützte Sprachen zu erzeugen (unter anderem Versionen von C#, Guile, Java, MzScheme, OCaml, PHP und Tcl). Jedoch ist etwas zusätzliche Programmierarbeit für komplizierte APIs erforderlich, bei deren Übersetzung SWIG ein wenig Hilfe benötigt. Weitergehende Informationen zu SWIG finden Sie auf der Projektseite unter http://www.swig.org/.

Subversion verfügt ebenfalls über Sprachbindungen für Java. Die javahl-Bindungen (zu finden in `subversion/bindings/java` im Subversion-Quelltext-Baum) sind nicht SWIG-basiert sondern ein Gemisch aus Java und handgeschriebenem JNI. Javahl deckt die meisten client-seitigen Subversion-APIs ab und zielt besonders auf Implementierer Java-basierter Subversion-Clients und IDE-Integrationen ab.

Zwar gilt den Sprachbindungen von Subversion seitens der Entwickler nicht dieselbe Aufmerksamkeit wie den Subversion-Kernmodulen, doch sie sind durchaus bereit für den Einsatz. Eine Reihe von Skripten und Anwendungen, alternative graphische Subversion-Clients und andere Werkzeuge von Drittanbietern machen heute schon erfolgreich Gebrauch von den Subversion-Sprachbindungen, um ihre Subversion-Integration zustande zu bringen.

An dieser Stelle ist es erwähnenswert, dass es auch andere Optionen gibt, um Subversion-Schnittstellen in anderen Sprachen zu verwenden: alternative Subversion-Bindungen, die gar nicht aus der Subversion-Entwicklergemeinde stammen. Es gibt davon ein paar beliebte, die besonders erwähnenswert sind. Zunächst seien die PySVN-Bindungen von Barry Scott genannt (http://pysvn.tigris.org/), die eine beliebte Alternative zur Python-Bindung darstellen. PySVN rühmt sich, eine Schnittstelle zu liefern, die Python-typischer ist als die eher C-ähnlichen APIs der Subversion-eigenen Python-Bindungen. Und falls Sie eine reine Java-Implementierung suchen, wehen Sie sich SVNKit (http://svnkit.com/) an, das eine vollständige Reimplementierung von Subversion in Java ist.

SVNKit verglichen mit javahl

Im Jahr 2005 kündigte eine kleine Firma namens TMate die Version 1.0.0 von JavaSVN an – eine reine Java-Implementierung von Subversion. Seitdem wurde das Projekt in SVNKit umbenannt (verfügbar unter http://svnkit.com/) und wurde erfolgreich als Anbieter von Subversion-Funktionen für zahlreiche Subversion-Clients, IDE-Integrationen und andere Werkzeuge von Drittanbietern eingesetzt.

Die SVNKit-Bibliothek ist insofern interessant, als dass sie, anders als die javahl-Bibliothek, nicht bloß eine Umhüllung der Subversion-Kernbibliotheken darstellt. Tatsächlich teilt sie sich überhaupt keinen Code mit Subversion. Doch obwohl es einfach ist, SVNKit mit javahl zu verwechseln und noch einfacher, gar nicht wahrzunehmen, welche dieser Bibliotheken benutzt wird, sollten sich alle bewusst sein, dass SVNKit sich in einigen entscheidenden Punkten von javahl unterscheidet. Erstens, obwohl SVNKit ebenso wie Subversion als quelloffene Software entwickelt wird, ist die Lizenz von SVNKit restriktiver als die von Subversion.[5] Da letztendlich SVNKit darauf abzielt, eine reine Java Subversion-Bibliothek zu sein, ist SVNKit beim Klonen von Teilen Subversions eingeschränkt, da es sonst den Anschluss an die Versionen von Subversion verlieren würde. Das ist bereits einmal passiert – SVNKit kann nicht über das Protokoll `file://` auf BDB-basierte Subversion-Projektarchive zugreifen, da es keine reine Java-Implementierung von Berkeley DB gibt, deren Dateiformat kompatibel zur ursprünglichen Implementierung dieser Bibliothek ist.

Abgesehen davon, hat sich SVNKit einen guten Ruf in Sachen Zuverlässigkeit erarbeitet. Und im Angesicht von Programmierfehlern ist eine Java-Lösung robuster – ein Fehler in SVNKit könnte eine behandelbare Ausnahme provozieren, ein Fehler in den Kernbibliotheken von Subversion jedoch könnte beim Zugriff über javahl Ihre gesamte Java-Laufzeitumgebung zum Absturz bringen. Wägen Sie also die Kosten sorgfältig ab, falls Sie sich für eine Java-basierte Subversion-Implementierung entscheiden sollten.

[5] Die Weitergabe in jeder Form muss die Information beigefügt sein, wie der vollständige Quelltext der Software beschafft werden kann, der SVNKit verwendet, sowie der begleitenden Software, die wiederum Software verwendet, die SVNKit verwendet. Zu Details, siehe http://svnkit.com/license.html.

Beispielcode

Beispiel 8.1, „Verwendung der Projektarchiv-Schicht“ enthält einen Codeabschnitt (in C), der einige der erörterten Konzepte veranschaulicht. Er verwendet sowohl die Projektarchiv- als auch die Dateisystemschnittstelle (was anhand der Präfixe `svn_repos_` bzw. `svn_fs_` der Funktionsnamen erkennbar ist), um eine neue Revision zu erzeugen, in der ein Verzeichnis hinzugefügt wird. Sie können die Verwendung des APR-Pools erkennen, der zur Speicherzuteilung herumgereicht wird. Der Code verdeutlicht auch eine etwas undurchsichtige Angelegenheit bezüglich der Fehlerbehandlung von Subversion – alle Fehler von Subversion müssen explizit behandelt werden, um Speicherlöcher zu verhindern (und vereinzelt Programmabstürze).

Beispiel 8.1. Verwendung der Projektarchiv-Schicht

```
/* Umwandlung eines Subversion-Fehlers in einen einfachen Boole'schen
 * Fehlerwert.
 *
 * NOTE:  Subversion-Fehler müssen zurückgesetzt werden (mit
 *        svn_error_clear()), da sie aus dem globalen Pool zugeteilt
 *        werden, ansonsten treten Speicherlöcher auf.
 */
#define INT_ERR(expr)                           \
  do {                                          \
    svn_error_t *__temperr = (expr);            \
    if (__temperr)                              \
      {                                         \
        svn_error_clear(__temperr);             \
        return 1;                               \
      }                                         \
    return 0;                                   \
  } while (0)


/* Ein neues Verzeichnis im Pfad NEW_DIRECTORY des
 * Subversion-Projektarchivs bei REPOS_PATH erzeugen. Sämtliche
 * Speicherzuteilungen in Pool durchführen. Diese Funktion erzeugt
 * eine neue Revision, damit NEW_DIRECTORY hinzugefügt werden kann. Im
 * Erfolgsfall Null, sonst einen Wert ungleich Null zurückgeben.
 */
static int
make_new_directory(const char *repos_path,
                   const char *new_directory,
                   apr_pool_t *pool)
{
  svn_error_t *err;
  svn_repos_t *repos;
  svn_fs_t *fs;
  svn_revnum_t youngest_rev;
  svn_fs_txn_t *txn;
  svn_fs_root_t *txn_root;
  const char *conflict_str;


  /* Projektarchiv bei REPOS_PATH öffnen.
   */
  INT_ERR(svn_repos_open(&repos, repos_path, pool));


  /* Zeiger auf das Dateisystemobjekt in REPOS holen.
   */
  fs = svn_repos_fs(repos);


  /* Anfrage beim Dateisystem nach der aktuell letzten existierenden
   * Revision.
   */
```

```
  INT_ERR(svn_fs_youngest_rev(&youngest_rev, fs, pool));


  /* Starten einer neuen Transaktion basierend auf YOUNGEST_REV. Die
   * Wahrscheinlichkeit einer später wegen Konflikte abgelehnten
   * Übergabe sinkt, wenn wir stets versuchen, unsere Änderungen auf
   * der letzten Momentaufnahme des Dateisystembaums zu machen.
   */
  INT_ERR(svn_repos_fs_begin_txn_for_commit2(&txn, repos, youngest_rev,
                                             apr_hash_make(pool), pool));


  /* Nach dem Start der Transaktion wird ein Wurzelobjekt geholt, das
   * diese Transaktion repräsentiert.
   */
  INT_ERR(svn_fs_txn_root(&txn_root, txn, pool));


  /* Das neue Verzeichnis unter der Transaktionswurzel im Pfad
   * NEW_DIRECTORY anlegen.
   */
  INT_ERR(svn_fs_make_dir(txn_root, new_directory, pool));


  /* Die Transaktion übergeben, indem eine neue Revision des
   * Dateisystems erzeugt wird, die unseren hinzugefügten
   * Verzeichnispfad enthält.
   */
  err = svn_repos_fs_commit_txn(&conflict_str, repos,
                                &youngest_rev, txn, pool);
  if (! err)
    {

      /* Kein Fehler? Ausgezeichnet! Eine kurze Erfolgsnachricht
       * ausgeben.
       */

      printf("Verzeichnis '%s' ist erfolgreich als neue Revision "
             "'%ld' hinzugefügt worden.\n", new_directory, youngest_rev);
    }
  else if (err->apr_err == SVN_ERR_FS_CONFLICT)
    {

      /* Oh-ha. Die Übergabe schlug wegen eines Konfliktes fehl
       * (jemand anderes scheint im gleichen Bereich des Dateisystems
       * Änderungen gemacht zu haben, das wir ändern wollten). Eine
       * Fehlermeldung ausgeben.
       */

      printf("Ein Konflikt trat im Pfad '%s' auf, als versucht wurde, "
             "das Verzeichnis '%s' dem Projektarchiv bei '%s' hinzuzufügen.\n",
             conflict_str, new_directory, repos_path);
    }
  else
    {

      /* Ein anderer Fehler ist aufgetreten. Eine Fehlermeldung
       * ausgeben.
       */

      printf("Beim Versuch, das Verzeichnis '%s' dem Projektarchiv bei "
             "'%s' hinzuzufügen, trat ein Fehler auf.\n",
             new_directory, repos_path);
    }

  INT_ERR(err);
}
```

Beachten Sie, dass der Code in Beispiel 8.1, „Verwendung der Projektarchiv-Schicht“ die Transaktion ebenso einfach mit `svn_fs_commit_txn()` hätte übergeben können. Allerdings weiß die Dateisystem-API nichts über den Hook-Mechanismus der Projektarchiv-Bibliothek. Falls Sie möchten, dass Ihr Subversion-Projektarchiv bei jeder Transaktionsübergabe eine Nicht-Subversion-Aufgabe ausführt (z.B. eine E-Mail, die alle Änderungen in dieser Transaktion beschreibt, an Ihre Entwickler-Mailingliste senden), müssen Sie diejenige Version dieser Funktion verwenden, die durch die Bibliothek `libsvn_repos` umschlossen ist und die Auslösung von Hooks beinhaltet – in diesem Fall `svn_repos_fs_commit_txn()`. (Weitere Informationen zu Subversions Projektarchiv-Hooks in „Erstellen von Projektarchiv-Hooks“.)

Lassen Sie uns nun die Sprachen wechseln. Beispiel 8.2, „Verwendung der Projektarchiv-Schicht mit Python“ ist ein Beispielprogramm, das die SWIG-Python-Bindungen von Subversion verwendet, um rekursiv die jüngste Projektarchiv-Revision zu traversieren und dabei die zahlreichen Versionspfade auszugeben.

Beispiel 8.2. Verwendung der Projektarchiv-Schicht mit Python

```
#!/usr/bin/python


"""Durchwandern eines Projektarchivs mit Ausgabe der Projektarchiv-Pfadnamen."""

import sys
import os.path
import svn.fs, svn.core, svn.repos

def crawl_filesystem_dir(root, directory):

    """Rekursives durchwandern von DIRECTORY unterhalb von ROOT im
    Dateisystem und eine Liste aller Pfade unterhalb von DIRECTORY
    zurückgeben."""


    # Ausgabe dieses Pfadnamens.
    print directory + "/"


    # Verzeichniseinträge für DIRECTORY holen.
    entries = svn.fs.svn_fs_dir_entries(root, directory)


    # Einträge abarbeiten.
    names = entries.keys()
    for name in names:

        # Den vollen Pfadnamen des Eintrags berechnen.
        full_path = directory + '/' + name


        # Falls der Eintrag ein Verzeichnis ist, rekursiv bearbeiten.
        # Die Rekursion gibt eine Liste mit dem Eintrag und all seiner
        # Kinder zurück, die der aktuellen Pfadliste hinzugefügt wird.
        if svn.fs.svn_fs_is_dir(root, full_path):
            crawl_filesystem_dir(root, full_path)
        else:

            # Sonst handelt es sich um eine Datei, also den Pfad
            # ausgeben.
            print full_path

def crawl_youngest(repos_path):

    """Öffnen des Projektarchivs bei REPOS_PATH, und rekursives
    Durchwandern seiner jüngsten Revision."""
```

```
    # Öffnen des Projektarchivs bei REPOS_PATH, und holen einer Referenz
    # auf sein versioniertes Dateisystem.
    repos_obj = svn.repos.svn_repos_open(repos_path)
    fs_obj = svn.repos.svn_repos_fs(repos_obj)


    # Die aktuell jüngste Revision abfragen.
    youngest_rev = svn.fs.svn_fs_youngest_rev(fs_obj)


    # Ein Wurzelobjekt öffnen, das die jüngste (HEAD) Revision
    # repräsentiert.
    root_obj = svn.fs.svn_fs_revision_root(fs_obj, youngest_rev)


    # Rekursiv durchwandern.
    crawl_filesystem_dir(root_obj, "")

if __name__ == "__main__":

    # Überprüfung auf korrekten Aufruf.
    if len(sys.argv) != 2:
        sys.stderr.write("Usage: %s REPOS_PATH\n"
                         % (os.path.basename(sys.argv[0])))
        sys.exit(1)


    # Den Projektarchiv-Pfad kanonisieren.
    repos_path = svn.core.svn_path_canonicalize(sys.argv[1])


    # Eigentliche Arbeit machen.
    crawl_youngest(repos_path)
```

Das gleiche Programm in C hätte sich noch um das APR-Speicher-Pool-System kümmern müssen. Python jedoch verwaltet den Speicher automatisch, und die Python-Bindungen von Subversion berücksichtigen diese Konvention. In C würden Sie mit angepassten Datentypen (etwa denen aus der APR-Bibliothek) arbeiten, um den Hash mit Einträgen und die Liste der Pfade zu repräsentieren, doch Python verfügt über Hashes („Dictionarys“ genannt) und Listen als eingebaute Datentypen und bietet eine reichhaltige Sammlung aus Funktionen, um mit diesen Datentypen umzugehen. Also übernimmt SWIG (mithilfe einiger Anpassungen in der Sprachbindungsschicht von Subversion) die Abbildung dieser speziellen Datentypen auf die spezifischen Datentypen der Zielsprache. Dies bietet Benutzern dieser Sprache eine intuitivere Schnittstelle.

Auch für Operationen in der Arbeitskopie können die Python-Bindungen von Subversion verwendet werden. Im vorangegangenen Abschnitt dieses Kapitels erwähnten wir die Schnittstelle `libsvn_client` und ihren Zweck zur Vereinfachung der Erstellung eines Subversion-Clients. Beispiel 8.3, „Status in Python“ ist ein kurzes Beispiel wie auf diese Bibliothek über die SWIG-Python-Bindungen zugegriffen werden kann, um eine abgespeckte Version des Befehls **svn status** nachzubauen.

Beispiel 8.3. Status in Python

```
#!/usr/bin/env python


"""Durchwandern eines Arbeitskopieverzeichnisses mit Ausgabe von
Statusinformation."""

import sys
import os.path
import getopt
import svn.core, svn.client, svn.wc

def generate_status_code(status):
```

```
    """Übersetzen eines Stauswerts in einen Ein-Zeichen-Statuscode,
    wobei dieselbe Logik wie beim Subversion-Kommandozeilen-Client
    verwendet wird."""
    code_map = { svn.wc.svn_wc_status_none        : ' ',
                 svn.wc.svn_wc_status_normal      : ' ',
                 svn.wc.svn_wc_status_added       : 'A',
                 svn.wc.svn_wc_status_missing     : '!',
                 svn.wc.svn_wc_status_incomplete  : '!',
                 svn.wc.svn_wc_status_deleted     : 'D',
                 svn.wc.svn_wc_status_replaced    : 'R',
                 svn.wc.svn_wc_status_modified    : 'M',
                 svn.wc.svn_wc_status_conflicted  : 'C',
                 svn.wc.svn_wc_status_obstructed  : '~',
                 svn.wc.svn_wc_status_ignored     : 'I',
                 svn.wc.svn_wc_status_external    : 'X',
                 svn.wc.svn_wc_status_unversioned : '?',
               }
    return code_map.get(status, '?')

def do_status(wc_path, verbose, prefix):

    # Einen "Staffelstab" für den Client-Kontext erzeugen.
    ctx = svn.client.svn_client_create_context()

    def _status_callback(path, status):

        """Eine Rückruffunktion für svn_client_status."""


        # Ausgeben des Pfades, ohne den Teil, der sich mit der Wurzel
        # des zu durchlaufenden Baums überlappt
        text_status = generate_status_code(status.text_status)
        prop_status = generate_status_code(status.prop_status)
        prefix_text = ''
        if prefix is not None:
            prefix_text = prefix + " "
        print '%s%s%s  %s' % (prefix_text, text_status, prop_status, path)


    # Das Durchlaufen starten, _status_callback() als Rückruffunktion
    # verwenden.
    revision = svn.core.svn_opt_revision_t()
    revision.type = svn.core.svn_opt_revision_head
    svn.client.svn_client_status2(wc_path, revision, _status_callback,
                                  svn.core.svn_depth_infinity, verbose,
                                  0, 0, 1, ctx)

def usage_and_exit(errorcode):

    """Ausgabe des Verwendungshinweises und beenden mit ERRORCODE."""
    stream = errorcode and sys.stderr or sys.stdout

    stream.write("""Verwendung: %s OPTIONWN AK-PATH
Optionen:
  --help, -h    : Diesen Hinweis anzeigen
  --prefix ARG  : ARG gefolgt von einem Leerzeichen vor jeder Zeile ausgeben
  --verbose, -v : Zeige alle Status, auch uninteressante
""" % (os.path.basename(sys.argv[0])))
    sys.exit(errorcode)

if __name__ == '__main__':

    # Kommandozeilenoptionen parsen.
    try:
        opts, args = getopt.getopt(sys.argv[1:], "hv",
                                   ["help", "prefix=", "verbose"])
    except getopt.GetoptError:
        usage_and_exit(1)
```

```
verbose = 0
prefix = None
for opt, arg in opts:
    if opt in ("-h", "--help"):
        usage_and_exit(0)
    if opt in ("--prefix"):
        prefix = arg
    if opt in ("-v", "--verbose"):
        verbose = 1
if len(args) != 1:
    usage_and_exit(2)


# Projektarchivpfad kanonisieren.
wc_path = svn.core.svn_path_canonicalize(args[0])


# Eigentliche Arbeit machen.
try:
    do_status(wc_path, verbose, prefix)
except svn.core.SubversionException, e:

    sys.stderr.write("Fehler (%d): %s\n" % (e.apr_err, e.message))
    sys.exit(1)
```

Wie in Beispiel 8.2, „Verwendung der Projektarchiv-Schicht mit Python“ verwendet auch dieses Programm keine Pools und benutzt meist normale Python-Datentypen.

Lassen Sie Pfade, die von Anwendern mitgegeben werden, durch `svn_path_canonicalize()` filtern, bevor sie an andere API-Funktionen weitergeleitet werden. Ein *unterlassen* kann dazu führen, dass Annahmen der darunter liegenden C-Bibliotheken nicht mehr zutreffen, was wiederum einen ziemlich plötzlichen und ungezwungenen Programmabsturz bedeutet.

Von besonderem Interesse für Anwender der Python-Variante von Subversions Programmierschnittstelle ist die Implementierung von Rückruffunktionen. Wie bereits erwähnt wurde, macht die C-Programmierschnittstelle von Subversion regen Gebrauch vom Rückruffunktion-Baton-Paradigma. Schnittstellenfunktionen, die in C ein Funktion-Baton-Paar akzeptieren, erlauben in Python nur einen Parameter mit einer Rückruffunktion. Wie soll der Aufrufer dann beliebige Kontextinformationen an die Rückruffunktion übergeben? In Python wird das durch Ausnutzung der Regeln zum Gültigkeitsbereich und der Standard-Argumentwerte erreicht. Sie können sich die Umsetzung in Beispiel 8.3, „Status in Python“ ansehen. Der Funktion `svn_client_status2()` wird eine Rückruffunktion (`_status_callback()`) aber kein Baton mitgegeben; `_status_callback()` hat Zugriff auf den vom Anwender zur Verfügung gestellten Präfix, da diese Variable automatisch in den Gültigkeitsbereich der Funktion fällt.

Zusammenfassung

Eine der größten Vorteile von Subversion bekommen Sie nicht über den Kommandozeilen-Client oder sonstige Werkzeuge. Es ist die Tatsache, dass Subversion modular entworfen wurde und eine stabile öffentliche API bereitstellt, so dass andere – etwa Sie selbst – eigene Software erstellen können, die die Kernfunktion von Subversion ansteuert.

In diesem Kapitel haben wir uns die Architektur von Subversion etwas näher angesehen, indem wir seine logischen Schichten untersucht und die öffentliche API beschrieben haben; die API, die auch von den Subversion-eigenen Schichten verwendet wird, um miteinander zu kommunizieren. Viele Entwickler haben interessante Verwendungen für die Subversion-API gefunden, von einfachen Projektarchiv-Hook-Skripten über Integrationen zwischen Subversion und einer anderen Anwendung bis zu vollständig unterschiedlichen Versionskontrollsystemen. Womit wollen *Sie* es versuchen?

Kapitel 9. Die vollständige Subversion Referenz

Dieses Kapitel soll als vollständige Referenz für die Verwendung von Subversion dienen. Es umfasst Befehlsübersichten und Beispiele für alle Kommandozeilenwerkzeuge, die zur Grundausstattung des Subversion-Paketes mitgeliefert werden, Konfigurationsinformationen für die Module des Subversion-Servers und andere Informationen, die sich für ein Referenzformat anbieten.

svn – Subversion-Kommandozeilen-Client

svn ist der offizielle Kommandozeilen-Client von Subversion. Er bietet eine nicht geringe Anzahl von Unterbefehlen und Optionen. Unterbefehle und weitere Argumente, die keine Optionen sind, müssen beim Aufruf von **svn** auf der Kommandozeile in einer festgelegten Reihenfolge angegeben werden. Optionen dagegen können überall in der Kommandozeile auftauchen (natürlich nach dem Programm-Namen), wobei deren Reihenfolge im Allgemeinen unerheblich ist. Alle der folgenden Beispiele zeigen gültige Aufrufe von **svn status** und werden exakt auf dieselbe Art und Weise interpretiert:

```
$ svn -vq status myfile
$ svn status -v -q myfile
$ svn -q status -v myfile
$ svn status -vq myfile
$ svn status myfile -qv
```

svn-Optionen

Obwohl Subversion verschiedene Optionen für seine Unterbefehle hat, existieren alle Optionen in einem einzigen Namensraum – dass heißt, dass alle Optionen dasselbe bedeuten, egal mit welchem Unterbefehl sie angegeben werden. Beispielsweise bedeutet `--verbose` (`-v`) stets „ausführliche Ausgabe“, unabhängig vom Unterbefehl, dem diese Option mitgegeben wird.

Der Kommandozeilen-Client **svn** bricht normalerweise sofort mit einer Fehlermeldung ab, falls Sie ihm eine Option mitgeben, die nicht auf den angegebenen Unterbefehl anwendbar ist. Seit Subversion 1.5 jedoch werden viele Optionen, die auf alle – oder fast alle – Unterbefehle anwendbar sind, von allen Unterbefehlen akzeptiert, auch wenn sie für einige keine Auswirkungen haben. In der Bedienhilfe des Kommandozeilen-Clients werden diese Optionen als globale Optionen zusammengefasst. Dies wurde zur Unterstützung derjenigen gemacht, die Skripte schreiben, die den Kommandozeilen-Client umhüllen. Die globalen Optionen sind die folgenden:

`--config-dir` *`DIR`*
: Weist Subversion an, Konfigurationsinformationen aus dem angegebenen Verzeichnis zu lesen, statt aus dem Standardverzeichnis (`.subversion` im Heimatverzeichnis des Benutzers).

`--config-option` *`FILE`*:*`SECTION`*:*`OPTION`*=[*`VALUE`*]
: Setzt den Wert einer Laufzeitkonfigurationsoption für die Dauer eines Befehls. *`FILE`* und *`SECTION`* bestimmen jeweils die Konfigurationsdatei (entweder `config` oder `servers`) bzw. den darin befindlichen Abschnitt, in dem sich die Option befindet, die Sie ändern möchten. *`OPTION`* ist natürlich die Option selbst, und *`VALUE`* der Wert (sofern vorhanden), den Sie der Option zuweisen möchten. Wenn Sie beispielsweise vorübergehend das automatische Setzen von Eigenschaften verhindern möchten, verwenden Sie `--config-option=config:miscellany:enable-auto-props=no`. Sie können diese Option mehrfach verwenden, um gleichzeitig unterschiedliche Optionswerte zu ändern.

`--no-auth-cache`
: Verhindert die Zwischenspeicherung von Authentisierungsinformationen (z.B. Anwendername und Passwort) in den Laufzeitkonfigurationsverzeichnissen von Subversion.

`--non-interactive`
Unterbindet sämtliche Nachfragen. Beispiele für solche Nachfragen sind Aufforderungen zur Eingabe von Zugangsdaten und Entscheidungen zur Konfliktauflösung. Dies ist nützlich, falls Sie Subversion innerhalb eines automatisierten Skriptes aufrufen und somit ein Abbruch mit Fehlermeldung angebrachter ist als eine Nachfrage.

`--password` *PASSWD*
Gibt das Passwort zur Authentisierung gegenüber einem Subversion-Server an. Falls es nicht mitgegeben wird oder falsch ist, fragt Subversion bei Bedarf nach.

`--trust-server-cert`
Zusammen mit `--non-interactive`, wird Subversion aufgefordert, ohne Nachfrage beim Anwender von unbekannten Zertifizierungsstellen herausgegebene SSL Server-Zertifikate zu akzeptieren. Aus Sicherheitsgründen sollten Sie diese Option nur in dem Fall verwenden, wenn sichergestellt ist, dass die Integrität des Servers und die Netzverbindung zu Ihrem Client vertrauernswürdig ist.

`--username` *NAME*
Gibt den Anwendernamen zur Authentisierung gegenüber einem Subversion-Server an. Falls er nicht mitgegeben wird oder falsch ist, fragt Subversion bei Bedarf nach.

Der Rest der Optionen ist nur auf eine Teilmenge der Unterbefehle anwendbar und wird auch nur von einer Teilmenge der Unterbefehle akzeptiert. Diese sind:

`--accept` *ACTION*
Gibt eine Aktion zur automatischen Konfliktauflösung an. Mögliche Aktionen sind `postpone`, `base`, `mine-full`, `theirs-full`, `edit` und `launch`.

`--auto-props`
Ermöglicht Auto-Props, wobei die Anweisung `enable-auto-props` in der Datei `config` nichtig gemacht wird.

`--change` (`-c`) *ARG*
Wird verwendet, um sich auf eine bestimmte „Änderung“ (also Revision) zu beziehen. Diese Option ist syntaktisch schöner als „-r ARG-1:ARG“.

`--changelist` *ARG*
Weist Subversion an, nur auf Elementen der Änderungsliste *ARG* zu arbeiten. Sie können diese Option mehrfach angeben, um Mengen aus Änderungslisten anzugeben.

`--cl` *ARG*
Ein Alias für die Option `--changelist`.

`--depth` *ARG*
Weist Subversion an, den Bereich einer Operation auf eine bestimmte Tiefe des Baums zu beschränken. *ARG* kann einen der Werte `empty` (nur das Ziel selbst), `files` (das Ziel und alle unmittelbaren Kind-Dateien), `immediates` (das Ziel und alle seine unmittelbaren Kinder) oder `infinity` (das Ziel und alle seine Nachfolger, vollrekursiv) annehmen.

`--diff-cmd` *CMD*
Dient der Angabe eines externen Programms zur Anzeige von Unterschieden zwischen Dateien. Wird **svn diff** ohne diese Option aufgerufen, verwendet es den eingebauten diff-Algorithmus von Subversion, der standardmäßig eine Ausgabe im unified-diff-Format erzeugt. Wenn Sie ein externes Programm verwenden wollen, benutzen Sie `--diff-cmd`. Sie können mit der Option `--extensions` (`-x`) Optionen an das externe Programm übergeben (mehr dazu später in diesem Kapitel).

`--diff3-cmd` *CMD*
Gibt ein externes Programm zum Zusammenführen von Dateien an.

`--dry-run`
Simuliert alle Stufen einer Befehlsausführung, nimmt jedoch keine Änderungen vor – weder auf der Platte noch im Projektarchiv.

`--editor-cmd` *CMD*
Gibt ein externes Programm zum Editieren einer Protokollnachricht oder eines Eigenschafts-Wertes an. Zum Angeben

eines Standardeditors siehe den Abschnitt `editor-cmd` in „Config“.

`--encoding` *`ENC`*
Teilt Subversion mit, dass Ihre Protokollnachricht mit dem angegebenen Zeichensatz kodiert ist. Standard ist die sprachabhängige Locale Ihres Betriebssystems; Sie sollten die Kodierung angeben, wenn sie vom Standard abweicht.

`--extensions` (`-x`) *`ARG`*
Bestimmt Anpassungen für die Berechnung von Dateiunterschieden durch Subversion. Gültige Erweiterungen beinhalten:

`--ignore-space-change` (`-b`)
Änderungen bei der Anzahl an Leerraumzeichen ignorieren.

`--ignore-all-space` (`-w`)
Sämtlichen Leerraum ignorieren.

`--ignore-eol-style`
Änderungen bei der Art der Zeilenendungen ignorieren.

`--show-c-function` (`-p`)
C-Funktionsnamen in der Ausgabe von diff anzeigen.

`--unified` (`-u`)
Drei Zeilen unifizierten diff-Kontext anzeigen.

Der Standardwert von *`ARG`* ist `-u`. Falls Sie mehrere Argumente übergeben möchten, müssen Sie alle in Anführungszeichen setzen.

Beachten Sie, dass der Wert der Option `--extension` (`-x`) nicht auf die oben angeführten Optionen beschränkt ist, sofern Subversion für ein externes diff-Programm konfiguriert worden ist, sondern *irgendwelche* zusätzlichen Argumente beinhalten kann, den Subversion an dieses Programm weiterreichen soll.

`--file` (`-F`) *`FILENAME`*
Verwendet den Inhalt der benannten Datei für den angegebenen Befehl; verschiedene Unterbefehle interpretieren den Inhalt auf unterschiedliche Weise. Beispielsweise verwendet **svn commit** den Inhalt als Protokollnachricht, während **svn propset** ihn als Eigenschafts-Wert interpretiert.

`--force`
Erzwingt die Ausführung eines bestimmten Befehls oder einer Operation. Normalerweise hindert Sie Subversion daran, bestimmte Operationen auszuführen, doch können Sie Subversion mit dieser Option mitteilen „ich weiß, was ich tue und bin mir aller möglichen Konsequenzen bewusst, also lass mich 'ran“. Diese Option ist das programmtechnische Äquivalent dazu, bei eingeschaltetem Strom an den Leitungen herumzudoktern – wenn Sie nicht wissen, was Sie tun, bekommen Sie einen gehörigen Schlag.

`--force-log`
Erzwingt, dass ein zweifelhafter Parameter, der an die Optionen `--message` (`-m`) oder `--file` (`-F`) übergeben wird, als gültig akzeptiert wird. Standardmäßig erzeugt Subversion eine Fehlermeldung, falls Parameter für diese Optionen den Eindruck erwecken, sie seien stattdessen Ziele des Unterbefehls. Wenn Sie beispielsweise den Pfad einer versionierten Datei der Option `--file` (`-F`) übergeben, nimmt Subversion an, dass Sie einen Fehler gemacht haben, der Pfad als Zieldatei für die Operation gedacht war und Sie einfach vergessen haben, eine andere – unversionierte – Datei als Quelle Ihrer Protokollnachricht anzugeben. Um Ihre Absicht zu bestätigen und die Fehlermeldungen zu verhindern, übergeben Sie die Option `--force-log` an Unterbefehle, die Protokollnachrichten akzeptieren.

`--help` (`-h` oder `-?`)
Wird diese Option mit einem oder mehreren Unterbefehlen verwendet, zeigt es den eingebauten Hilfetext für jeden Unterbefehl an. Wird sie alleine verwendet, wird der allgemeine Hilfetext des Clients angezeigt.

`--ignore-ancestry`
Teilt Subversion mit, beim Ermitteln von Unterschieden die Abstammung zu ignorieren (allein der Inhalt von Pfaden wird berücksichtigt).

`--ignore-externals`
Teilt Subversion mit, Externals-Definitionen und die von ihnen verwalteten externen Arbeitskopien zu ignorieren.

`--incremental`

Erzeugt Ausgaben in einem Format, dass zum Verketten geeignet ist.

`--keep-changelists`
Teilt Subversion mit, Änderungslisten nach der Übergabe nicht zu löschen.

`--keep-local`
Erhält die lokale Kopie einer Datei oder eines Verzeichnisses (verwendet in Verbindung mit dem Befehl **svn delete**).

`--limit` (`-l`) *`NUM`*
Zeigt nur die ersten *`NUM`* Protokollnachrichten an.

`--message` (`-m`) *`MESSAGE`*
Zeigt an, dass Sie eine Protokollnachricht oder einen Sperrkommentar auf der Kommandozeile nach dieser Option angeben. Zum Beispiel:

```
$ svn commit -m "Sie schaffen es nicht bis Sonntag."
```

`--native-eol` *`ARG`*
Veranlasst **svn export**, eine bestimmte Zeilenende-Sequenz zu verwenden, als wäre es die auf der Client-Plattform übliche. *`ARG`* kann entweder `CR`, `LF` oder `CRLF` sein.

`--new` *`ARG`*
Verwendet *`ARG`* als das neuere Ziel (in Verbindung mit **svn diff**).

`--no-auto-props`
Verhindert Auto-Props, wobei die Anweisung `enable-auto-props` in der Datei `config` aufgehoben wird.

`--no-diff-deleted`
Verhindert, dass Subversion Unterschiede gelöschter Dateien anzeigt. Das Standardverhalten für gelöschte Dateien ist, dass **svn diff** die gleichen Unterschiede anzeigt, die Sie sähen, wenn Sie die Datei behalten aber ihren Inhalt gelöscht hätten.

`--no-ignore`
Zeigt Dateien in der Statusliste, die normalerweise nicht angezeigt würden, da deren Name auf ein Muster passt, dass in der Konfigurationsoption `global-ignores` oder der Eigenschaft `svn:ignore` angegeben ist. Siehe „Config“ und „Ignorieren unversionierter Objekte“ für weitergehende Informationen.

`--no-unlock`
Teilt Subversion mit, Dateien nicht automatisch zu entsperren (das Standardverhalten nach der Übergabe ist es, alle Dateien, die übergeben wurden, zu entsperren). Siehe „Sperren“ für weitergehende Informationen.

`--non-recursive` (`-N`)
Überholt. Verhindert, dass ein Unterbefehl rekursiv auf Unterverzeichnisse angewendet wird. Die meisten Unterbefehle verhalten sich standardmäßig rekursiv, doch einige nicht. Anwender sollten diese Option vermeiden und stattdessen die präzisere Option `--depth` verwenden. Bei den meisten Unterbefehlen führt die `--non-recursive` zu demselben Verhalten wie die Angabe von `--depth=files`, es gibt jedoch Ausnahmen: das nicht-rekursive **svn status** arbeitet auf der Tiefe `immediates`, und die nicht-rekursiven Formen von **svn revert**, **svn add** und **svn commit** arbeiten auf der Tiefe `empty`.

`--notice-ancestry`
Berücksichtigt beim Ermitteln von Unterschieden die Abstammung.

`--old` *`ARG`*
Verwendet *`ARG`* als das ältere Ziel (in Verbindung mit **svn diff**).

`--parents`
Erzeugt und fügt im Rahmen einer Operation nicht existierende oder unversionierte Elternverzeichnisse der Arbeitskopie oder dem Projektarchiv hinzu. Das ist nützlich, um automatisch mehrere Unterverzeichnisse zu erzeugen, wo aktuell keine existieren. Wenn es auf einen URL angewendet wird, werden alle Verzeichnisse bei einer einzigen Übergabe erzeugt.

`--quiet` (`-q`)
Fordert den Client auf, nur die wichtigsten Informationen beim Ausführen einer Operation auszugeben.

`--record-only`
Markiert Revisionen als zusammengeführt; zur Verwendung mit `--revision` (`-r`).

`--recursive` (`-R`)
Veranlasst einen Unterbefehl, rekursiv Unterverzeichnisse zu durchlaufen. Die meisten Unterbefehle machen das standardmäßig.

`--reintegrate`
Bei Verwendung mit dem Unterbefehl **svn merge** werden alle Änderungen des Quell-URL mit der Arbeitskopie zusammengeführt. Für Details siehe „Einen Zweig synchron halten“.

`--relocate` *`FROM TO [PATH...]`*
Bei Verwendung mit dem Unterbefehl **svn switch** wird der Ort des Projektarchivs geändert, auf den sich Ihre Arbeitskopie bezieht. Das ist dann nützlich, wenn sich der Ort Ihres Projektarchivs ändert und Sie eine bestehende Arbeitskopie haben, die Sie weiterverwenden möchten. Für weitere Details sowie ein Beispiel siehe svn switch (sw).

`--remove` *`ARG`*
Entfernt *`ARG`* aus einer Änderungsliste.

`--revision` (`-r`) *`REV`*
Zeigt an, dass Sie eine Revision (oder ein Revisionsintervall) für eine bestimmte Operation angeben. Sie können der Option dazu Revisionsnummern, Schlüsselworte oder Daten (innerhalb von geschweiften Klammern) als Argument übergeben. Wenn Sie ein Revisionsintervall angeben möchten, können Sie zwei durch einen Doppelpunkt getrennte Revisionen übergeben. Zum Beispiel:

```
$ svn log -r 1729
$ svn log -r 1729:HEAD
$ svn log -r 1729:1744
$ svn log -r {2001-12-04}:{2002-02-17}
$ svn log -r 1729:{2002-02-17}
```

Für weitere Informationen siehe „Revisions-Schlüsselworte“.

`--revprop`
Wirkt auf eine Revisions-Eigenschaft anstatt auf eine datei- oder verzeichnisspezifische Eigenschaft. Diese Option erfordert die Angabe einer Revision mit der Option `--revision` (`-r`).

`--set-depth` *`ARG`*
Setzt die Wirktiefe eines Verzeichnisses in der Arbeitskopie auf einen der Werte `exclude`, `empty`, `files`, `immediates` oder `infinity`. Für eine detaillierte Erörterung derer Bedeutung und wie diese Option zu verwenden ist, siehe „Verzeichnis-Teilbäume“.

`--show-revs` *`ARG`*
Verwendet, um **svn mergeinfo** mitzuteilen, dass `merged` (zusammengeführte) oder `eligible` (in Frage kommende) Revisionen angezeigt werden sollen.

`--show-updates` (`-u`)
Teilt dem Client mit, Informationen über die Dateien anzuzeigen, die in Ihrer Arbeitskopie nicht mehr dem neuesten Stand entsprechen. Hierdurch wird keine Ihrer Dateien aktualisiert – es wird nur angezeigt, welche Dateien aktualisiert würden, wenn Sie anschließend **svn update** verwendeten.

`--stop-on-copy`
Veranlasst einen Subversion-Unterbefehl, der die Geschichte einer versionierten Ressource durchläuft, mit der Sammlung der Daten aufzuhören, sobald eine Kopie – dass heißt, ein Ort in der Versionsgeschichte, der von einem anderen Ort des Projektarchivs kopiert wurde – angetroffen wird.

`--strict`

Veranlasst Subversion, eine strenge Semantik anzuwenden, ein ziemlich ungenaues Konzept, es sei denn, es betrifft bestimmte Unterbefehle (und zwar **svn propget**).

`--summarize`
Statt detaillierter Ausgabe nur grob zusammengefasste Informationen über die Operation anzeigen.

`--targets` *`FILENAME`*
Teilt Subversion mit, zusätzliche Zielpfade für die Operation aus *`FILENAME`* zu lesen. *`FILENAME`* soll einen Pfad pro Zeile beinhalten, wobei dieselbe Zeichenkodierung und Formatierung erwartet wird, die auch direkt als Kommandozeilenargument angegeben worden wäre.

`--use-merge-history` (`-g`)
Zusätzliche Informationen aus der Geschichte der Zusammenführungen wird verwendet oder angezeigt.

`--verbose` (`-v`)
Fordert den Client auf, beim Ausführen eines Unterbefehls soviel Information auszugeben, wie er kann. Das kann dazu führen, dass Subversion zusätzliche Felder, detaillierte Informationen zu jeder Datei oder zusätzliche Informationen über seine Tätigkeiten ausgibt.

`--version`
Gibt die Versionsinformation des Clients aus. Diese Information umfasst nicht nur die Versionsnummer des Clients sondern auch eine Auflistung aller vom Client unterstützten Module für den Zugriff auf ein Subversion-Projektarchiv. Mit `--quiet` (`-q`) wird nur die Versionsnummer in Kurzform ausgegeben.

`--with-all-revprops`
In Verbindung mit der Option `--xml` von **svn log** werden alle Revisions-Eigenschaften, sowohl die standardmäßig von Subversion verwendeten als auch etwaige anwenderdefinierte, abgerufen und in der Protokollausgabe angezeigt.

`--with-no-revprops`
In Verbindung mit der Option `--xml` von **svn log** werden alle Revisions-Eigenschaften in der Protokollausgabe unterdrückt, einschließlich der Standard-Protokollnachricht, des Autors und des Zeitstempels der Revision.

`--with-revprop` *`ARG`*
In Verbindung mit einem Befehl, der in das Projektarchiv schreibt, wird die Revisions-Eigenschaft, bei Verwendung des Formats *`NAME=VALUE`*, *`NAME`* auf den Wert *`VALUE`* gesetzt. In Verbindung mit **svn log** im `--xml`-Modus wird der Wert von *`ARG`* in der Protokollausgabe angezeigt.

`--xml`
Die Ausgabe erfolgt im XML-Format.

svn-Unterbefehle

Hier sind die verschiedenen Unterbefehle für das Programm **svn**. Der Kürze halber lassen wir die globalen Optionen (beschrieben in „svn-Optionen“) bei den folgenden Beschreibungen der Unterbefehle aus.

Name

svn add — Dateien, Verzeichnisse oder symbolische Links hinzufügen.

Aufruf

```
svn add PATH...
```

Beschreibung

Dateien, Verzeichnisse oder symbolische Links in Ihrer Arbeitskopie werden zum Hinzufügen ins Projektarchiv vorgemerkt. Bei Ihrer nächsten Übergabe werden sie in das Projektarchiv geladen. Wenn Sie etwas hinzufügen möchten, es sich vor der Übergabe aber anders überlegen sollten, können Sie die Vormerkung mit **svn revert** rückgängig machen.

Optionen

```
--auto-props
--depth ARG
--force
--no-auto-props
--no-ignore
--parents
--quiet (-q)
--targets FILENAME
```

Beispiele

Eine Datei zur Arbeitskopie hinzufügen:

```
$ svn add foo.c
A         foo.c
```

Beim Hinzufügen eines Verzeichnisses ist Rekursion das Standardverhalten von **svn add**:

```
$ svn add testdir
A         testdir
A         testdir/a
A         testdir/b
A         testdir/c
A         testdir/d
```

Sie können ein Verzeichnis ohne seinen Inhalt hinzufügen:

```
$ svn add --depth=empty otherdir
A         otherdir
```

Normalerweise überspringt der Befehl **svn add *** Verzeichnisse, die sich bereits unter Versionskontrolle befinden. Manchmal möchten Sie jedoch jedes unversionierte Objekt in Ihrer Arbeitskopie hinzufügen, auch solche, die tiefer verborgen sind. Die Option `--force` veranlasst **svn add**, rekursiv versionierte Verzeichnisse zu bearbeiten:

```
$ svn add * --force
A         foo.c
A         somedir/bar.c
A  (bin)  otherdir/docs/baz.doc
…
```

Name

svn blame (praise, annotate, ann) — Autor- und Revisionsinformationen innerhalb der angegebenen Dateien oder URLs ausgeben.

Aufruf

```
svn blame TARGET[@REV]...
```

Beschreibung

Autor- und Revisionsinformationen innerhalb der angegebenen Dateien oder URLs ausgeben. Jede Textzeile erhält am Zeilenanfang eine Anmerkung mit dem Autor (Anwendernamen) und der Revisionsnummer der letzten Änderung.

Optionen

```
--extensions (-x) ARG
--force
--incremental
--revision (-r) REV
--use-merge-history (-g)
--verbose (-v)
--xml
```

Beispiele

Wenn Sie den Quelltext von `readme.txt` in Ihrem Test-Projektarchiv mit Anmerkungen sehen wollen:

```
$ svn blame http://svn.red-bean.com/repos/test/readme.txt
     3      sally This is a README file.
     5      harry You should read this.
```

Auch wenn **svn blame** behauptet, dass Harry `readme.txt` zuletzt in Revision 5 geändert habe, sollten Sie genau prüfen, was diese Revision geändert hat, um sicherzugehen, dass Harry den *Kontext* der Zeile geändert hat – es könnte sein, dass er nur Leerzeichen angepasst hat.

Falls Sie die Option `--xml` benutzen, bekommen Sie die Anmerkungen als XML-Ausgabe, jedoch nicht den eigentlichen Inhalt der Zeilen:

```
$ svn blame --xml http://svn.red-bean.com/repos/test/readme.txt
<?xml version="1.0"?>
<blame>
<target
   path="readme.txt">
<entry
   line-number="1">
<commit
   revision="3">
<author>sally</author>
<date>2008-05-25T19:12:31.428953Z</date>
</commit>
</entry>
<entry
   line-number="2">
<commit
```

```
   revision="5">
<author>harry</author>
<date>2008-05-29T03:26:12.293121Z</date>
</commit>
</entry>
</target>
</blame>
```

Name

svn cat — Ausgeben des Inhalts der angegebenen Datei oder des URLs.

Aufruf

```
svn cat TARGET[@REV]...
```

Beschreibung

Ausgeben des Inhalts der angegebenen Datei oder des URLs. Um den Inhalt von Verzeichnissen aufzulisten, siehe **svn list** später in diesem Kapitel.

Optionen

```
--revision (-r) REV
```

Beispiele

Wenn Sie die Datei `readme.txt` in Ihrem Projektarchiv ansehen möchten, ohne sie auszuchecken:

```
$ svn cat http://svn.red-bean.com/repos/test/readme.txt
This is a README file.
You should read this.
$
```

Falls Ihre Arbeitskopie nicht auf dem neuesten Stand ist (oder falls Sie lokale Änderungen haben) und Sie die HEAD-Revision einer Datei in Ihrer Arbeitskopie betrachten möchten, holt **svn cat -r HEAD *FILENAME*** die HEAD-Revision automatisch vom angegebenen Pfad:

```
$ cat foo.c
This file is in my local working copy
and has changes that I've made.
$ svn cat -r HEAD foo.c
Latest revision fresh from the repository!
$
```

Name

svn changelist (cl) — Hinzufügen (oder Entfernen) lokaler Pfade zu (oder von) einer Änderungsliste.

Aufruf

```
changelist CLNAME TARGET...

changelist --remove TARGET...
```

Beschreibung

Verwendet zum Aufteilen von Dateien einer Arbeitskopie in eine Änderungsliste (eine logisch benannte Gruppierung), um Benutzern die Möglichkeit zu geben, auf einfache Weise mit mehreren Dateisammlungen innerhalb einer einzigen Arbeitskopie zu arbeiten.

Optionen

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--remove
--targets FILENAME
```

Beispiel

Ändern Sie drei Dateien, fügen Sie sie einer Änderungsliste hinzu und übergeben dann nur die Dateien auf dieser Änderungsliste:

```
$ svn changelist issue1729 foo.c bar.c baz.c

Pfad »foo.c« ist nun ein Element der Änderungsliste »issue1729«.
Pfad »bar.c« ist nun ein Element der Änderungsliste »issue1729«.
Pfad »baz.c« ist nun ein Element der Änderungsliste »issue1729«.
$ svn status
A       someotherfile.c
A       test/sometest.c


--- Änderungsliste 'issue1729':
A       foo.c
A       bar.c
A       baz.c
$ svn commit --changelist issue1729 -m "Fixing Issue 1729."

Hinzufügen     bar.c
Hinzufügen     baz.c
Hinzufügen     foo.c
Übertrage Daten...
Revision 2 übertragen.
$ svn status
A       someotherfile.c
A       test/sometest.c
$
```

Beachten Sie, dass im vorigen Beispiel nur die Dateien der Änderungsliste `issue1729` übergeben wurden.

Name

svn checkout (co) — Auschecken einer Arbeitskopie aus einem Projektarchiv.

Aufruf

```
svn checkout URL[@REV]... [PATH]
```

Beschreibung

Checkt eine Arbeitskopie aus einem Projektarchiv aus. Wird *PATH* ausgelassen, wird der Basisname des URL als Ziel verwendet. Werden mehrere URLs angegeben, wird jeder in ein Unterverzeichnis von *PATH* ausgecheckt, wobei der Name des Unterverzeichnisses dem Basisnamen des URL entspricht.

Optionen

```
--depth ARG
--force
--ignore-externals
--quiet (-q)
--revision (-r) REV
```

Beispiele

Eine Arbeitskopie in ein Verzeichnis `mine` auschecken:

```
$ svn checkout file:///var/svn/repos/test mine
A    mine/a
A    mine/b
A    mine/c
A    mine/d

Ausgecheckt. Revision 20.
$ ls
mine
$
```

Zwei unterschiedliche Verzeichnisse in zwei getrennte Arbeitskopien auschecken:

```
$ svn checkout file:///var/svn/repos/test \
               file:///var/svn/repos/quiz
A    test/a
A    test/b
A    test/c
A    test/d

Ausgecheckt. Revision 20.
A    quiz/l
A    quiz/m

Ausgecheckt. Revision 13.
$ ls
quiz  test
$
```

Zwei unterschiedliche Verzeichnisse in zwei getrennte Arbeitskopien auschecken, jedoch beide in ein Verzeichnis `working-copies`:

```
$ svn checkout file:///var/svn/repos/test \
               file:///var/svn/repos/quiz \
               working-copies
A    working-copies/test/a
A    working-copies/test/b
A    working-copies/test/c
A    working-copies/test/d

Ausgecheckt. Revision 20.
A    working-copies/quiz/l
A    working-copies/quiz/m

Ausgecheckt. Revision 13.
$ ls
working-copies
```

Falls Sie das Auschecken unterbrechen (oder das Auschecken durch irgendetwas anderes unterbrochen wird, etwa durch Verlust der Netzverbindung o.ä.), können Sie es erneut durch Eingabe des gleichen Befehls oder durch die Aktualisierung der unvollständigen Arbeitskopie wiederholen:

```
$ svn checkout file:///var/svn/repos/test mine
A    mine/a
A    mine/b
^C

svn: Die Operation wurde unterbrochen
svn: Abbruchsignal empfangen
$ svn checkout file:///var/svn/repos/test mine
A    mine/c
^C

svn: Die Operation wurde unterbrochen
svn: Abbruchsignal empfangen
$ svn update mine
A    mine/d

Aktualisiert zu Revision 20.
$
```

Falls Sie eine andere als die letzte Revision auschecken möchten, können Sie dem Befehl **svn checkout** die Option `--revision` (`-r`) mitgeben:

```
$ svn checkout -r 2 file:///var/svn/repos/test mine
A    mine/a

Ausgecheckt. Revision 2.
$
```

Standardmäßig wird sich Subversion beschweren, falls Sie versuchen, ein Verzeichnis auf einem bestehenden Verzeichnis auszuchecken, das Dateien oder Unterverzeichnisse beinhaltet, die der Checkout selbst angelegt hätte. Verwenden Sie die Option `--force`, um diese Schutzmaßnahme zu umgehen. Wenn Sie mit der Option `--force` auschecken, gerät jede unversionierte Dateien im Checkout-Zielverzeichnisbaum, die normalerweise den Checkout verhindert hätte, unter Versionskontrolle, jedoch ändert Subversion ihren Inhalt nicht. Wenn dieser Inhalt sich vom Inhalt der Datei an dieser Stelle

im Projektarchiv unterscheidet (der als Teil des Checkouts heruntergeladen wurde), erscheint die Datei nach vollendetem Checkout als lokal geändert, wobei die Änderungen der Überführung der versionierten, ausgecheckten Datei in die unversionierte Datei vor dem Checkout entsprechen.

```
$ mkdir project
$ mkdir project/lib
$ touch project/lib/file.c
$ svn checkout file:///var/svn/repos/project/trunk project

svn: Verzeichnis »project/lib« konnte nicht hinzugefügt werden: ein nicht versi
oniertes Verzeichnis mit demselben Namen existiert bereits
$ svn checkout file:///var/svn/repos/project/trunk project --force
E    project/lib
A    project/lib/subdir
E    project/lib/file.c
A    project/lib/anotherfile.c
A    project/include/header.h

Ausgecheckt, Revision 21.
$ svn status wc
M       project/lib/file.c
$ svn diff wc
Index: project/lib/file.c
===================================================================
--- project/lib/file.c (revision 1)
+++ project/lib/file.c (working copy)
@@ -3 +0,0 @@
-/* file.c: Code for acting file-ishly. */
-#include <stdio.h>
-/* Not feeling particularly creative today. */

$
```

Wie auch bei anderen Arbeitskopien haben Sie die üblichen Möglichkeiten zur Auswahl: die lokalen „Änderungen“ teilweise oder vollständig rückgängig machen, sie übergeben oder mit Änderungen in Ihrer Arbeitskopie fortfahren.

Besonders nützlich ist diese Funktionalität, um vor Ort unversionierte Verzeichnisbäume zu importieren. Indem zunächst der Baum in das Projektarchiv importiert und dann über den unversionierten Baum mit der Option `--force` ausgecheckt wird, bewirken Sie hiermit eine Überführung des unversionierten Baums in eine Arbeitskopie.

```
$ svn mkdir -m "Create newproject project root." \
      file://var/svn/repos/newproject
$ svn import -m "Import initial newproject codebase." newproject \
      file://var/svn/repos/newproject/trunk

Hinzufügen     newproject/include
Hinzufügen     newproject/include/newproject.h
Hinzufügen     newproject/lib
Hinzufügen     newproject/lib/helpers.c
Hinzufügen     newproject/lib/base.c
Hinzufügen     newproject/notes
Hinzufügen     newproject/notes/README


Revision 22 übertragen.
$ svn checkout file://`pwd`/repos-1.6/newproject/trunk newproject --force
E    newproject/include
E    newproject/include/newproject.h
E    newproject/lib
E    newproject/lib/helpers.c
E    newproject/lib/base.c
E    newproject/notes
E    newproject/notes/README
```

```
Ausgecheckt. Revision 2.
$ svn status newproject
$
```

Name

svn cleanup — Die Arbeitskopie rekursiv aufräumen.

Aufruf

```
svn cleanup [PATH...]
```

Beschreibung

Die Arbeitskopie rekursiv aufräumen, wobei Sperren der Arbeitskopie aufgehoben werden und unvollendete Operationen wiederaufgenommen werden. Falls Sie jemals die Fehlermeldung `Arbeitskopie gesperrt` bekommen sollten, rufen Sie diesen Befehl auf, um alte Sperren zu entfernen und Ihre Arbeitskopie wieder in einen benutzbaren Zustand zu überführen.

Falls aus irgendwelchen Gründen **svn update** wegen eines Problems mit einem externen diff-Programm fehlschlagen sollte (z.B. durch Anwendereingaben oder Netzprobleme), übergeben Sie die Option `--diff3-cmd`, um dem cleanup-Prozess die Möglichkeit zu geben, notwendige Zusammenführungen mit Ihrem externen diff-Programm abzuschließen. Sie können mit der Option `--config-dir` auch irgendein Konfigurationsverzeichnis angeben, allerdings sollten Sie diese Optionen nur äußerst selten benötigen.

Optionen

```
--diff3-cmd CMD
```

Beispiele

An dieser Stelle gibt es nicht viel an Beispielen, da **svn cleanup** keine Ausgaben erzeugt. Falls Sie *PATH* nicht angeben, wird „`.`“ verwendet:

```
$ svn cleanup

$ svn cleanup /var/svn/working-copy
```

Name

svn commit (ci) — Änderungen aus der Arbeitskopie an das Projektarchiv übergeben.

Aufruf

```
svn commit [PATH...]
```

Beschreibung

Änderungen aus der Arbeitskopie an das Projektarchiv übergeben. Falls Sie keine Protokollnachricht, mit einer der Optionen `--file` (`-F`) oder `--message` (`-m`), angeben, startet **svn** Ihren Editor zum Verfassen einer Protokollnachricht. Siehe auch den Listeneintrag zu `editor-cmd` in „Config“.

svn commit verschickt alle gefundenen Sperrmarken und gibt Sperren auf alle mit *`PATH`* angegebenen Pfade frei, die (rekursiv) übergeben werden, sofern die Option `--no-unlock` nicht angegeben ist.

Falls Sie eine Übergabe einleiten und Subversion Ihren Editor zum Verfassen einer Protokollnachricht startet, können Sie immer noch abbrechen, ohne Ihre Änderungen zu übergeben. Wenn Sie die Übergabe abbrechen wollen, beenden Sie einfach Ihren Editor, ohne die Protokollnachricht zu sichern; dann wird Subversion Sie fragen, ob Sie die Übergabe abbrechen, ohne Protokollnachricht weitermachen oder die Nachricht erneut editieren möchten.

Optionen

```
--changelist ARG
--depth ARG
--editor-cmd CMD
--encoding ENC
--file (-F) FILENAME
--force-log
--keep-changelists
--message (-m) MESSAGE
--no-unlock
--quiet (-q)
--targets FILENAME
--with-revprop ARG
```

Beispiele

Übergabe einer einfachen Änderung an einer Datei mit der Protokollnachricht auf der Kommandozeile und Ihrem aktuellen Verzeichnis als implizites Ziel („`.`“):

```
$ svn commit -m "added howto section."
Sending        a

Übertrage Daten .
Revision 3 übertragen.
```

Übergabe einer Änderung an der Datei `foo.c` (ausdrücklich auf der Kommandozeile angegeben) mit der Protokollnachricht in der Datei `msg`:

```
$ svn commit -F msg foo.c

Sende          foo.c
Übertrage Daten .
Revision 5 übertragen.
```

Falls Sie mit der Option `--file` (`-F`) eine Datei unter Versionskontrolle für Ihre Protokollnachricht verwenden möchten, müssen Sie die Option `--force-log` angeben:

```
$ svn commit -F file_under_vc.txt foo.c

svn: Die Datei für die Logmeldung ist unter Versionskontrolle
svn: Die Datei für den Sperrkommentar ist versioniert; geben Sie »--force-log« an, um
sie zu verwenden

$ svn commit --force-log -F file_under_vc.txt foo.c

Sende          foo.c
Übertrage Daten .
Revision 6 übertragen.
```

Eine zur Löschung vorgemerkte Datei übergeben:

```
$ svn commit -m "removed file 'c'."

Lösche         c

Revision 7 übertragen.
```

Name

svn copy (cp) — Kopieren einer Datei oder eines Verzeichnisses in einer Arbeitskopie oder im Projektarchiv.

Aufruf

```
svn copy SRC[@REV]... DST
```

Beschreibung

Kopieren einer Datei oder mehrerer Dateien in der Arbeitskopie. *`SRC`* und *`DST`* können beide entweder ein Pfad in der Arbeitskopie (AK) oder ein URL sein. Sollen mehrere Quellen kopiert werden, fügen Sie diese als unmittelbare Kindobjekte *`DST`* hinzu (welches natürlich ein Verzeichnis sein muss).

AK # AK
: Element kopieren und zum Hinzufügen vormerken (mit Geschichte).

AK # URL
: Eine Kopie aus der AK direkt an den URL übergeben.

URL # AK
: URL in AK auschecken und zum Hinzufügen vormerken.

URL # URL
: Kopie vollständig auf dem Server. Dies wird normalerweise zum Anlegen von Zweigen und zum Etikettieren mit Tags verwendet.

Wird keine Peg-Revision (z.B. *`@REV`*) angegeben, so wird standardmäßig die Revision `BASE` für Dateien verwendet, die aus der Arbeitskopie kopiert werden, wohingegen die Revision `HEAD` für Dateien herangezogen wird, die von einem URL kopiert werden.

Sie können Dateien nur innerhalb eines einzelnen Projektarchivs kopieren. Subversion unterstützt nicht das Kopieren zwischen Projektarchiven.

Optionen

```
--editor-cmd CMD
--encoding ENC
--file (-F) FILENAME
--force-log
--ignore-externals
--message (-m) MESSAGE
--parents
--quiet (-q)
--revision (-r) REV
--with-revprop ARG
```

Beispiele

Kopie eines Objektes innerhalb Ihrer Arbeitskopie (dies merkt die Kopie vor – bis zur Übergabe wird nichts ins Projektarchiv übertragen):

```
$ svn copy foo.txt bar.txt
A         bar.txt
```

```
$ svn status
A  +   bar.txt
```

Kopieren mehrerer Dateien der Arbeitskopie in ein Unterverzeichnis:

```
$ svn copy bat.c baz.c qux.c src
A         src/bat.c
A         src/baz.c
A         src/qux.c
```

Kopieren von `bat.c`, Revision 8 in Ihre Arbeitskopie unter einem anderen Namen:

```
$ svn copy -r 8 bat.c ya-old-bat.c
A         ya-old-bat.c
```

Kopie eines Objektes in Ihrer Arbeitskopie zu einem URL im Projektarchiv (dies ist eine sofortige Übergabe, so dass Sie eine Protokollnachricht angeben müssen):

```
$ svn copy near.txt file:///var/svn/repos/test/far-away.txt -m "Remote copy."

Revision 8 übertragen.
```

Kopie eines Objektes aus dem Projektarchiv in Ihre Arbeitskopie (dies merkt die Kopie nur vor – vor der Übergabe gelangt nichts ins Projektarchiv):

```
$ svn copy file:///var/svn/repos/test/far-away -r 6 near-here
A         near-here
```

Dies ist die empfohlene Methode, eine gelöschte Datei im Projektarchiv wiederherzustellen!

Schließlich eine Kopie zwischen zwei URLs:

```
$ svn copy file:///var/svn/repos/test/far-away \
           file:///var/svn/repos/test/over-there -m "remote copy."

Revision 9 übertragen.
```

```
$ svn copy file:///var/svn/repos/test/trunk \
           file:///var/svn/repos/test/tags/0.6.32-prerelease -m "tag tree"
```

```
Revision 12 übertragen.
```

Dies ist die einfachste Methode, um eine Revision in Ihrem Projektarchiv mit einem „Tag“ zu etikettieren – kopieren Sie diese Revision (normalerweise HEAD lediglich mit **svn copy**) in Ihr `tags`-Verzeichnis.

Machen Sie sich keine Sorgen, falls sie das Etikettieren einmal vergessen haben sollten – Sie können immer eine ältere Revision angeben und jederzeit ein Tag vergeben:

```
$ svn copy -r 11 file:///var/svn/repos/test/trunk \
           file:///var/svn/repos/test/tags/0.6.32-prerelease \
           -m "Forgot to tag at rev 11"

Revision 13 übertragen.
```

Name

svn delete — Ein Objekt aus einer Arbeitskopie oder dem Projektarchiv löschen.

Aufruf

```
svn delete PATH...

svn delete URL...
```

Beschreibung

Die durch *PATH* angegebenen Objekte werden zur Löschung bei der nächsten Übergabe vorgemerkt. Dateien (und nicht übergebene Verzeichnisse) werden sofort aus der Arbeitskopie entfernt, sofern nicht die Option `--keep-local` angegeben ist. Der Befehl entfernt keine unversionierten oder geänderten Objekte; verwenden Sie die Option `--force`, um dieses Verhalten zu ändern.

Werden Objekte durch einen URL bestimmt, werden sie durch eine sofortige Übergabe aus dem Projektarchiv entfernt. Mehrere URLs werden atomar übergeben (alle oder keine).

Optionen

```
--editor-cmd CMD
--encoding ENC
--file (-F) FILENAME
--force
--force-log
--keep-local
--message (-m) MESSAGE
--quiet (-q)
--targets FILENAME
--with-revprop ARG
```

Beispiele

Die Verwendung von **svn delete** zum Löschen einer Datei aus der Arbeitskopie löscht die lokale Kopie der Datei, merkt die Datei jedoch nur zum Löschen aus dem Projektarchiv vor. Bei der Übergabe wird die Datei im Projektarchiv gelöscht.

```
$ svn delete myfile
D         myfile

$ svn commit -m "Deleted file 'myfile'."

Lösche         myfile
Übertrage Daten .
Revision 14 übertragen.
```

Das Löschen eines URLs wird jedoch sofort wirksam, so dass Sie eine Protokollnachricht angeben müssen:

```
$ svn delete -m "Deleting file 'yourfile'" \
             file:///var/svn/repos/test/yourfile

Revision 15 übertragen.
```

Hier ist ein Beispiel, wie die Löschung einer Datei mit lokalen Änderungen erzwungen werden kann:

```
$ svn delete over-there

svn: Versuch, eine beschränkte Operation für veränderte Ressource auszuführen
svn: Benutzen Sie »--force«, um diese Einschränkung aufzuheben
svn: »over-there« hat lokale Änderungen

$ svn delete - -force over-there
D         over-there
```

Verwenden Sie die Option `--keep-local`, um das standardmäßige Verhalten von **svn delete** zu verhindern, dass auch die Zieldatei entfernt wird, die für eine versionierte Löschung vorgemerkt war. Das ist in dem Fall nützlich, falls Sie feststellen, dass Sie versehentlich eine hinzugefügte Datei übergeben haben, die Sie zwar in Ihrer Arbeitskopie benötigen, jedoch nicht unter Versionskontrolle stehen soll.

```
$ svn delete --keep-local conf/program.conf
D         conf/program.conf

$ svn commit -m "Remove accidentally-added configuration file."

Lösche         conf/program.conf
Übertrage Daten .
Revision 21 übertragen.
$ svn status
?       conf/program.conf
$
```

Name

svn diff (di) — Anzeige der Unterschiede zwischen zwei Revisionen oder Pfaden.

Aufruf

```
diff [-c M | -r N[:M]] [TARGET[@REV]...]

diff [-r N[:M]] --old=OLD-TGT[@OLDREV] [--new=NEW-TGT[@NEWREV]] [PATH...]

diff OLD-URL[@OLDREV] NEW-URL[@NEWREV]
```

Beschreibung

Anzeige der Unterschiede zwischen zwei Pfaden. Sie können **svn diff** wie folgt verwenden:

- Verwenden Sie nur **svn diff**, um lokale Änderungen in einer Arbeitskopie anzuzeigen.
- Anzeige der Änderungen an *TARGET*s in *REV* zwischen zwei Revisionen. Alle *TARGET*s können entweder nur Pfade in der Arbeitskopie oder nur *URL*s sein. Falls *TARGET*s Pfade in der Arbeitskopie sind, ist der Standardwert von *N* BASE und *M* die Arbeitskopie; falls *TARGET*s *URL*s sind, muss *N* angegeben werden, und der Standardwert von *M* ist HEAD. Die Option `-c M` ist äquivalent zu `-r N:M`, wobei `N = M-1`. Die Verwendung von `-c -M` bewirkt den umgekehrten Fall: `-r M:N`, wobei `N = M-1`.
- Anzeige der Unterschiede zwischen *OLD-TGT* in *OLDREV* und *NEW-TGT* in *NEWREV*. Sind *PATH*s angegeben, sind sie relativ zu *OLD-TGT* und *NEW-TGT*, und die Ausgabe wird auf Unterschiede für diese Pfade beschränkt. *OLD-TGT* und *NEW-TGT* können Pfade in der Arbeitskopie sein oder *URL[@REV]*. Der Standardwert von *NEW-TGT* ist *OLD-TGT*, falls nicht angegeben. `-r N` setzt den Standardwert von *OLDREV* auf N; `-r N:M` setzt den Standardwert von *OLDREV* auf *N* und *NEWREV* auf *M*.

`svn diff OLD-URL[@OLDREV] NEW-URL[@NEWREV]` ist die Kurzform für **`svn diff --old=OLD-URL[@OLDREV] --new=NEW-URL[@NEWREV].`**

`svn diff -r N:M URL` ist die Kurzform für **`svn diff -r N:M --old=URL --new=URL`**.

`svn diff [-r N[:M]] URL1[@N] URL2[@M]` ist die Kurzform für **`svn diff [-r N[:M]] --old=URL1 --new=URL2`**.

Falls *TARGET* ein URL ist, können die Revisionen N und M entweder mir der Option `--revision` (`-r`) oder in der Notation „@“ angegeben werden, wie vorher beschrieben.

Falls es sich bei *TARGET* um einen Pfad in der Arbeitskopie handelt, besteht das Standardverhalten (sofern die Option `--revision` (`-r`) nicht angegeben wird) darin, die Unterschiede zwischen der Basisversion und der Arbeitskopie von *TARGET* auszugeben. Wird in diesem Fall jedoch die Option `--revision` (`-r`) angegeben, bedeutet das:

`--revision N:M`
: Der Server vergleicht *TARGET@N* und *TARGET@M*.

`--revision N`
: Der Client vergleicht *TARGET@N* mit der Arbeitskopie.

Wird die alternative Syntax verwendet, vergleicht der Server *URL1* und *URL2* in Revision *N* bzw. *M*. Wird entweder *N* oder *M* ausgelassen, wird der Wert HEAD angenommen.

Standardmäßig ignoriert **svn diff** die Herkunft von Dateien und vergleicht lediglich den Inhalt der beiden Dateien. Falls Sie `--notice-ancestry` verwenden, wird die Herkunft der Pfade beim Vergleich der Revisionen berücksichtigt (d.h., falls Sie **svn diff** auf zwei Dateien gleichen Inhalts jedoch unterschiedlicher Herkunft anwenden, wird es dargestellt, als sei der gesamte Inhalt entfernt und wieder hinzugefügt worden).

Optionen

```
--change (-c) ARG
--changelist ARG
--depth ARG
--diff-cmd CMD
--extensions (-x) ARG
--force
--new ARG
--no-diff-deleted
--notice-ancestry
--old ARG
--revision (-r) ARG
--summarize
--xml
```

Beispiele

Vergleich von BASE und Ihrer Arbeitskopie (eine der beliebtesten Anwendungen von **svn diff**):

```
$ svn diff COMMITTERS
Index: COMMITTERS
===================================================================

--- COMMITTERS (Revision 4404)
+++ COMMITTERS (Arbeitskopie)
```

Betrachten der Änderungen in der Datei COMMITTERS, Revision 9115:

```
$ svn diff -c 9115 COMMITTERS
Index: COMMITTERS
===================================================================

--- COMMITTERS (Revision 3900)
+++ COMMITTERS (Arbeitskopie)
```

Die Änderungen in der Arbeitskopie mit einer älteren Revision vergleichen:

```
$ svn diff -r 3900 COMMITTERS
Index: COMMITTERS
===================================================================

--- COMMITTERS (Revision 3900)
+++ COMMITTERS (Arbeitskopie)
```

Vergleich von Revision 3000 mit Revision 3500 unter Verwendung der „@“-Syntax:

```
$ svn diff http://svn.collab.net/repos/svn/trunk/COMMITTERS@3000 \
           http://svn.collab.net/repos/svn/trunk/COMMITTERS@3500
Index: COMMITTERS
===================================================================
```

```
--- COMMITTERS (Revision 3000)
+++ COMMITTERS (Revision 3500)
…
```

Vergleich von Revision 3000 mit Revision 3500 unter Verwendung der Bereichsschreibweise (in diesem Fall wird nur ein URL angegeben):

```
$ svn diff -r 3000:3500 http://svn.collab.net/repos/svn/trunk/COMMITTERS
Index: COMMITTERS
===================================================================

--- COMMITTERS (Revision 3000)
+++ COMMITTERS (Revision 3500)
```

Vergleich der Revision 3000 mit Revision 3500 aller Dateien in `trunk` unter Verwendung der Bereichsschreibweise:

```
$ svn diff -r 3000:3500 http://svn.collab.net/repos/svn/trunk
```

Vergleich der Revision 3000 mit Revision 3500 von nur drei Dateien in `trunk` unter Verwendung der Bereichsschreibweise:

```
$ svn diff -r 3000:3500 --old http://svn.collab.net/repos/svn/trunk \
           COMMITTERS README HACKING
```

Falls Sie eine Arbeitskopie haben, können Sie die Unterschiede ermitteln, ohne die langen URLs einzugeben:

```
$ svn diff -r 3000:3500 COMMITTERS
Index: COMMITTERS
===================================================================

--- COMMITTERS (Revision 3000)
+++ COMMITTERS (Revision 3500)
…
```

Verwendung von `--diff-cmd` *`CMD`* `--extensions` (`-x`), um Argumente direkt an das externe diff-Programm zu übergeben:

```
$ svn diff --diff-cmd /usr/bin/diff -x "-i -b" COMMITTERS
Index: COMMITTERS
===================================================================
0a1,2
> This is a test
>
```

Schließlich können Sie die Option `--xml` zusammen mit der Option `--summarize` verwenden, um die Änderungen zwischen den Revisionen, nicht jedoch den Inhalt des eigentlichen Diffs, in XML anzusehen:

```
$ svn diff --summarize --xml http://svn.red-bean.com/repos/test@r2 \
           http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<diff>
<paths>
<path
   props="none"
   kind="file"
   item="modified">http://svn.red-bean.com/repos/test/sandwich.txt</path>
<path
   props="none"
   kind="file"
   item="deleted">http://svn.red-bean.com/repos/test/burrito.txt</path>
<path
   props="none"
   kind="dir"
   item="added">http://svn.red-bean.com/repos/test/snacks</path>
</paths>
</diff>
```

Name

svn export — Exportieren eines sauberen Verzeichnisbaums.

Aufruf

```
svn export [-r REV] URL[@PEGREV] [PATH]

svn export [-r REV] PATH1[@PEGREV] [PATH2]
```

Beschreibung

Die erste Form exportiert einen sauberen Verzeichnisbaum aus dem mit *URL* bezeichneten Projektarchiv – falls angegeben, in Revision *REV*, sonst von HEAD – nach *PATH*. Wird *PATH* nicht angegeben, wird die letzte Komponente des *URL* als lokaler Verzeichnisname verwendet.

Die zweite Form exportiert einen sauberen Verzeichnisbaum aus der mit *PATH1* bezeichneten Arbeitskopie nach *PATH2*. Alle lokalen Änderungen bleiben erhalten, jedoch werden nur versionskontrollierte Dateien kopiert.

Optionen

```
--depth ARG
--force
--ignore-externals
--native-eol EOL
--quiet (-q)
--revision (-r) REV
```

Beispiele

Export aus Ihrer Arbeitskopie (zeigt nicht jede Datei und jedes Verzeichnis an):

```
$ svn export a-wc my-export

Export abgeschlossen.
```

Export direkt aus dem Projektarchiv (zeigt jede Datei und jedes Verzeichnis an):

```
$ svn export file:///var/svn/repos my-export
A    my-export/test
A    my-export/quiz
…

Exportiert, Revision 15.
```

Beim Erstellen betriebssystemspezifischer Release-Pakete kann es nützlich sein, einen Baum zu exportieren, der ein bestimmtes Zeichen für Zeilenenden verwendet. Die Option `--native-eol` sorgt dafür, es sind davon jedoch nur Dateien betroffen, die mit `svn:eol-style = native`-Eigenschaften versehen sind. Um beispielsweise einen Baum mit CRLF-Zeilenenden zu exportieren (vielleicht für die Verteilung in einer Windows `.zip`-Datei):

```
$ svn export file:///var/svn/repos my-export --native-eol CRLF
```

```
A    my-export/test
A    my-export/quiz
…

Exportiert, Revision 15.
```

Bei der Option `--native-eol` können Sie `LR`, `CR` oder `CRLF` als Zeilenendetyp angeben.

Name

svn help (h, ?) — Hilfe!

Aufruf

```
svn help [SUBCOMMAND...]
```

Beschreibung

Dies ist Ihr bester Freund wenn Sie Subversion benutzen und dieses Buch nicht greifbar ist!

Optionen

Keine

Name

svn import — Eine unversionierte Datei oder einen unversionierten Baum in das Projektarchiv übertragen.

Aufruf

```
svn import [PATH] URL
```

Beschreibung

Eine Kopie von *PATH* wird rekursiv nach *URL* übertragen. Wird *PATH* nicht angegeben, wird „.“ angenommen. Elternverzeichnisse werden im Projektarchiv bei Bedarf angelegt. Unversionierbare Objekte, wie Gerätedateien oder Pipes, werden ignoriert, auch wenn die Option `--force` verwendet wird.

Optionen

```
--auto-props
--depth ARG
--editor-cmd CMD
--encoding ENC
--file (-F) FILENAME
--force
--force-log
--message (-m) MESSAGE
--no-auto-props
--no-ignore
--quiet (-q)
--with-revprop ARG
```

Beispiele

Dieser Aufruf importiert das lokale Verzeichnis `myproj` nach `trunk/misc` in Ihrem Projektarchiv. Das Verzeichnis `trunk/misc` braucht vor dem Import nicht vorhanden zu sein – **svn import** erzeugt rekursiv die Verzeichnisse für Sie.

```
$ svn import -m "New import" myproj \
             http://svn.red-bean.com/repos/trunk/misc

Hinzufügen     myproj/sample.txt
…
Übertrage Daten .........
Revision 16 übertragen.
```

Denken Sie daran, dass das Verzeichnis `myproj` *nicht* im Projektarchiv angelegt wird. Falls Sie das möchten, fügen Sie einfach `myproj` an das Ende des URL an:

```
$ svn import -m "New import" myproj \
             http://svn.red-bean.com/repos/trunk/misc/myproj

Hinzufügen     myproj/sample.txt
…
Übertrage Daten .........
Revision 16 übertragen.
```

Beachten Sie, dass nach dem Importieren der Daten der Originalbaum *nicht* unter Versionskontrolle ist. Um mit der Arbeit zu beginnen, müssen Sie noch mit **svn checkout** eine frische Arbeitskopie des Baums erzeugen.

Name

svn info — Informationen über ein lokales oder entferntes Objekt anzeigen.

Aufruf

```
svn info [TARGET[@REV]...]
```

Beschreibung

Informationen über die angegebenen Arbeitskopiepfade oder URLs ausgeben. Die für jeden Pfad angezeigte Information kann (sofern für das an dieser Stelle vorhandene Objekt passend) beinhalten:

- Information über das Projektarchiv in dem das Objekt versioniert wird
- die letzte für die angegebene Version des Objektes durchgeführte Übergabe
- etwaige von Anwendern angelegte Sperren auf diesem Objekt
- lokale Informationen für Vormerkungen (hinzugefügt, gelöscht, kopiert, usw.)
- lokale Konfliktinformationen

Optionen

```
--changelist ARG
--depth ARG
--incremental
--recursive (-R)
--revision (-r) REV
--targets FILENAME
--xml
```

Beispiele

svn info zeigt Ihnen alle nützlichen Informationen über Objekte in der Arbeitskopie. Es zeigt Ihnen Informationen für Dateien:

```
$ svn info foo.c

Pfad: foo.c
Name: foo.c
URL: http://svn.red-bean.com/repos/test/foo.c
Basis des Projektarchivs: http://svn.red-bean.com/repos/test
UUID des Projektarchivs: 5e7d134a-54fb-0310-bd04-b611643e5c25
Revision: 4417
Knotentyp: file
Plan: normal
Letzter Autor: sally
Letzte geänderte Rev: 20
Letztes Änderungsdatum: 2003-01-13 16:43:13 -0600 (Mon, 13 Jan 2003)
Text zuletzt geändert: 2003-01-16 21:18:16 -0600 (Thu, 16 Jan 2003)
Eigenschaften zuletzt geändert: 2003-01-13 21:50:19 -0600 (Mon, 13 Jan 2003)
Prüfsumme: d6aeb60b0662ccceb6bce4bac344cb66
```

Informationen über Verzeichnisse werden auch angezeigt:

```
$ svn info vendors

Pfad: vendors
URL: http://svn.red-bean.com/repos/test/vendors
Basis des Projektarchivs: http://svn.red-bean.com/repos/test
UUID des Projektarchivs: 5e7d134a-54fb-0310-bd04-b611643e5c25
Revision: 19
Knotentyp: directory
Plan: normal
Letzter Autor: harry
Letzte geänderte Rev: 19
Letztes Änderungsdatum: 2003-01-16 23:21:19 -0600 (Thu, 16 Jan 2003)
Eigenschaften zuletzt geändert: 2003-01-16 23:39:02 -0600 (Thu, 16 Jan 2003)
```

svn info funktioniert auch mit URLs (beachten Sie, dass die Datei `readme.doc` in diesem Beispiel gesperrt ist, so dass auch Informationen zur Sperre angezeigt werden):

```
$ svn info http://svn.red-bean.com/repos/test/readme.doc

Pfad: readme.doc
Name: readme.doc
URL: http://svn.red-bean.com/repos/test/readme.doc
Basis des Projektarchivs: http://svn.red-bean.com/repos/test
UUID des Projektarchivs: 5e7d134a-54fb-0310-bd04-b611643e5c25
Revision: 1
Knotentyp: file
Plan: normal
Letzter Autor: sally
Letzte geänderte Rev: 42
Letztes Änderungsdatum: 2003-01-14 23:21:19 -0600 (Tue, 14 Jan 2003)
Sperrmarke: opaquelocktoken:14011d4b-54fb-0310-8541-dbd16bd471b2
Sperreigner: harry
Sperre erzeugt: 2003-01-15 17:35:12 -0600 (Wed, 15 Jan 2003)
Sperrkommentar (1 Zeile):
My test lock comment
```

Schließlich ist die Ausgabe von **svn info** im XML-Format verfügbar, wenn die Option `--xml` angegeben wird:

```
$ svn info --xml http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<info>
<entry
   kind="dir"
   path="."
   revision="1">
<url>http://svn.red-bean.com/repos/test</url>
<repository>
<root>http://svn.red-bean.com/repos/test</root>
<uuid>5e7d134a-54fb-0310-bd04-b611643e5c25</uuid>
</repository>
<wc-info>
<schedule>normal</schedule>
<depth>infinity</depth>
</wc-info>
<commit
   revision="1">
<author>sally</author>
<date>2003-01-15T23:35:12.847647Z</date>
</commit>
```

```
</entry>
</info>
```

Name

svn list (ls) — Verzeichniseinträge im Projektarchiv auflisten.

Aufruf

```
svn list [TARGET[@REV]...]
```

Beschreibung

Auflisten jeder Datei *TARGET* und den Inhalt jedes Verzeichnisses *TARGET* wie sie im Projektarchiv vorkommen. Falls *TARGET* ein Pfad in der Arbeitskopie ist, wird der entsprechende Projektarchiv-URL verwendet.

Standard für *TARGET* ist „`.`“, d.h. der Projektarchiv-URL des aktuellen Verzeichnisses in der Arbeitskopie.

Mit `--verbose` (`-v`) gibt **svn list** die folgenden Felder für jedes Objekt aus:

- Revisionsnummer der letzten Übergabe
- Autor der letzten Übergabe
- Falls gesperrt, der Buchstabe „O“ (für Details siehe den vorangegangenen Abschnitt über svn info).
- Größe (in Bytes)
- Datum und Zeit der letzten Übergabe

Mit `--xml` erfolgt die Ausgabe im XML-Format (mit einer XML-Deklaration und einem umschließenden Dokumentelement, sofern nicht gleichzeitig `--incremental` angegeben wird). Alle Informationen werden ausgegeben; die Option `--verbose` (`-v`) wird nicht akzeptiert.

Optionen

```
--depth ARG
--incremental
--recursive (-R)
--revision (-r) REV
--verbose (-v)
--xml
```

Beispiele

Am nützlichsten ist **svn list**, falls Sie sehen möchten, welche Dateien in einem Projektarchiv vorhanden sind, ohne eine Arbeitskopie herunterzuladen:

```
$ svn list http://svn.red-bean.com/repos/test/support
README.txt
INSTALL
examples/
...
```

Für zusätzliche Informationen können Sie die Option `--verbose` (`-v`) angeben, ähnlich wie bei dem Unix-Befehl **ls -l**:

```
$ svn list -v file:///var/svn/repos
     16 sally         28361 Jan 16 23:18 README.txt
     27 sally             0 Jan 18 15:27 INSTALL
     24 harry               Jan 18 11:27 examples/
```

Sie können die Ausgabe von **svn list** mit der Option `--xml` auch im XML-Format erhalten:

```
$ svn list --xml http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<lists>
<list
   path="http://svn.red-bean.com/repos/test">
<entry
   kind="dir">
<name>examples</name>
<size>0</size>
<commit
   revision="24">
<author>harry</author>
<date>2008-01-18T06:35:53.048870Z</date>
</commit>
</entry>
...
</list>
</lists>
```

Für weitere Einzelheiten, siehe den vorhergehenden Abschnitt „svn list“.

Name

svn lock — Arbeitskopiepfade oder URLs im Projektarchiv sperren, damit kein anderer Benutzer Änderungen daran übergeben kann.

Aufruf

```
svn lock TARGET...
```

Beschreibung

Jedes *TARGET* sperren. Sollte irgendein *TARGET* bereits durch einen anderen Benutzer gesperrt sein, wird eine Warnung ausgegeben und mit dem Sperren der restlichen *TARGET*s fortgefahren. Verwenden Sie `--force`, um eine Sperre von einem anderen Benutzer oder einer Arbeitskopie zu stehlen.

Optionen

```
--encoding ENC
--file (-F) FILENAME
--force
--force-log
--message (-m) MESSAGE
--targets FILENAME
```

Beispiele

Zwei Dateien in Ihrer Arbeitskopie sperren:

```
$ svn lock tree.jpg house.jpg

»tree.jpg« gesperrt durch »harry«.
»house.jpg« gesperrt durch »harry«.
```

Eine Datei in Ihrer Arbeitskopie sperren, die aktuell durch einen anderen Benutzer gesperrt ist:

```
$ svn lock tree.jpg

svn: Warnung: Pfad »/tree.jpg« ist bereits vom Benutzer »sally« im \
     Dateisystem »/var/svn/repos/db« gesperrt

$ svn lock --force tree.jpg

»tree.jpg« gesperrt durch »harry«.
```

Eine Datei ohne Arbeitskopie sperren:

```
$ svn lock http://svn.red-bean.com/repos/test/tree.jpg

»tree.jpg« gesperrt durch »harry«.
```

Für weitere Einzelheiten siehe „Sperren“.

Name

svn log — Übergabe-Protokollnachrichten anzeigen.

Aufruf

```
svn log [PATH]

svn log URL[@REV] [PATH...]
```

Beschreibung

Zeigt Protokollnachrichten aus dem Projektarchiv an. Falls keine Argumente angegeben werden, zeigt **svn log** die Protokollnachrichten aller Dateien und Verzeichnisse innerhalb (und inklusive) des aktuellen Arbeitsverzeichnisses Ihrer Arbeitskopie an. Sie können das Ergebnis verfeinern, indem Sie einen Pfad, eine oder mehrere Revisionen oder eine beliebige Kombination davon angeben. Der Standardbereich für Revisionen eines lokalen Pfades ist `BASE:1`.

Falls Sie nur einen URL angeben, werden Protokollnachrichten für alles ausgegeben, was unter diesem URL liegt. Falls Sie nach dem URL Pfade angeben, werden nur Nachrichten für diese Pfade unterhalb des URLs ausgegeben. Der Standardbereich für Revisionen eines URLs ist `HEAD:1`.

Mit `--verbose` (`-v`) gibt **svn log** alle betroffenen Pfade zusätzlich zu jeder Protokollnachricht aus. Mit `--quiet` (`-q`) gibt **svn log** den eigentlichen Rumpf der Protokollnachricht nicht aus; das ist kompatibel mit `--verbose` (`-v`).

Jede Protokollnachricht wird nur einmal ausgegeben, auch falls ausdrücklich mehr als einer der betroffenen Pfade für diese Revision angefordert wurde. Die Protokolle folgen standardmäßig der Kopiergeschichte. Verwenden Sie `--stop-on-copy`, um dieses Verhalten abzustellen, was nützlich bei der Ermittlung von Verzweigungspunkten sein kann.

Optionen

```
--change (-c) ARG
--incremental
--limit (-l) NUM
--quiet (-q)
--revision (-r) REV
--stop-on-copy
--targets FILENAME
--use-merge-history (-g)
--verbose (-v)
--with-all-revprops
--with-no-revprops
--with-revprop ARG
--xml
```

Beispiele

Sie können die Protokollnachrichten aller Pfade sehen, die sich in Ihrer Arbeitskopie geändert haben, indem Sie ganz oben **`svn log`** aufrufen:

```
$ svn log
------------------------------------------------------------------------
r20 | harry | 2003-01-17 22:56:19 -0600 (Fri, 17 Jan 2003) | 1 line

Tweak.
------------------------------------------------------------------------
r17 | sally | 2003-01-16 23:21:19 -0600 (Thu, 16 Jan 2003) | 2 lines
…
```

Untersuchen aller Protokollnachrichten für eine bestimmte Datei in Ihrer Arbeitskopie:

```
$ svn log foo.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

Sollten Sie keine Arbeitskopie verfügbar haben, können Sie die Protokolle auch über einen URL abrufen:

```
$ svn log http://svn.red-bean.com/repos/test/foo.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

Wollen Sie mehrere getrennte Pfade unterhalb des gleichen URLs möchten, können Sie die Syntax `URL [PATH...]` verwenden:

```
$ svn log http://svn.red-bean.com/repos/test/ foo.c bar.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r31 | harry | 2003-01-10 12:25:08 -0600 (Fri, 10 Jan 2003) | 1 line

Added new file bar.c
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

Die Option `--verbose` (`-v`) veranlasst **svn log**, Informationen über die in jeder angezeigten Revision geänderten Pfade hinzuzufügen. Diese Pfade, einer pro Ausgabezeile, werden mit einer Aktionskennung markiert, die darüber Aufschluss gibt, um welche Art von Änderung an dem Pfad es sich handelt.

```
$ svn log -v http://svn.red-bean.com/repos/test/ foo.c bar.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Geänderte Pfade:
   M /foo.c

Added defines.
------------------------------------------------------------------------
r31 | harry | 2003-01-10 12:25:08 -0600 (Fri, 10 Jan 2003) | 1 line

Geänderte Pfade:
```

```
   A /bar.c

Added new file bar.c
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

svn log verwendet nur eine handvoll Aktionskennungen, die denjenigen des Befehls **svn update** ähneln:

`A`
: Das Objekt wurde hinzugefügt.

`D`
: Das Objekt wurde gelöscht.

`M`
: Die Eigenschaften oder der textuelle Inhalt des Objektes wurden geändert.

`R`
: Das Objekt wurde am selben Ort durch ein anderes ersetzt.

Zusätzlich zu den Aktionskennungen, die den geänderten Pfaden vorangestellt werden, vermerkt **svn log** mit der Option `--verbose` (`-v`) ob ein Pfad als Ergebnis einer Kopieroperation hinzugefügt oder ersetzt wurde. Dabei wird nach solchen Pfaden (von *`COPY-FROM-PATH`*:*`COPY-FROM-REV`*) ausgegeben.

Falls Sie die Ausgaben mehrerer Aufrufe des Befehls verketten möchten, empfiehlt sich die Verwendung der Option `--incremental`. **svn log** gibt normalerweise zu Beginn einer Protokollnachricht, nach jeder weiteren sowie nach der letzten eine gestrichelte Linie aus. Falls Sie **svn log** über einen Bereich von zwei Revisionen aufgerufen hätten, würden Sie folgendes bekommen:

```
$ svn log -r 14:15
------------------------------------------------------------------------
r14 | …

------------------------------------------------------------------------
r15 | …

------------------------------------------------------------------------
```

Falls Sie jedoch zwei nicht aufeinanderfolgende Protokollnachrichten in einer Datei sammeln möchten, könnten Sie folgendes aufrufen:

```
$ svn log -r 14 > mylog
$ svn log -r 19 >> mylog
$ svn log -r 27 >> mylog
$ cat mylog
------------------------------------------------------------------------
r14 | …

------------------------------------------------------------------------
------------------------------------------------------------------------
r19 | …

------------------------------------------------------------------------
------------------------------------------------------------------------
r27 | …
```

```
------------------------------------------------------------------------
```

Sie können das störende doppelte Auftreten der gestrichelten Linien in der Ausgabe durch die Verwendung der Option `--incremental` verhindern:

```
$ svn log --incremental -r 14 > mylog
$ svn log --incremental -r 19 >> mylog
$ svn log --incremental -r 27 >> mylog
$ cat mylog
------------------------------------------------------------------------
r14 | …

------------------------------------------------------------------------
r19 | …

------------------------------------------------------------------------
r27 | …
```

Die Option `--incremental` bietet bei Verwendung der Option `--xml` eine ähnliche Ausgabekontrolle:

```
$ svn log --xml --incremental -r 1 sandwich.txt
<logentry
   revision="1">
<author>harry</author>
<date>2008-06-03T06:35:53.048870Z</date>
<msg>Initial Import.</msg>
</logentry>
```

Manchmal kann es passieren, dass beim Aufruf von **svn log** für einen bestimmten Pfad und eine bestimmte Revision keine Protokollinformationen ausgegeben werden wie etwa hier:

```
$ svn log -r 20 http://svn.red-bean.com/untouched.txt
------------------------------------------------------------------------
```

Das bedeutet nur, dass der Pfad in dieser Revision nicht geändert wurde. Um Protokollinformationen für diese Revision zu erhalten, sollten Sie den Befehl entweder für die Wurzel des Projektarchivs aufrufen oder einen Pfad angeben, von dem Sie wissen, dass er in dieser Revision geändert wurde:

```
$ svn log -r 20 touched.txt
------------------------------------------------------------------------
r20 | sally | 2003-01-17 22:56:19 -0600 (Fri, 17 Jan 2003) | 1 line

Made a change.
------------------------------------------------------------------------
```

Name

svn merge — Anwenden der Unterschiede zweier Quellen auf einen Pfad der Arbeitskopie.

Aufruf

```
svn merge sourceURL1[@N] sourceURL2[@M] [WCPATH]

svn merge sourceWCPATH1@N sourceWCPATH2@M [WCPATH]

svn merge [[-c M]... | [-r N:M]...] [SOURCE[@REV] [WCPATH]]
```

Beschreibung

In der ersten Form werden für die Quell-URLs die Revisionen *N* und *M* angegeben. Dabei handelt es sich um die zu vergleichenden Quelltexte. Wenn sie nicht angegeben werden, haben die Revisionen den Standardwert HEAD.

In der zweiten Form bestimmen die URLs, die den Arbeitskopiepfaden der Quellen entsprechen, die zu vergleichenden Quelltexte. Die Revisionen müssen angegeben werden.

In der dritten Form kann *SOURCE* entweder ein URL oder ein Arbeitskopiepfad (wobei der entsprechende URL verwendet wird) sein. Falls nicht angegeben, wird für *SOURCE* dasselbe wie für *WCPATH* angenommen. *SOURCE* in Revision *REV* wird verglichen, wie sie zwischen den Revisionen *N* und *M* für jeden angegebenen Revisionsbereich existiert hat. Falls *REV* nicht angegeben ist, wird HEAD angenommen.

`-c M` ist äquivalent zu `-r <M-1>:M`, und `-c -M` macht das Umgekehrte: `-r M:<M-1>`. Falls keine Revisionsbereiche angegeben werden, ist der Standardbereich `1:HEAD`. Mehrere Instanzen von `-c` und/oder `-r` können angegeben werden, und das Mischen von Vorwärts- und Rückwärtsbereichen ist erlaubt – die Bereiche werden intern auf die Minimalrepräsentation gestaucht bevor die Zusammenführung beginnt (was eine Nulloperation ergeben kann).

WCPATH ist der Pfad der Arbeitskopie, auf den die Änderungen angewendet werden. Wird *WCPATH* ausgelassen, wird der Wert „.“ angenommen, es sei denn, die Quellen haben identische Basisnamen, die auf eine Datei innerhalb von „.“ passen. In diesem Fall werden die Änderungen auf diese Datei angewendet.

Subversion zeichnet intern Metadaten über die Zusammenführung nur auf, falls die beiden Quellen abstammungsmäßig in Beziehung stehen – falls die erste Quelle ein Vorgänger der zweiten ist oder umgekehrt. Bei Verwendung der dritten Form ist das gewährleistet. Anders als **svn diff** berücksichtigt der Zusammenführungsbefehl die Herkunft einer Datei beim Zusammenführen. Dies ist sehr wichtig, falls Sie Änderungen von einem Zweig auf einen anderen übertragen und eine Datei auf einem Zweig umbenannt haben, jedoch nicht auf dem anderen.

Optionen

```
--accept ACTION
--change (-c) REV
--depth ARG
--diff3-cmd CMD
--dry-run
--extensions (-x) ARG
--force
--ignore-ancestry
--quiet (-q)
--record-only
--reintegrate
--revision (-r) REV
```

Beispiele

Zurückführen eines Zweigs auf den Stamm (unter der Annahme, dass Sie eine aktuelle Arbeitskopie des Stamms haben):

```
$ svn merge --reintegrate \
            http://svn.example.com/repos/calc/branches/my-calc-branch

--- Zusammenführen der Unterschiede zwischen Projektarchiv-URLs in ».«:
U    button.c
U    integer.c
U    Makefile
 U   .


$ # bauen, testen, verifizieren, ...

$ svn commit -m "Merge my-calc-branch back into trunk!"

Sende          .
Sende          button.c
Sende          integer.c
Sende          Makefile
Übertrage Daten ..
Revision 391 übertragen.
```

Zum Zusammenführen von Änderungen in eine einzelne Datei:

```
$ cd myproj
$ svn merge -r 30:31 thhgttg.txt
U  thhgttg.txt
```

Name

svn mergeinfo — Informationen bezüglich Zusammenführungen abrufen. Für Details, siehe „Zusammenführungsinformation und Vorschauen“.

Aufruf

```
svn mergeinfo SOURCE_URL[@REV] [TARGET[@REV]]
```

Beschreibung

Abrufen von Informationen bezüglich Zusammenführungen (oder potentiellen Zusammenführungen) zwischen *SOURCE-URL* und *TARGET*. Falls die Option `--show-revs` nicht angegeben ist, werden Revisionen angezeigt, die von *SOURCE-URL* nach *TARGET* zusammengeführt wurden. Ansonsten werden entweder zusammengeführte (`merged`) oder zum Zusammenführen in Frage kommende (`eligible`) Revisionen ausgegeben, je nachdem, was bei der Option `--show-revs` angegeben wird.

Optionen

```
--revision (-r) REV
--show-revs ARG
```

Beispiele

Ermittelt, welche Änderungsmengen vom Stamm mit Ihrem Testzweig zusammengeführt wurden:

```
$ svn propget svn:mergeinfo ^/branches/test
/branches/other:3-4
/trunk:11-13,14,16
$ svn mergeinfo --show-revs merged ^/trunk ^/branches/test
r11
r12
r13
r14
r16
$
```

Beachten Sie, dass der Befehl **svn mergeinfo** standardmäßig zusammengeführte Revisionen ausgibt, so dass die Option `--show-revs` nicht unbedingt angegeben werden muss.

Ermittelt, welche Änderungsmengen vom Stamm noch nicht mit Ihrem Testzweig zusammengeführt wurden:

```
$ svn mergeinfo --show-revs eligible ^/trunk ^/branches/test
r15
r17
r20
r21
r22
$
```

Name

svn mkdir — Ein neues Verzeichnis unter Versionskontrolle anlegen.

Aufruf

```
svn mkdir PATH...

svn mkdir URL...
```

Beschreibung

Ein neues Verzeichnis anlegen, dessen Name der letzten Komponente von *PATH* oder *URL* entspricht. Ein durch *PATH* in der Arbeitskopie bezeichnetes Verzeichnis wird zum Hinzufügen in der Arbeitskopie vorgemerkt. Ein durch einen URL bezeichnetes Verzeichnis wird durch eine unmittelbare Übergabe im Projektarchiv erzeugt. Mehrere Verzeichnis-URLs werden atomar übergeben. In beiden Fällen müssen Zwischenverzeichnisse bereits existieren, falls die Option `--parents` nicht angegeben wird.

Optionen

```
--editor-cmd CMD
--encoding ENC
--file (-F) FILENAME
--force-log
--message (-m) MESSAGE
--parents
--quiet (-q)
--with-revprop ARG
```

Beispiele

Ein Verzeichnis in Ihrer Arbeitskopie erzeugen:

```
$ svn mkdir newdir
A         newdir
```

Ein Verzeichnis im Projektarchiv erzeugen (dies ist eine sofortige Übergabe, so dass eine Protokollnachricht erforderlich ist):

```
$ svn mkdir -m "Making a new dir." http://svn.red-bean.com/repos/newdir

Revision 26 übertragen.
```

Name

svn move (mv) — Eine Datei oder ein Verzeichnis verschieben.

Aufruf

```
svn move SRC... DST
```

Beschreibung

Dieser Befehl verschiebt Dateien oder Verzeichnisse in Ihrer Arbeitskopie oder im Projektarchiv.

Dieser Befehl ist äquivalent zu **svn copy** gefolgt von **svn delete**.

Beim Verschieben mehrerer Quellen werden sie als Kinder von *DST* hinzugefügt, das ein Verzeichnis sein muss.

Subversion unterstützt nicht das Verschieben zwischen Arbeitskopien und URLs. Sie können auch Dateien innerhalb eines einzelnen Projektarchivs verschieben – Subversion unterstützt nicht das Verschieben zwischen Projektarchiven. Subversion unterstützt die folgenden Arten von Verschiebungen innerhalb eines einzelnen Projektarchivs:

AK # AK
: Eine Datei oder ein Verzeichnis verschieben und zum Hinzufügen vormerken (mit Geschichte).

URL # URL
: Umbenennung, vollständig serverseitig.

Optionen

```
--editor-cmd CMD
--encoding ENC
--file (-F) FILENAME
--force
--force-log
--message (-m) MESSAGE
--parents
--quiet (-q)
--revision (-r) REV
--with-revprop ARG
```

Beispiele

Eine Datei in Ihrer Arbeitskopie verschieben:

```
$ svn move foo.c bar.c
A         bar.c
D         foo.c
```

Mehrere Dateien in Ihrer Arbeitskopie in ein Unterverzeichnis verschieben:

```
$ svn move baz.c bat.c qux.c src
A         src/baz.c
D         baz.c
A         src/bat.c
D         bat.c
A         src/qux.c
D         qux.c
```

Eine Datei im Projektarchiv verschieben (dies ist eine unmittelbare Übergabe, so dass eine Protokollnachricht erforderlich ist):

```
$ svn move -m "Move a file" http://svn.red-bean.com/repos/foo.c \
                            http://svn.red-bean.com/repos/bar.c

Revision 27 übergeben.
```

Name

svn propdel (pdel, pd) — Eine Eigenschaft von einem Objekt entfernen.

Aufruf

```
svn propdel PROPNAME [PATH...]

svn propdel PROPNAME --revprop -r REV [TARGET]
```

Beschreibung

Entfernt Eigenschaften von Dateien, Verzeichnissen oder Revisionen. Die erste Form entfernt versionierte Eigenschaften in Ihrer Arbeitskopie und die zweite beseitigt unversionierte entfernte Eigenschaften einer Projektarchiv-Revision (*TARGET* bestimmt nur, auf welches Projektarchiv zugegriffen werden soll).

Optionen

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
```

Beispiele

Eine Eigenschaft von einer Datei Ihrer Arbeitskopie entfernen:

```
$ svn propdel svn:mime-type some-script

Eigenschaft »svn:mime-type« wurde von »some-script« gelöscht.
```

Eine Revisions-Eigenschaft löschen:

```
$ svn propdel --revprop -r 26 release-date

Eigenschaft »release-date« wurde von Revision 26 im Projektarchiv gelöscht
```

Name

svn propedit (pedit, pe) — Die Eigenschaft eines oder mehrerer Objekte unter Versionskontrolle bearbeiten. Siehe svn propset (pset, ps) später in diesem Kapitel.

Aufruf

```
svn propedit PROPNAME TARGET...

svn propedit PROPNAME --revprop -r REV [TARGET]
```

Beschreibung

Ein oder mehrere Eigenschaften mit Ihrem Lieblingseditor bearbeiten. Die erste Form bearbeitet versionierte Eigenschaften in Ihrer Arbeitskopie und die zweite unversionierte entfernte Eigenschaften einer Projektarchiv-Revision (*TARGET* bestimmt nur, auf welches Projektarchiv zugegriffen werden soll).

Optionen

```
--editor-cmd CMD
--encoding ENC
--file (-F) FILENAME
--force
--force-log
--message (-m) MESSAGE
--revision (-r) REV
--revprop
--with-revprop ARG
```

Beispiele

svn propedit erleichtert die Änderung von Eigenschaften mit Mehrfachwerten:

```
$ svn propedit svn:keywords foo.c

    # svn startet Ihren Lieblingseditor starten und öffnet eine
    # temporäre Datei mit dem aktuellen Inhalt der Eigenschaft
    # svn:keywords. Sie können einer Eigenschaft einfach
    # Mehrfachwerte hinzufügen, indem Sie pro Zeile einen Wert
    # angeben. Wenn Sie die temporäre Datei sichern und den Editor
    # beenden, wird Subversion die temporäre Datei erneut lesen und
    # ihren aktualisierten Inhalt als den neuen Wert für die
    # Eigenschaft verwenden.
Neuer Wert für Eigenschaft »svn:keywords« für »foo.c« gesetzt
```

Name

svn propget (pget, pg) — Den Wert einer Eigenschaft ausgeben.

Aufruf

```
svn propget PROPNAME [TARGET[@REV]...]

svn propget PROPNAME --revprop -r REV [URL]
```

Beschreibung

Den Wert einer Eigenschaft auf Dateien, Verzeichnissen oder Revisionen ausgeben. Die erste Form gibt die versionierte Eigenschaft eines oder mehrerer Objekte Ihrer Arbeitskopie aus und die zweite unversionierte entfernte Eigenschaften einer Projektarchiv-Revision. Siehe „Eigenschaften“ für weitere Informationen über Eigenschaften.

Optionen

```
--changelist ARG
--depth ARG
--recursive (-R)
--revision (-r) REV
--revprop
--strict
--verbose (-v)
--xml
```

Beispiele

Eine Eigenschaft einer Datei Ihrer Arbeitskopie untersuchen:

```
$ svn propget svn:keywords foo.c
Author
Date
Rev
```

Dasselbe für eine Revisions-Eigenschaft:

```
$ svn propget svn:log --revprop -r 20
Began journal.
```

Verwenden Sie die Option `--verbose` (`-v`) für eine strukturiertere Darstellung der Eigenschaften:

```
$ svn propget svn:keywords foo.c --verbose

Eigenschaften zu Â»foo.cÂ«:
  svn:keywords
    Author
    Date
    Rev
```

Standardmäßig hängt **svn propget** eine abschließende Zeilenende-Sequenz an einen ausgegebenen Eigenschaftswert. Meist ist dieses auch erwünscht, da es die Ausgabe lesbarer macht. Jedoch kann es vorkommen, dass Sie vielleicht den exakten Wert der Eigenschaft bekommen möchten, möglicherweise, weil der Wert nicht textueller Natur ist, sondern ein binäres Format besitzt (wie beispielsweise ein JPEG-Indexbild, das als binärer Eigenschaftswert gespeichert wird). Um die lesbarere Ausgabe von Eigenschaftswerten zu unterbinden, verwenden Sie die Option `--strict`.

Schließlich können Sie Ausgaben von **svn propget** mit der Option `--xml` im XML-Format erhalten:

```
$ svn propget --xml svn:ignore .
<?xml version="1.0"?>
<properties>
<target
   path="">
<property
   name="svn:ignore">*.o
</property>
</target>
</properties>
```

Name

svn proplist (plist, pl) — Alle Eigenschaften auflisten.

Aufruf

```
svn proplist [TARGET[@REV]...]

svn proplist --revprop -r REV [TARGET]
```

Beschreibung

Alle Eigenschaften auf Dateien, Verzeichnissen oder Revisionen auflisten. Die erste Form listet versionierte Eigenschaften in Ihrer Arbeitskopie auf und die zweite unversionierte entfernte Eigenschaften einer Projektarchiv-Revision (*TARGET* bestimmt nur, auf welches Projektarchiv zugegriffen werden soll).

Optionen

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
--verbose (-v)
--xml
```

Beispiele

Sie können **svn proplist** verwenden, um die Eigenschaften eines Objektes Ihrer Arbeitskopie anzusehen:

```
$ svn proplist foo.c

Eigenschaften zu »foo.c«:
  svn:mime-type
  svn:keywords
  owner
```

Mit der Option `--verbose` (`-v`) ist **svn proplist** jedoch sehr praktisch, da es Ihnen auch die Eigenschafts-Werte anzeigt:

```
$ svn proplist -v foo.c
Properties on 'foo.c':
  svn:mime-type
    text/plain
  svn:keywords
    Author Date Rev
  owner
    sally
```

Schließlich können Sie die Ausgabe von **svn proplist** mit der Option `--xml` im XML-Format erhalten:

```
$ svn proplist --xml
<?xml version="1.0"?>
<properties>
<target
   path=".">
<property
   name="svn:ignore"/>
</target>
</properties>
```

Name

svn propset (pset, ps) — Den Wert von *PROPNAME* für Dateien, Verzeichnisse oder Revisionen auf *PROPVAL* setzen.

Aufruf

```
svn propset PROPNAME [PROPVAL | -F VALFILE] PATH...
```

```
svn propset PROPNAME --revprop -r REV [PROPVAL | -F VALFILE] [TARGET]
```

Beschreibung

Setzt den Wert von *PROPNAME* für Dateien, Verzeichnisse oder Revisionen auf *PROPVAL*. Das erste Beispiel erzeugt eine versionierte Eigenschafts-Änderung in der Arbeitskopie und das zweite eine unversionierte, entfernte Eigenschafts-Änderung auf einer Projektarchiv-Revision (*TARGET* bestimmt nur, auf welches Projektarchiv zugegriffen werden soll).

Subversion verfügt über eine Anzahl „besonderer“ Eigenschaften, die sein Verhalten beeinflussen. Siehe „Subversion-Eigenschaften“ später in diesem Kapitel für Weiteres zu diesen Eigenschaften.

Optionen

```
--changelist ARG
--depth ARG
--encoding ENC
--file (-F) FILENAME
--force
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
--targets FILENAME
```

Beispiele

Den MIME-Typen einer Datei setzen:

```
$ svn propset svn:mime-type image/jpeg foo.jpg
Eigenschaft »svn:mime-type« für »foo.jpg« gesetzt
```

Wenn Sie auf einem Unix-System bei einer Datei die Ausführbarkeitsberechtigung setzen wollen:

```
$ svn propset svn:executable ON somescript
Eigenschaft »svn:executable« für »somescript« gesetzt
```

Vielleicht haben Sie eine interne Vereinbarung, bestimmte Eigenschaften zum Nutzen Ihrer Mitarbeiter zu setzen:

```
$ svn propset owner sally foo.c
```

```
Eigenschaft »owner« für »foo.c« gesetzt
```

Falls Sie einen Fehler in einer Protokollnachricht einer bestimmten Revision gemacht haben und sie nun ändern wollen, verwenden Sie `--revprop` und setzen Sie den Wert von `svn:log` auf die neue Nachricht:

```
$ svn propset --revprop -r 25 svn:log "Journaled about trip to New York."
Eigenschaft »svn:log« wurde für Revision 25 im Projektarchiv gesetzt

$ svn propset --revprop -r 25 svn:log "Journaled about trip to New York."
```

Wenn Sie keine Arbeitskopie haben, können Sie einen URL angeben:

```
$ svn propset --revprop -r 26 svn:log "Document nap." \
              http://svn.red-bean.com/repos
Eigenschaft »svn:log« wurde für Revision 25 im Projektarchiv gesetzt
```

Schließlich können Sie **svn propset** mitteilen, seine Eingaben aus einer Datei zu holen. Sie können es sogar verwenden, um den Inhalt einer Eigenschaft auf einen binären Wert zu setzen:

```
$ svn propset owner-pic -F sally.jpg moo.c
Eigenschaft »owner-pic« für »moo.c« gesetzt\n"
```

Standardmäßig können Sie Revisions-Eigenschaften in einem Subversion-Projektarchiv nicht ändern. Der Administrator des Projektarchivs muss die Änderung von Revisions-Eigenschaften ausdrücklich erlauben, indem er einen Hook namens `pre-revprop-change` erstellt. Siehe „Erstellen von Projektarchiv-Hooks“ für weiterführende Informationen zu Hook-Skripten.

Name

svn resolve — Konflikte in Dateien und Verzeichnissen der Arbeitskopie auflösen.

Aufruf

```
svn resolve PATH...
```

Beschreibung

Den „Konfliktzustand“ von Dateien oder Verzeichnissen der Arbeitskopie auflösen. Diese Routine löst Konfliktmarken zwar nicht semantisch auf, ersetzt jedoch *PATH* durch die Version, die bei `--accept` angegeben ist, und anschließend werden konfliktbezogene Dateiartefakte gelöscht. Hierdurch wird ermöglicht, dass *PATH* noch einmal übergeben werden kann – d.h., Subversion wird mitgeteilt, dass die Konflikte „aufgelöst“ wurden. Je nachdem wie Sie Ihren Konflikt auflösen wollen, können Sie der Option `--accept` die folgenden Argumente mitgeben:

`base`
: Auswahl der Datei, die die BASE-Revision gewesen war, bevor Sie Ihre Arbeitskopie aktualisierten. Das heißt, die Datei, die Sie ausgecheckt hatten, bevor Sie Ihre letzten Änderungen vornahmen.

`working`
: Auswahl der aktuellen Datei in Ihrer Arbeitskopie unter der Annahme, dass Sie Konflikte manuell aufgelöst haben.

`mine-full`
: Auswahl der Kopien konfliktbehafteter Dateien, mit dem Inhalt zum Zeitpunkt unmittelbar vor Ihrem Aufruf von **svn update**.

`theirs-full`
: Auswahl der Kopien konfliktbehafteter Dateien, mit dem Inhalt der Revisionen, die Sie durch den Aufruf von **svn update** vom Server geholt haben.

Siehe „Lösen Sie etwaige Konflikte auf“ für eine tiefgehende Erörterung der Konfliktauflösung.

Optionen

```
--accept ACTION
--depth ARG
--quiet (-q)
--recursive (-R)
--targets FILENAME
```

Beispiele

In diesem Beispiel ersetzt **svn resolve** nach dem Aufschieben der Konfliktauflösung während der Aktualisierung alle Konflikte in `foo.c` mit Ihren Änderungen:

```
$ svn update

Konflikt in »foo.c« entdeckt.
Auswahl: (p) zurückstellen, (df) voller Diff, (e) editieren,
         (mc) eigene konfliktbehaftete Datei, (tc) fremde konfliktbehaftete Datei,
         (s) alle Optionen anzeigen: p
C    foo.c
Aktualisiert zu Revision 5.
Konfliktübersicht:
   Textkonflikte: 1
```

```
$ svn resolve --accept mine-full foo.c

Konflikt von »foo.c« aufgelöst
$
```

Name

svn resolved — *Abgeraten*. Den „Konfliktzustand“ einer Datei oder eines Verzeichnisses der Arbeitskopie beenden.

Aufruf

```
svn resolved PATH...
```

Beschreibung

Von diesem Befehl wird abgeraten; verwenden Sie stattdessen **`svn resolve --accept working`** ***`PATH`***. Siehe svn resolve im vorhergehenden Abschnitt für Details.

Beenden des „Konfliktzustandes“ einer Datei oder eines Verzeichnisses der Arbeitskopie. Diese Routine löst Konfliktmarken nicht semantisch auf sondern entfernt lediglich konfliktbezogene Dateiartefakte und erlaubt es, *`PATH`* erneut zu übergeben, d.h., Subversion wird mitgeteilt, dass die Konflikte „aufgelöst“ wurden. Siehe „Lösen Sie etwaige Konflikte auf“ für eine tiefgehende Erörterung der Konfliktauflösung.

Optionen

```
--depth ARG
--quiet (-q)
--recursive (-R)
--targets FILENAME
```

Beispiele

Falls Sie bei einer Aktualisierung einen Konflikt erhalten, werden drei neue Dateien in Ihrer Arbeitskopie angelegt:

```
$ svn update
C  foo.c

Aktualisiert zu Revision 31.
Konfliktübersicht:
   Textkonflikte: 1
$ ls
foo.c
foo.c.mine
foo.c.r30
foo.c.r31
$
```

Sobald Sie die Konflikte aufgelöst haben und die Datei `foo.c` bereit zur Übergabe ist, rufen Sie **svn resolved** auf, damit Ihre Arbeitskopie weiß, dass Sie sich um alles gekümmert haben.

Sie *können* zwar bloß die Konfliktdateien löschen und eine Übergabe machen, jedoch bereinigt **svn resolved** zusätzlich zum Löschen der Konfliktdateien einige Kontrolldaten im Verwaltungsbereich der Arbeitskopie, so dass wir Ihnen nahelegen, stattdessen diesen Befehl zu verwenden.

Name

svn revert — Alle lokalen Änderungen rückgängig machen.

Aufruf

```
svn revert PATH...
```

Beschreibung

Macht alle lokalen Änderungen an einer Datei oder einem Verzeichnis rückgängig und löst etwaige Konfliktzustände auf. **svn revert** macht nicht nur inhaltliche Änderungen eines Objektes in der Arbeitskopie rückgängig sondern auch Änderungen an Eigenschaften. Schließlich können Sie hiermit etwaige geplante Operationen zurücknehmen (z.B. kann die Markierung von zum Hinzufügen oder Löschen vorgemerkten Dateien wieder entfernt werden).

Optionen

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--targets FILENAME
```

Beispiele

Änderungen an einer Datei verwerfen:

```
$ svn revert foo.c
```

Falls Sie die Änderungen eines ganzen Verzeichnisbaums rückgängig machen wollen, verwenden Sie die Option `--depth=infinity`:

```
$ svn revert --depth=infinity .

Rückgängig gemacht: newdir/afile
Rückgängig gemacht: foo.c
Rückgängig gemacht: bar.txt
```

Schließlich können Sie die Markierung für geplante Operationen entfernen:

```
$ svn add mistake.txt whoops
A         mistake.txt
A         whoops
A         whoops/oopsie.c

$ svn revert mistake.txt whoops

Rückgängig gemacht: mistake.txt
Rückgängig gemacht: whoops
```

```
$ svn status
?       mistake.txt
?       whoops
```

svn revert ist von Natur aus gefährlich, da sein einziger Zweck das Beseitigen von Daten ist – nämlich Ihre noch nicht übergebenen Änderungen. Sobald Sie irgendetwas rückgängig gemacht haben, bietet Ihnen Subversion *keine Möglichkeit*, wieder an die noch nicht übergebenen Änderungen heranzukommen.

Falls Sie **svn revert** keine Zielobjekte mitgeben, macht es nichts. Um Sie vor dem versehentlichen Verlust von Änderungen Ihrer Arbeitskopie zu bewahren, erwartet **svn revert**, dass Sie ausdrücklich mindestens ein Zielobjekt angeben.

Name

svn status (stat, st) — Ausgabe des Zustands von Dateien und Verzeichnissen der Arbeitskopie.

Aufruf

```
svn status [PATH...]
```

Beschreibung

Gibt den Zustand von Dateien und Verzeichnissen der Arbeitskopie aus. Ohne Argumente werden nur lokal geänderte Objekte ausgegeben (ohne Zugriff auf das Projektarchiv). Mit `--show-updates` (`-u`) werden Informationen über die Arbeitsrevision und in Bezug auf den Server nicht mehr aktuelle Revisionen angezeigt. Mit `--verbose` (`-u`) werden vollständige Revisionsinformationen für jedes Objekt ausgegeben. Mit `--quiet` (`-u`) wird nur eine Zusammenfassung über lokal geänderte Objekte ausgegeben.

Die ersten sieben Spalten der Ausgabe sind jeweils ein Zeichen breit und beinhalten Informationen über einen unterschiedlichen Aspekt jedes Objektes der Arbeitskopie.

Die erste Spalte gibt Auskunft darüber, ob ein Objekt hinzugefügt, gelöscht oder anderweitig geändert wurde:

`' '`
: Keine Änderungen.

`'A'`
: Objekt ist zum Hinzufügen vorgemerkt.

`'D'`
: Objekt ist zum Löschen vorgemerkt.

`'M'`
: Objekt wurde geändert.

`'R'`
: Objekt wurde in Ihrer Arbeitskopie ersetzt. Das bedeutet, dass die Datei zum Löschen und an deren Stelle eine neue Datei gleichen Namens zum Hinzufügen vorgemerkt wurde.

`'C'`
: Der Inhalt (im Gegensatz zu den Eigenschaften) des Objektes steht in Konflikt mit dem aktualisierten Inhalt aus dem Projektarchiv.

`'X'`
: Objekt ist aufgrund einer externals-Definition vorhanden.

`'I'`
: Objekt wird ignoriert (d.h., durch die Eigenschaft `svn:ignore`).

`'?'`
: Objekt ist nicht unter Versionskontrolle.

`'!'`
: Objekt wird vermisst (d.h., Sie haben es verschoben oder gelöscht, ohne **svn** zu verwenden). Das deutet auch darauf hin, dass ein Verzeichnis unvollständig ist (ein Checkout oder eine Aktualisierung wurde unterbrochen).

`'~'`
: Objekt ist als eine bestimmte Objektart versioniert (Datei, Verzeichnis, Link), wurde jedoch durch eine andere Objektart ersetzt.

Die zweite Spalte gibt Auskunft über die Eigenschaften einer Datei oder eines Verzeichnisses:

`' '`
Keine Änderungen.

`'M'`
Eigenschaften dieses Objektes wurden geändert.

`'C'`
Eigenschaften dieses Objektes stehen in Konflikt mit Eigenschafts-Aktualisierungen aus dem Projektarchiv.

Die dritte Spalte wird nur gefüllt, falls das Verzeichnis der Arbeitskopie gesperrt ist (siehe „Manchmal müssen Sie einfach nur aufräumen“):

`' '`
Objekt ist nicht gesperrt.

`'L'`
Objekt ist gesperrt.

Die vierte Spalte wird nur gefüllt, falls das Objekt zum Hinzufügen samt Geschichte vorgemerkt ist:

`' '`
Geschichte soll nicht übergeben werden.

`'+'`
Geschichte soll übergeben werden.

Die fünfte Spalte wird nur gefüllt, falls das Objekt relativ zu seinem Elternobjekt auf einem anderen Zweig liegt (siehe „Zweige durchlaufen“):

`' '`
Objekt ist Kind des Elternverzeichnisses.

`'S'`
Objekt kommt von einem anderen Zweig.

Die sechste Spalte wird mit Informationen zu Sperren gefüllt:

`' '`
Falls `--show-updates` (`-u`) verwendet wird, ist die Datei nicht gesperrt. Falls `--show-updates` (`-u`) *nicht* verwendet wird, bedeutet es lediglich, dass die Datei nicht in der Arbeitskopie gesperrt ist.

K
Datei ist in dieser Arbeitskopie gesperrt.

O
Die Datei ist entweder durch einen anderen Benutzer oder in einer anderen Arbeitskopie gesperrt. Das erscheint nur, falls `--show-updates` (`-u`) verwendet wird.

T
Die Datei war in dieser Arbeitskopie gesperrt worden, die Sperre wurde jedoch „gestohlen“ und ist ungültig. Die Datei ist momentan im Projektarchiv gesperrt. Das erscheint nur, falls `--show-updates` (`-u`) verwendet wird.

B
Die Datei war in dieser Arbeitskopie gesperrt worden, die Sperre wurde jedoch „aufgebrochen“ und ist ungültig. Die Datei ist nicht mehr gesperrt. Das erscheint nur, falls `--show-updates` (`-u`) verwendet wird.

Die siebte Spalte ist nur dann belegt, falls das Objekt das Opfer eines Baumkonfliktes ist:

`' '`
Objekt ist nicht das Opfer eines Baumkonfliktes.

`'C'`
Objekt ist das Opfer eines Baumkonfliktes.

Die achte Spalte ist immer leer.

Aktualitätsinformation erscheint in der neunten Spalte (nur wenn Sie die Option `--show-updates` (`-u`) angeben):

`' '`
Das Objekt in Ihrer Arbeitskopie ist aktuell.

`'*'`
Auf dem Server ist eine neuere Revision verfügbar.

Die übrigen Felder haben eine variable Breite und werden durch Leerzeichen begrenzt. Das nächste Feld gibt die Arbeitsrevision an, falls die Option `--show-updates` (`-u`) oder `--verbose` (`-v`) angegeben wird.

Falls die Option `--verbose` (`-v`) angegeben wird, folgt die letzte übergebene Revision und der Autor derselben.

Der Pfad der Arbeitskopie ist stets das letzte Feld, so dass es auch Leerzeichen enthalten kann.

Optionen

```
--changelist ARG
--depth ARG
--ignore-externals
--incremental
--no-ignore
--quiet (-q)
--show-updates (-u)
--verbose (-v)
--xml
```

Beispiele

Auf diese Weise lässt sich am einfachsten herausfinden, welche Änderungen Sie in der Arbeitskopie gemacht haben:

```
$ svn status wc
 M      wc/bar.c
A  +    wc/qax.c
```

Falls Sie herausfinden möchten, welche Dateien Ihrer Arbeitskopie nicht mehr aktuell sind, geben Sie die Option `--show-updates` (`-u`) mit (es werden *keine* Änderungen an der Arbeitskopie vorgenommen). Hier können Sie sehen, dass sich `wc/foo.c` seit unserer letzten Aktualisierung im Projektarchiv geändert hat:

```
$ svn status -u wc
 M            965    wc/bar.c
        *     965    wc/foo.c
```

```
A  +          965    wc/qax.c

Status bezogen auf Revision:    981
```

`--show-updates` (`-u`) fügt *nur* Objekten ein Sternchen hinzu, falls sie nicht mehr aktuell sind (d.h., Objekte die aus dem Projektarchiv beim nächsten **svn update** aktualisiert würden). `--show-updates` (`-u`) veranlasst die Ausgabe *nicht*, die Projektarchiv-Revision des Objektes anzuzeigen (Sie können die Revisionsnummer im Projektarchiv jedoch sehen, indem Sie die Option `--verbose` (`-v`) angeben).

Die umfangreichste Information bekommen Sie wie folgt:

```
$ svn status -u -v wc
 M            965       938 sally        wc/bar.c
        *     965       922 harry        wc/foo.c
A  +          965       687 harry        wc/qax.c
              965       687 harry        wc/zig.c

Status bezogen auf Revision:   981
```

Schließlich können Sie die Ausgabe von **svn status** mit der Option `--xml` im XML-Format erhalten:

```
$ svn status --xml wc
<?xml version="1.0"?>
<status>
<target
   path="wc">
<entry
   path="qax.c">
<wc-status
   props="none"
   item="added"
   revision="0">
</wc-status>
</entry>
<entry
   path="bar.c">
<wc-status
   props="normal"
   item="modified"
   revision="965">
<commit
   revision="965">
<author>sally</author>
<date>2008-05-28T06:35:53.048870Z</date>
</commit>
</wc-status>
</entry>
</target>
</status>
```

Für wesentlich mehr Beispiele von **svn status**, siehe „Verschaffen Sie sich einen Überblick über Ihre Änderungen“.

Name

svn switch (sw) — Arbeitskopie auf einen anderen URL aktualisieren.

Aufruf

```
svn switch URL[@PEGREV] [PATH]

switch --relocate FROM TO [PATH...]
```

Beschreibung

Die erste Variante dieses Unterbefehls (ohne die Option `--relocate`) aktualisiert Ihre Arbeitskopie so, dass sie auf einen neuen URL zeigt – normalerweise ein URL, der, allerdings nicht notwendigerweise, einen gemeinsamen Vorgänger mit Ihrer Arbeitskopie hat. Auf diese Weise lässt Subversion eine Arbeitskopie einen neuen Zweig verfolgen. Wenn *`PEGREV`* angegeben wird, bezeichnet es die Revision, bei der das Ziel zuerst gesucht wird. Siehe „Zweige durchlaufen“ für eine detaillierte Betrachtung des Umschaltens.

Wird die Option `--force` verwendet, verursachen unversionierte Pfade, die sich beim Versuch umzuschalten im Weg befinden, nicht automatisch einen Fehler, falls durch das Umschalten versucht wird, den selben Pfad anzulegen. Wen der im Weg liegende Pfad den selben Typ (Datei oder Verzeichnis) wie der entsprechende Pfad im Projektarchiv hat, wird er versioniert, der Inhalt bleibt jedoch in der Arbeitskopie unverändert. Das bedeutet, dass die Kindelemente eines sich im Weg befindlichen Verzeichnisses ebenfalls im Weg befinden und versioniert werden können. Alle inhaltlichen Unterschiede von sich im Weg befindlichen Dateien zum Projektarchiv werden als lokale Änderung an der Arbeitskopie betrachtet. Alle Eigenschaften aus dem Projektarchiv werden auf den sich im Weg befindlichen Pfad angewendet.

Wie bei den meisten Unterbefehlen können Sie den Wirkbereich des Umschaltbefehls mit der Option `--depth` auf einen bestimmten Baum beschränken. Alternativ können Sie die Option `--set-depth` verwenden, um eine neue Wirktiefe für das Umschaltziel in der Arbeitskopie festzulegen.

Die Option `--relocate` veranlasst **svn switch**, etwas ganz anderes zu machen: es aktualisiert Ihre Arbeitskopie so, dass es auf *das gleiche* Verzeichnis im Projektarchiv zeigt, allerdings unter einem anderen Projektarchiv-URL (typischerweise weil ein Administrator das Projektarchiv entweder auf einen anderen Server verschoben oder den URL geändert hat).

Optionen

```
--accept ACTION
--depth ARG
--diff3-cmd CMD
--force
--ignore-externals
--quiet (-q)
--relocate
--revision (-r) REV
--set-depth ARG
```

Beispiele

Falls Sie sich momentan innerhalb des Verzeichnisses `vendors` befinden, das nach `vendors-with-fix` abgezweigt wurde, und Sie Ihre Arbeitskopie nun auf diesen Zweig umschalten möchten:

```
$ svn switch http://svn.red-bean.com/repos/branches/vendors-with-fix .

Aktualisiert zu Revision 31.
```

Um zurückzuschalten, brauchen Sie nur den URL des Ortes im Projektarchiv anzugeben, von dem Sie ursprünglich Ihre

Arbeitskopie ausgecheckt haben:

```
$ svn switch http://svn.red-bean.com/repos/trunk/vendors .
U    myproj/foo.txt
U    myproj/bar.txt
U    myproj/baz.c
U    myproj/qux.c

Aktualisiert zu Revision 31.
```

Sie können auch nur einen Teil Ihrer Arbeitskopie auf einen Zweig umschalten, falls Sie nicht Ihre gesamte Arbeitskopie nehmen wollen.

Manchmal kann es vorkommen, dass ein Administrator den Ort (oder den scheinbaren Ort) Ihres Projektarchivs ändert – mit anderen Worten: der Inhalt des Projektarchivs ändert sich nicht, wohl aber der Wurzel-URL des Projektarchivs. So kann sich beispielsweise der Name des Wirtsrechners, das URL-Schema oder irgend ein Teil des URL, der auf das Projektarchiv zeigt, ändern. Statt eine neue Arbeitskopie auszuchecken, können Sie den Befehl **svn switch** dazu verwenden, die Metadaten im Verwaltungsbereich Ihrer Arbeitskopie „überschreiben“ zu lassen, damit diese auf den neuen Ort des Projektarchivs verweisen. Falls Sie **svn switch** die Option `--relocate` mitgeben, nimmt Subversion Verbindung mit dem Projektarchiv auf, um die Gültigkeit der Anfrage zu bestätigen (indem es im Projektarchiv natürlich unter dem neuen URL nachfragt) und anschließend die Metadaten zu überschreiben. Durch diese Operation werden keinerlei Dateiinhalte verändert – es werden lediglich die Metadaten der Arbeitskopie verändert.

```
$ svn checkout file:///var/svn/repos test
A    test/a
A    test/b
…

$ mv /var/svn/repos /var/svn/newlocation

$ svn update test/

svn: Kann keine ra_local-Verbindung zu einer URL aufbauen
svn: Projektarchiv »file:///var/svn/repos« kann nicht geöffnet werden

$ svn switch --relocate file:///var/svn/repos \
                        file:///var/svn/tmp/newlocation test/

$ svn update test/

Revision 3.
```

Seien Sie vorsichtig, wenn Sie die Option `--relocate` verwenden. Falls Sie sich beim Argument vertippen, kann es dazu führen, dass Sie in der Arbeitskopie unsinnige URLs erzeugen, die Ihren Arbeitsbereich unbrauchbar machen, so dass er nur schwer wiederherzustellen ist. Es ist auch wichtig zu verstehen, wann `--relocate` zu verwenden ist und wann nicht. Hier ist die Faustregel:

- Falls die Arbeitskopie ein neues Verzeichnis *innerhalb* des Projektarchivs repräsentieren soll, verwenden Sie nur **svn switch**.
- Falls die Arbeitskopie das gleiche Verzeichnis im Projektarchiv repräsentiert, sich jedoch der Ort des Projektarchivs selbst geändert hat, verwenden Sie **svn switch** mit der Option `--relocate`.

Name

svn unlock — Arbeitskopiepfade oder URLs entsperren,

Aufruf

```
svn unlock TARGET...
```

Beschreibung

Die Sperre für jedes *TARGET* aufheben. Falls irgend ein *TARGET* durch einen anderen Benutzer gesperrt ist oder in der Arbeitskopie keine gültige Sperrmarke existiert, wird eine Warnung ausgegeben und die verbleibenden *TARGET*s entsperrt. Verwenden Sie `--force`, um eine Sperre eines anderen Benutzers oder einer anderen Arbeitskopie aufzubrechen.

Optionen

```
--force
--targets FILENAME
```

Beispiele

Zwei Dateien in Ihrer Arbeitskopie entsperren:

```
$ svn unlock tree.jpg house.jpg

»tree.jpg« freigegeben.
»house.jpg« freigegeben.
```

Entsperren einer Datei in Ihrer Arbeitskopie, die momentan durch einen anderen Benutzer gesperrt ist:

```
$ svn unlock tree.jpg

svn: »tree.jpg« ist in dieser Arbeitskopie nicht gesperrt
$ svn unlock --force tree.jpg

»tree.jpg« freigegeben.
```

Entsperren einer Datei ohne Arbeitskopie:

```
$ svn unlock http://svn.red-bean.com/repos/test/tree.jpg

»tree.jpg« freigegeben.
```

Näheres unter „Sperren“.

Name

svn update (up) — Aktualisieren Ihrer Arbeitskopie.

Aufruf

```
svn update [PATH...]
```

Beschreibung

svn update holt Änderungen aus dem Projektarchiv in Ihre Arbeitskopie. Falls keine Revision angegeben ist, wird Ihre Arbeitskopie relativ zur Revision `HEAD` aktualisiert. Ansonsten wird Ihre Arbeitskopie mit der Revision synchronisiert, die bei der Option `--revision` (`-r`) angegeben ist. Während der Synchronisierung entfernt **svn update** auch veraltete Sperren (siehe „Manchmal müssen Sie einfach nur aufräumen“) aus der Arbeitskopie.

Für jedes aktualisierte Objekt wird eine Zeile ausgegeben, die mit einem Zeichen beginnt, das Auskunft über die vorgenommene Aktion gibt. Die Zeichen haben die folgende Bedeutung:

`A`
: Hinzugefügt

`B`
: Aufgebrochene Sperre (nur in der dritten Spalte)

`D`
: Gelöscht

`U`
: Aktualisiert

`C`
: In Konflikt

`G`
: Zusammengeführt

`E`
: Existierte

Ein Zeichen in der ersten Spalte zeigt eine Aktualisierung für die eigentliche Datei an, während Aktualisierungen für die Eigenschaften einer Datei in der zweiten Zeile angezeigt werden. Informationen zu Sperren werden in der dritten Spalte ausgegeben.

Wie bei den meisten Unterbefehlen können Sie den Wirkbereich des Umschaltbefehls mit der Option `--depth` auf einen bestimmten Baum beschränken. Alternativ können Sie die Option `--set-depth` verwenden, um eine neue Wirktiefe für das Umschaltziel in der Arbeitskopie festzulegen.

Optionen

```
--accept ACTION
--changelist
--depth ARG
--diff3-cmd CMD
--editor-cmd CMD
--force
--ignore-externals
--quiet (-q)
--revision (-r) REV
--set-depth ARG
```

Beispiele

Abholen der Änderungen seit Ihrer letzten Aktualisierung aus dem Projektarchiv:

```
$ svn update
A    newdir/toggle.c
A    newdir/disclose.c
A    newdir/launch.c
D    newdir/README

Aktualisiert zu Revision 32.
```

Sie können Ihre Arbeitskopie mit „update“ auch auf eine ältere Revision aktualisieren (Subversion kennt keine „klebrigen“ Dateien wie CVS; siehe Anhang B, *Subversion für CVS-Benutzer*):

```
$ svn update -r30
A    newdir/README
D    newdir/toggle.c
D    newdir/disclose.c
D    newdir/launch.c
U    foo.c

Aktualisiert zu Revision 30.
```

Falls Sie eine ältere Revision einer einzelnen Datei untersuchen möchten, verwenden Sie stattdessen besser **svn cat** – hierdurch wird Ihre Arbeitskopie nicht verändert.

svn update ist ebenfalls das Mittel der Wahl bei der Einrichtung von Arbeitskopien mit teilweisen Checkouts. Wird der Befehl mit der Option `--set-depth` aufgerufen, lässt die Aktualisierung individuelle Elemente aus oder fügt sie hinzu, indem die verzeichnete Tiefe aus ihrer Umgebung auf die von Ihnen angegebene Tiefe angepasst wird (wobei nötigenfalls Informationen aus dem Projektarchiv abgerufen wird). Mehr zu teilweise ausgecheckten Verzeichnissen unter „Verzeichnis-Teilbäume“.

svnadmin – Subversion Projektarchiv-Verwaltung

svnadmin ist das Verwaltungswerkzeug zum Überwachen und Reparieren Ihres Subversion-Projektarchivs. Detaillierte Informationen zur Verwaltung von Projektarchiven finden Sie im Abschnitt zur Wartung für „svnadmin“.

Da **svnadmin** über direkten Projektarchiv-Zugriff arbeitet (und somit nur auf der Maschine verwendet werden kann, auf der sich das Projektarchiv befindet), greift es auf das Projektarchiv mittels eines Pfades statt eines URLs zu.

svnadmin-Optionen

Optionen für **svmadin** sind global, genau so wie in **svn**:

`--bdb-log-keep`
: (Spezifisch für Berkeley DB.) Verhindert, dass Protokolldateien der Datenbank automatisch entfernt werden. Bei der Wiederherstellung nach einem katastrophalen Fehler kann es nützlich sein, auf diese Protokolldateien zurückzugreifen.

`--bdb-txn-nosync`

(Spezifisch für Berkeley DB.) Verhindert fsync bei der Übergabe von Datenbanktransaktionen. In Verbindung mit dem Befehl **svnadmin create** verwendet, um ein Berkeley-DB-basiertes Projektarchiv mit aktiviertem `DB_TXN_NOSYNC` zu erstellen (was zu mehr Schnelligkeit führt, jedoch einige Risiken birgt).

`--bypass-hooks`
Das Hook-System des Projektarchivs umgehen.

`--clean-logs`
Nicht benötigte Protokolldateien von Berkeley DB entfernen.

`--config-dir` *`DIR`*
Veranlasst Subversion, Informationen zur Konfiguration aus dem angegebenen Verzeichnis zu lesen, statt aus dem standardmäßigen Ort (`.subversion` im Heimatverzeichnis des Anwenders.

`--deltas`
Bei der Erstellung einer Auszugsdatei sollen die Änderungen an versionierten Eigenschaften und Dateiinhalten als Deltas zum vorherigen Zustand angegeben werden.

`--fs-type` *`ARG`*
Bei der Erstellung eines Projektarchivs soll *`ARG`* als gewünschter Dateisystem-Typ verwendet werden. *`ARG`* kann entweder `bdb` oder `fsfs` sein.

`--force-uuid`
Beim Laden von Daten in ein Projektarchiv, das bereits Revisionen enthält, ignoriert **svnadmin** standardmäßig die UUID aus dem Auszugs-Datenstrom. Diese Option führt dazu, dass die UUID des Projektarchivs auf die UUID des Datenstroms gesetzt wird.

`--ignore-uuid`
Beim Laden von Daten in ein leeres Projektarchivs setzt **svnadmin** standardmäßig die UUID des Projektarchivs auf die UUID des Auszugs-Datenstroms. Diese Option erzwingt, dass die UUID aus dem Datenstrom ignoriert wird,

`--incremental`
Ein Auszug enthält nur die Unterschiede zur Vorgängerrevision anstatt des kompletten Textes.

`--parent-dir` *`DIR`*
Beim Laden einer Auszugsdatei werden Pfade unter *`DIR`* statt unter `/` eingehängt.

`--pre-1.4-compatible`
Verwendet beim Erstellen eines Projektarchiv ein Format, das zu älteren Versionen als Subversion 1.4 kompatibel ist.

`--pre-1.5-compatible`
Verwendet beim Erstellen eines Projektarchiv ein Format, das zu älteren Versionen als Subversion 1.5 kompatibel ist.

`--pre-1.6-compatible`
Verwendet beim Erstellen eines Projektarchiv ein Format, das zu älteren Versionen als Subversion 1.5 kompatibel ist.

`--revision` (`-r`) *`ARG`*
Gibt eine bestimmte Revision an, mit der gearbeitet werden soll.

`--quiet` (`-q`)
Zeigt nicht den normalen Fortgang an — lediglich Fehler.

`--use-post-commit-hook`
Beim Laden einer Auszugsdatei wird der `post-commit`-Hook des Projektarchivs nach Fertigstellung jeder neu geladenen Revision aufgerufen.

`--use-post-revprop-change-hook`
Beim Ändern einer Revisions-Eigenschaft wird anschließend der `post-revprop-change`-Hook des Projektarchivs aufgerufen.

`--use-pre-commit-hook`
Beim Laden einer Auszugsdatei wird vor der Abschlussbehandlung jeder neu geladenen Revision der `pre-commit`-Hook des Projektarchiv ausgeführt. Falls der Hook fehlschlägt, wird die Übergabe abgebrochen und der Ladeprozess beendet.

`--use-pre-revprop-change-hook`
Beim Ändern einer Revisions-Eigenschaft wird vorher der `pre-revprop-change`-Hook des Projektarchivs aufgerufen. Falls der Hook fehlschlägt, wird die Änderung abgebrochen und beendet.

`--wait`
Bei Operationen, die einen exklusiven Zugriff auf das Projektarchiv erfordern, soll gewartet werden, bis die benötigte Sperre des Projektarchivs verfügbar ist, anstatt bei Nichtverfügbarkeit sofort mit einer Fehlermeldung abzubrechen.

svnadmin-Unterbefehle

Hier sind die verschiedenen Unterbefehle des Programms **svnadmin**.

Name

svnadmin crashtest — Simuliert einen abstürzenden Prozess.

Aufruf

```
svnadmin crashtest REPOS_PATH
```

Beschreibung

Öffnet das Projektarchiv bei *REPOS_PATH* und bricht dann ab; somit wird ein abstürzender Prozess simuliert, der eine offene Verbindung zum Projektarchiv hält. Dieser Befehl wird verwendet, um die automatische Wiederherstellung des Projektarchivs zu testen (eine Neuerung in Berkeley DB 4.4). Es ist eher unwahrscheinlich, dass Sie diesen Befehl aufrufen müssen.

Optionen

Keine

Beispiele

```
$ svnadmin crashtest /var/svn/repos
Aborted
```

Hochinteressant, nicht wahr?

Name

svnadmin create — Erstellt ein neues, leeres Projektarchiv.

Aufruf

```
svnadmin create REPOS_PATH
```

Beschreibung

Am angegebenen Ort wird ein neues, leeres Projektarchiv erstellt. Falls das angegebene Verzeichnis nicht vorhanden ist, wird es für Sie angelegt.[1] Seit Subversion 1.2 basieren die von **svnadmin** erstellten neuen Projektarchive standardmäßig auf dem FSFS-Dateisystem.

Obwohl **svnadmin create** das Basisverzeichnis für ein neues Projektarchiv erzeugt, werden keine zwischenliegenden Verzeichnisse angelegt. Wenn Sie beispielsweise ein leeres Verzeichnis `/var/svn` haben, wird das Anlegen von `/var/svn/repos` funktionieren, während der Versuch, `/var/svn/subdirectory/repos` anzulegen, mit einem Fehler abbricht. Denken Sie auch daran, dass Sie eventuell **svnadmin** als Anwender mit erweiterten Rechten ausführen müssen (wie etwa `root`), abhängig davon, wo Sie das Projektarchiv auf Ihrem System erstellen.

Optionen

```
--bdb-log-keep
--bdb-txn-nosync
--config-dir DIR
--fs-type TYPE
--pre-1.4-compatible
--pre-1.5-compatible
--pre-1.6-compatible
```

Beispiele

Das Erstellen eines neuen Projektarchivs ist so einfach:

```
$ cd /var/svn
$ svnadmin create repos
$
```

In Subversion 1.0, a rd stets ein Berkeley-DB-Projektarchiv erstellt. In Subversion 1.1 ist ein Berkeley-DB-Projektarchiv der Standardtyp für Projektarchive, obwohl mit der Option `--fs-type` ein FSFS-Projektarchiv erstellt werden kann:

```
$ cd /var/svn
$ svnadmin create repos --fs-type fsfs
$
```

[1] Denken Sie daran, dass **svnadmin** nur mit lokalen *Pfaden* funktioniert, nicht mit *URLs*.

Name

svnadmin deltify — Geänderte Pfade im Revisionsbereich deltifizieren.

Aufruf

```
svnadmin deltify [-r LOWER[:UPPER]] REPOS_PATH
```

Beschreibung

In aktuellen Versionen von Subversion ist **svnadmin deltify** nur aus historischen Gründen vorhanden. Von diesem nicht mehr benötigten Befehl wird abgeraten.

Es stammt aus der Zeit, als Subversion Administratoren eine umfangreichere Kontrolle über Kompressionsstrategien im Projektarchiv einräumte. Es stellte sich heraus, dass es jede Menge Komplexität bei *sehr* wenig Gewinn bedeutete, deshalb wird hiervon abgeraten.

Optionen

```
--quiet (-q)
--revision (-r) REV
```

Name

svnadmin dump — Den Inhalt des Dateisystems nach `stdout` schreiben.

Aufruf

```
svnadmin dump REPOS_PATH [-r LOWER[:UPPER]] [--incremental] [--deltas]
```

Beschreibung

Schreibt den Inhalt des Dateisystems nach `stdout`, indem ein Format verwendet wird, dass portabel zu „Auszugsdateien“ ist; Rückmeldungen werden nach `stderr` geschrieben. Ausgegeben werden die Revisionen *`LOWER`* bis *`UPPER`*. Wenn keine Revisionen angegeben sind, werden alle Revisionsbäume ausgegeben. Wird nur *`LOWER`* angegeben, wird nur dieser Baum ausgegeben. Siehe „Projektarchiv-Daten woanders hin verschieben“ für einen praktischen Anwendungsfall.

Standardmäßig beinhaltet der Auszugsstrom von Subversion eine einzige Revision (die erste im angegebenen Revisionsbereich) in der jede Datei und jedes Verzeichnis dieser Revision im Projektarchiv präsentiert wird, als sei der gesamte Baum auf einmal hinzugefügt worden. Es folgen die anderen Revisionen (der Rest der Revisionen aus dem angegebenen Bereich), die nur die in diesen Revisionen geänderten Dateien und Verzeichnisse umfassen. Für eine geänderte Datei werden sowohl die vollständige Textrepräsentation ihres Inhalts als auch ihre Eigenschaften in den Auszug übernommen; für ein Verzeichnis werden alle seine Eigenschaften übernommen.

Zwei nützliche Optionen beeinflussen das Verhalten der Erstellung der Auszugsdatei. Die erste Option ist `--incremental`, die einfach bewirkt, dass bei der ersten Revision nur die in ihr geänderten Dateien und Verzeichnisse ausgegeben werden, statt es so aussehen zu lassen, als sei ein vollständiger Baum hinzugefügt worden; die Ausgabe erfolgt somit auf dieselbe Weise wie für die anderen Revisionen. Das ist nützlich, um eine relativ kleine Auszugsdatei zu erzeugen, die in ein anderes Projektarchiv geladen werden soll, welches bereits die Dateien und Verzeichnisse aus dem Original-Projektarchiv beinhaltet.

Die zweite nützliche Option ist `--deltas`. Diese Option veranlasst **svnadmin dump**, statt Volltextrepräsentationen von Dateiinhalten und Eigenschafts-Listen nur die jeweiligen Unterschiede zu früheren Versionen auszugeben. Das verringert (in einigen Fällen erheblich) den Umfang der Auszugsdatei, die **svnadmin dump** erzeugt. Allerdings gibt es bei der Option auch Nachteile – deltifizierte Auszugsdateien erfordern bei der Erstellung mehr Rechenkapazität, können nicht durch **svndumpfilter** behandelt werden und tendieren dazu, sich mit Werkzeugen von Drittanbietern, so wie **gzip** oder **bzip2**, nicht so gut komprimieren zu lassen wie die entsprechenden undeltifizierten Auszugsdateien.

Optionen

```
--deltas
--incremental
--quiet (-q)
--revision (-r) REV
```

Beispiele

Das gesamte Projektarchiv ausgeben:

```
$ svnadmin dump /var/svn/repos > full.dump

* Revision 0 ausgegeben.
* Revision 1 ausgegeben.
* Revision 2 ausgegeben.
…
```

Eine einzelne Transaktion Ihres Projektarchivs inkrementell ausgeben:

```
$ svnadmin dump /var/svn/repos -r 21 --incremental > incr.dump
* Revision 21 ausgegeben.
```

Name

svnadmin help (h, ?) — Hilfe!

Aufruf

```
svnadmin help [SUBCOMMAND...]
```

Beschreibung

Dieser Unterbefehl ist nützlich, falls Sie auf einer einsamen Insel gestrandet sind – ohne Netzverbindung oder dieses Buch.

Name

svnadmin hotcopy — Eine Kopie des Projektarchivs während des laufenden Betriebs erstellen.

Aufruf

```
svnadmin hotcopy REPOS_PATH NEW_REPOS_PATH
```

Beschreibung

Dieser Unterbefehl erstellt eine vollständige Sicherheitskopie Ihres Projektarchivs während des laufenden Betriebs; dabei werden alle Hooks, Konfigurationsdateien und Datenbankdateien berücksichtigt. Falls Sie die Option `--clean-logs` angeben, führt **svnadmin** die Sicherungskopie Ihres Projektarchivs durch und löscht anschließend nicht benötigte Berkeley-DB-Protokolldateien aus Ihrem Original-Projektarchiv. Sie können diesen Befehl jederzeit aufrufen und eine Sicherungskopie des Projektarchivs anlegen, egal, ob andere Prozesse gerade auf das Projektarchiv zugreifen.

Optionen

`--clean-logs`

Wie in „Berkeley DB“ erläutert, sind mit **svnadmin hotcopy** erstellte Berkeley-DB-Projektarchive *nicht* über Betriebssystemgrenzen portabel, sie funktionieren auch nicht auf Maschinen mit anderer „Byte-Reihenfolge“ als auf der Maschine, auf der sie erzeugt wurden.

Name

svnadmin list-dblogs — Anfrage an die Berkeley DB, welche Protokolldateien für ein gegebenes Subversion-Projektarchiv vorhanden sind (nur für Projektarchive, die auf BDB basieren).

Aufruf

```
svnadmin list-dblogs REPOS_PATH
```

Beschreibung

Berkeley DB erzeugt Protokolle für alle Änderungen am Projektarchiv, die ihm im Angesicht einer Katastrophe die Wiederherstellung erlauben. Falls Sie nicht DB_LOG_AUTOREMOVE aktivieren, sammeln sich die Protokolldateien an, obwohl die meisten nicht mehr benötigt werden und gelöscht werden könnten, um Plattenplatz zurückzuholen. Siehe „Plattenplatzverwaltung“ für weitere Informationen.

Name

svnadmin list-unused-dblogs — Anfrage an Berkeley DB, welche Protokolldateien zum Löschen freigegeben sind (nur für Projektarchive, die auf BDB basieren).

Aufruf

```
svnadmin list-unused-dblogs REPOS_PATH
```

Beschreibung

Berkeley DB erzeugt Protokolle für alle Änderungen am Projektarchiv, die ihm im Angesicht einer Katastrophe die Wiederherstellung erlauben. Falls Sie nicht DB_LOG_AUTOREMOVE aktivieren, sammeln sich die Protokolldateien an, obwohl die meisten nicht mehr benötigt werden und gelöscht werden könnten, um Plattenplatz zurückzuholen. Siehe „Plattenplatzverwaltung“ für weitere Informationen.

Beispiele

Alle nicht benötigten Protokolldateien aus dem Projektarchiv entfernen:

```
$ svnadmin list-unused-dblogs /var/svn/repos
/var/svn/repos/log.0000000031
/var/svn/repos/log.0000000032
/var/svn/repos/log.0000000033

$ svnadmin list-unused-dblogs /var/svn/repos | xargs rm

## Plattenplatz zurückgeholt!
```

Name

svnadmin load — Einen Projektarchiv-Auszugsstrom von `stdin` laden.

Aufruf

```
svnadmin load REPOS_PATH
```

Beschreibung

Liest einen Projektarchiv-Auszugsstrom von `stdin` und übergibt neue Revisionen an das Dateisystem des Projektarchivs. Hinweise zum Fortschritt werden nach `stdout` geschrieben.

Optionen

```
--force-uuid
--ignore-uuid
--parent-dir
--quiet (-q)
--use-post-commit-hook
--use-pre-commit-hook
```

Beispiele

Dies zeigt die Anfangsphase des Ladens eines Projektarchivs aus einer Sicherungsdatei (die natürlich mit **svnadmin dump** erstellt wurde):

```
$ svnadmin load /var/svn/restored < repos-backup

<<< Neue Transaktion basierend auf Originalrevision 1 gestartet
     * Füge Pfad hinzu: test ... erledigt.
     * Füge Pfad hinzu: test/a ... erledigt.
…
```

Oder falls Sie in ein Unterverzeichnis laden möchten:

```
$ svnadmin load --parent-dir new/subdir/for/project \
                /var/svn/restored < repos-backup

<<< Neue Transaktion basierend auf Originalrevision 1 gestartet
     * Füge Pfad hinzu: test ... erledigt.
     * Füge Pfad hinzu: test/a ... erledigt.
…
```

Name

svnadmin lslocks — Ausgeben der Beschreibungen zu allen Sperren.

Aufruf

```
svnadmin lslocks REPOS_PATH [PATH-IN-REPOS]
```

Beschreibung

Gibt Beschreibungen zu allen Sperren im Projektarchiv *`REPOS_PATH`* unterhalb des Pfades *`PATH-IN-REPOS`* aus. Falls *`PATH-IN-REPOS`* nicht angegeben ist, wird standardmäßig das Wurzelverzeichnis des Projektarchivs angenommen.

Optionen

Keine

Beispiele

Dies zeigt die einzige gesperrte Datei im Projektarchiv unter `/var/svn/repos` an:

```
$ svnadmin lslocks /var/svn/repos

Pfad: /tree.jpg
UUID Marke: opaquelocktoken:ab00ddf0-6afb-0310-9cd0-dda813329753
Eigentümer: harry
Erstellt: 2005-07-08 17:27:36 -0500 (Fri, 08 Jul 2005)
Läuft ab:
Kommentar (1 Zeilen):
Rework the uppermost branches on the bald cypress in the foreground.
```

Name

svnadmin lstxns — Die Namen aller unvollendeten Transaktionen ausgeben.

Aufruf

```
svnadmin lstxns REPOS_PATH
```

Beschreibung

Gibt die Namen aller unvollendeten Transaktionen aus. Siehe „Entfernen unvollendeter Transaktionen“ für Informationen darüber, wie unvollendete Transaktionen erzeugt werden und was Sie damit tun sollten.

Beispiele

Alle ausstehenden Transaktionen in einem Projektarchiv ausgeben:

```
$ svnadmin lstxns /var/svn/repos/
1w
1x
```

Name

svnadmin pack — Das Projektarchiv nach Möglichkeit in ein effizienteres Speichermodell komprimieren.

Aufruf

```
svnadmin pack REPOS_PATH
```

Beschreibung

Siehe „FSFS Filtersystem packen“ für weitere Informationen.

Optionen

Keine

Name

svnadmin recover — Stellt wieder einen konsistenten Zustand der Projektarchiv-Datenbank her (nur anwendbar für Projektarchiv, die auf `BDB` basieren). Falls `repos/conf/passwd` nicht vorhanden ist, wird darüber hinaus eine Standard-Passwort-Datei erstellt.

Aufruf

```
svnadmin recover REPOS_PATH
```

Beschreibung

Rufen Sie diesen Befehl auf, falls sie eine Fehlermeldung erhalten, die darauf hindeutet, dass das Projektarchiv wiederhergestellt werden muss.

Optionen

```
--wait
```

Beispiele

Wiederherstellung eines aufgehängten Projektarchivs:

```
$ svnadmin recover /var/svn/repos/

Exklusiven Zugriff auf das Projektarchiv erlangt
Bitte warten, die Wiederherstellung des Projektarchivs kann einige Zeit dauern ...

Wiederherstellung vollständig abgeschlossen.
Die neueste Revision des Projektarchivs ist 34.
```

Die Wiederherstellung der Datenbank erfordert eine exklusive Sperre auf dem Projektarchiv. (Das ist eine „Datenbank-Sperre“; siehe Anmerkung Die drei Bedeutungen von „Sperre“.) Falls ein anderer Prozess auf das Projektarchiv zugreift, gibt **svnadmin recover** einen Fehler aus:

```
$ svnadmin recover /var/svn/repos

svn: Konnte keinen exklusiven Zugriff auf das Projektarchiv erlangen
Vielleicht hat noch ein anderer Prozess (httpd, svnserve, svn)
das Projektarchiv geöffnet?

$
```

Die Option `--wait` veranlasst **svnadmin recover** auf unbestimmte Zeit auf das Abmelden anderer Prozesse zu warten:

```
$ svnadmin recover /var/svn/repos --wait

Warte auf Freigabe des Projektarchivs; Vielleicht ist es durch einen anderen Prozess
geöffnet?

### Zeit vergeht…
```

```
Exklusiven Zugriff auf das Projektarchiv erlangt
Bitte warten, die Wiederherstellung des Projektarchivs kann einige Zeit dauern ...

Wiederherstellung vollständig abgeschlossen.
Die neueste Revision des Projektarchivs ist 34.
```

Name

svnadmin rmlocks — Eine oder mehrere Sperren vom Projektarchiv bedingungslos entfernen.

Aufruf

```
svnadmin rmlocks REPOS_PATH LOCKED_PATH...
```

Beschreibung

Entfernt eine oder mehrere Sperren von jedem *`LOCKED_PATH`*.

Optionen

Keine

Beispiele

Das entfernt die Sperren auf `tree.jpg` und `house.jpg` im Projektarchiv unter `/var/svn/repos`:

```
$ svnadmin rmlocks /var/svn/repos tree.jpg house.jpg

Sperre für »/tree.jpg« entfernt.
Sperre für »/house.jpg« entfernt.
```

Name

svnadmin rmtxns — Transaktionen aus einem Projektarchiv löschen.

Aufruf

```
svnadmin rmtxns REPOS_PATH TXN_NAME...
```

Beschreibung

Löscht nicht abgeschlossene Transaktionen aus einem Projektarchiv. Dies wird detailliert in „Entfernen unvollendeter Transaktionen“ behandelt.

Optionen

```
--quiet (-q)
```

Beispiele

Benannte Transaktionen entfernen:

```
$ svnadmin rmtxns /var/svn/repos/ 1w 1x
```

Glücklicherweise lässt sich die Ausgabe von **svnadmin lstxns** hervorragend als Eingabe für **svnadmin rmtxns** verwenden:

```
$ svnadmin rmtxns /var/svn/repos/  `svnadmin lstxns /var/svn/repos/`
```

Das entfernt alle nicht abgeschlossenen Transaktionen aus Ihrem Projektarchiv.

Name

svnadmin setlog — Die Protokollnachricht einer Revision setzen.

Aufruf

```
svnadmin setlog REPOS_PATH -r REVISION FILE
```

Beschreibung

Setzt die Protokollnachricht von Revision *`REVISION`* auf den Inhalt von *`FILE`*.

Dies ist ähnlich der Verwendung von **svn propset** mit der Option `--revprop`, um die Eigenschaft `svn:log` einer Revision zu setzen, mit der Ausnahme, dass Sie auch die Option `--bypass-hooks` verwenden können, um die Ausführung irgendwelcher Vor- oder Nach-Übergabe-Hooks zu verhindern. Das kann nützlich sein, falls die Änderung von Revisions-Eigenschaften im Hook `pre-revprop-change` nicht ermöglicht wurde.

Revisions-Eigenschaften sind nicht versionskontrolliert, so dass vorherige Protokolldateien durch diesen Befehl dauerhaft überschrieben werden.

Optionen

```
--bypass-hooks
--revision (-r) REV
```

Beispiele

Setzt die Protokollnachricht für Revision 19 auf den Inhalt der Datei `msg`:

```
$ svnadmin setlog /var/svn/repos/ -r 19 msg
```

Name

svnadmin setrevprop — Setzt eine Eigenschaft für eine Revision.

Aufruf

```
svnadmin setrevprop REPOS_PATH -r REVISION NAME FILE
```

Beschreibung

Setzt die Eigenschaft *`NAME`* für die Revision *`REVISION`* auf den Inhalt von *`FILE`*. Verwenden Sie `--use-pre-revprop-change-hook` oder `--use-post-revprop-change-hook`, um die Hooks auszulösen, die mit Revisions-Eigenschaften in Verbindung stehen (falls Sie beispielsweise von Ihrem `post-revprop-change-hook` eine Benachrichtigung per E-Mail erhalten wollen).

Optionen

```
--revision (-r) ARG
--use-post-revprop-change-hook
--use-pre-revprop-change-hook
```

Beispiele

Im Folgenden wird die Revisions-Eigenschaft `repository-photo` auf den Inhalt der Datei `sandwich.png` gesetzt:

```
$ svnadmin setrevprop /var/svn/repos -r 0 repository-photo sandwich.png
```

Wie Sie sehen, gibt **svnadmin setrevprop** im Erfolgsfall nichts aus.

Name

svnadmin setuuid — Setzt die Projektarchiv-UUID zurück.

Aufruf

```
svnadmin setuuid REPOS_PATH [NEW_UUID]
```

Beschreibung

Setzt die Projektarchiv-UUID für das Projektarchiv bei *`REPOS_PATH`* zurück. Falls *`NEW_UUID`* angegeben ist, wird das als neue Projektarchiv-UUID verwendet; ansonsten wird eine nagelneue UUID für das Projektarchiv erzeugt.

Optionen

Keine

Beispiele

Falls Sie `/var/svn/repos` mit **svnsync** nach `/var/svn/repos-new` synchronisiert haben und `repos-new` als Projektarchiv verwenden möchten, sollten Sie die UUID für `repos-new` auf die UUID von `repos` setzen, damit Ihre Benutzer keine neue Arbeitskopie auschecken müssen, um die Änderung wirksam zu machen:

```
$ svnadmin setuuid /var/svn/repos-new 2109a8dd-854f-0410-ad31-d604008985ab
```

Wie Sie sehen, gibt **svnadmin setuuid** im Erfolgsfall nichts aus.

Name

svnadmin upgrade — Ein Projektarchiv auf die neueste unterstützte Schemaversion aktualisieren.

Aufruf

```
svnadmin upgrade REPOS_PATH
```

Beschreibung

Aktualisiert das Projektarchiv bei *REPOS_PATH* auf die neueste unterstützte Schemaversion.

Diese Funktionalität wird für Projektarchiv-Administratoren zur Verfügung gestellt, die neue Subversion-Funktionen verwenden wollen, ohne eine möglicherweise teure Auszug- und Ladeaktion durchführen zu müssen. Die Aktualisierung führt nur die dafür notwendigen Arbeiten aus und bewahrt dabei die Integrität des Projektarchivs. Während ein Auszug und das anschließende Laden den optimiertesten Projektarchiv-Zustand garantieren, kann **svnadmin upgrade** dies nicht gewährleisten.

Vor einer Aktualisierung sollten Sie *immer* eine Sicherung Ihres Projektarchivs vornehmen.

Optionen

Keine

Beispiele

Aktualisieren des Projektarchivs unter `/var/repos/svn`:

```
$ svnadmin upgrade /var/repos/svn

Exklusiven Zugriff auf das Projektarchiv erlangt.
Bitte warten, die Wiederherstellung des Projektarchivs kann einige Zeit dauern ..."

Aktualisierung abgeschlossen.
```

Name

svnadmin verify — Verifizieren der im Projektarchiv gespeicherten Daten.

Aufruf

```
svnadmin verify REPOS_PATH
```

Beschreibung

Rufen Sie diesen Befehl auf, wenn Sie die Integrität Ihres Projektarchivs verifizieren möchten. Im Wesentlichen werden dabei alle Revisionen im Projektarchiv durchlaufen, indem ein Auszug aller Revisionen vorgenommen wird, wobei die Ausgabe verworfen wird – es ist keine schlechte Idee, diesen Befehl regelmäßig laufen zu lassen, um Festplattenfehlern und „schlechten Bits“ vorzubeugen. Schlägt dieser Befehl fehl – was er beim ersten Anzeichen eines Problems machen wird – bedeutet das, dass Ihr Projektarchiv mindestens eine schadhafte Revision hat, und Sie sollten die schadhafte Revision aus einer Sicherung wiederherstellen (Sie haben doch eine Sicherung gemacht, oder?).

Optionen

```
--quiet (-q)
--revision (-r) ARG
```

Beispiele

Verifizieren eines aufgehängten Projektarchivs:

```
$ svnadmin verify /var/svn/repos/

* Revision 1729 verifiziert.
```

svnlook – Subversion Projektarchiv-Untersuchung

svnlook ist ein Kommandozeilenwerkzeug zur Untersuchung verschiedener Aspekte eines Subversion-Projektarchivs. Es nimmt keinerlei Änderungen am Projektarchiv vor – es wird nur zum Nachsehen benutzt. **svnlook** wird typischerweise von den Projektarchiv-Hooks verwendet, doch auch einem Projektarchiv-Administrator könnte es zu Diagnosezwecken dienlich sein.

Da **svnlook** über direkten Projektarchiv-Zugriff arbeitet (und deshalb nur auf der Maschine verwendet werden kann, auf der das Projektarchiv liegt), greift es auf das Projektarchiv über einen Pfad statt über einen URL zu.

Falls keine Revision oder Transaktion angegeben ist, bezieht sich **svnlook** standardmäßig auf die neueste (letzte) Revision des Projektarchivs.

svnlook Optionen

Optionen für **svnlook** sind global, genauso wie bei **svn** und **svnadmin**; jedoch treffen die meisten Optionen auf nur einen Unterbefehl zu, da der Umfang der Funktionalität von **svnlook** (absichtlich) eingeschränkt ist:

```
--copy-info
```
Veranlasst **svnlook changed** detaillierte Informationen zur Herkunft der Kopie anzuzeigen.

```
--diff-copy-from
```

Unterschiede von kopierten Objekten zur Quelle anzeigen.

`--extensions` (`-x`) *`ARG`*
Dient der Einstellung von Anpassungen, die Subversion bei der Berechnung von Unterschieden berücksichtigen soll. Gültige Erweiterungen umfassen:

`--ignore-space-change` (`-b`)
Änderungen der Anzahl von Leerraumzeichen ignorieren.

`--ignore-all-space` (`-w`)
Leerraum vollständig ignorieren.

`--ignore-eol-style`
Änderungen der Zeilenende-Markierungen ignorieren.

`--unified` (`-u`)
Drei Zeilen im Unified-Diff-Format anzeigen.

Der Standardwert ist `-u`.

Beachten Sie, dass der Wert der Option `--extension` (`-x`) nicht auf die oben erwähnten Optionen beschränkt ist, sondern *beliebige* zusätzliche Argumente umfassen kann, die an ein externen Diff-Befehl weitergereicht werden, falls Subversion hierfür konfiguriert wurde. Wenn Sie mehrere Argumente übergeben möchten, müssen alle innerhalb von Anführungszeichen stehen.

`--full-paths`
Veranlasst **svnlook tree**, vollständige Pfade anstatt hierarchischer, eingerückter Pfadkomponenten anzuzeigen.

`--limit` (`-l`) *`ARG`*
Ausgabe auf maximal *`ARG`* Objekte beschränken.

`--no-diff-deleted`
Verhindert, dass **svnlook diff** Unterschiede für gelöschte Dateien ausgibt. Das Standardverhalten bei gelöschten Dateien ist es, dieselben Unterschiede auszugeben, die Sie bekommen hätten, wenn Sie die Datei beibehalten, jedoch den Inhalt gelöscht hätten.

`--no-diff-added`
Verhindert, dass **svnlook diff** Unterschiede für hinzugefügte Dateien ausgibt. Das Standardverhalten für hinzugefügte Dateien ist es, dieselben Unterschiede auszugeben, die Sie erhalten hätten, wenn Sie den gesamten Inhalt einer bestehenden (leeren) Datei hinzugefügt hätten.

`--non-recursive` (`-N`)
Nur auf ein einzelnes Verzeichnis anwenden.

`--revision` (`-r`)
Gibt eine bestimmte Revisionsnummer an, die Sie untersuchen möchten.

`--revprop`
Arbeitet auf einer Revisions-Eigenschaft statt auf einer datei- oder verzeichnisspezifischen Eigenschaft. Diese Option verlangt eine Revision, die mit der Option `--revision` (`-r`) angegeben wird.

`--transaction` (`-t`)
Gibt eine bestimmte Transaktions-ID an, die Sie untersuchen möchten.

`--show-ids`
Gibt für jeden Pfad im Dateisystembaum die Revisions-IDs der Dateisystemknoten an.

`--verbose` (`-v`)
Ausführliche Ausgabe. In Verbindung mit **svnlook proplist** veranlasst es Subversion beispielsweise, nicht nur die Liste der Eigenschaften anzuzeigen, sondern auch deren Werte.

`--xml`

svnlook Unterbefehle

Hier sind die zahlreichen Unterbefehle für das Programm **svnlook**.

Name

svnlook author — Gibt den Autor aus.

Aufruf

```
svnlook author REPOS_PATH
```

Beschreibung

Gibt den Autor für eine Revision oder Transaktion im Projektarchiv aus.

Optionen

```
--revision (-r) REV
--transaction (-t) TXN
```

Beispiele

svnlook author ist praktisch, jedoch nicht sehr aufregend.

```
$ svnlook author -r 40 /var/svn/repos
sally
```

Name

svnlook cat — Gibt den Inhalt einer Datei aus.

Aufruf

```
svnlook cat REPOS_PATH PATH_IN_REPOS
```

Beschreibung

Gibt den Inhalt einer Datei aus.

Optionen

```
--revision (-r) REV
--transaction (-t) TXN
```

Beispiele

Dies gibt den Inhalt einer Datei bei `/trunk/README` in Transaktion `ax8` aus:

```
$ svnlook cat -t ax8 /var/svn/repos /trunk/README

               Subversion, a version control system.
               =====================================

$LastChangedDate: 2003-07-17 10:45:25 -0500 (Thu, 17 Jul 2003) $

Contents:

     I. A FEW POINTERS
    II. DOCUMENTATION
   III. PARTICIPATING IN THE SUBVERSION COMMUNITY
…
```

Name

svnlook changed — Gibt die geänderten Pfade aus.

Aufruf

```
svnlook changed REPOS_PATH
```

Beschreibung

Sowohl die sich in einer bestimmten Revision oder Transaktion geänderten Pfade als auch die Zustandsbuchstaben im Stil von **svn update** in den ersten beiden Spalten werden ausgegeben:

`'A '`
: Objekt dem Projektarchiv hinzugefügt

`'D '`
: Objekt aus dem Projektarchiv gelöscht

`'U '`
: Dateiinhalt geändert

`'_U'`
: Eigenschaften eines Objektes geändert; beachten Sie den führenden Unterstrich

`'UU'`
: Dateiinhalt und Eigenschaften geändert

Dateien und Verzeichnisse können unterschieden werden, da Verzeichnispfade mit dem abschließenden Zeichen „/“ versehen sind.

Optionen

```
--copy-info
--revision (-r) REV
--transaction (-t) TXN
```

Beispiele

Dies gibt eine Liste aller geänderten Dateien und Verzeichnisse in Revision 39 eines Test-Projektarchivs aus. Beachten Sie, dass das erste geänderte Objekt ein Verzeichnis ist, was durch das abschließende `/` offensichtlich ist:

```
$ svnlook changed -r 39 /var/svn/repos
A   trunk/vendors/deli/
A   trunk/vendors/deli/chips.txt
A   trunk/vendors/deli/sandwich.txt
A   trunk/vendors/deli/pickle.txt
U   trunk/vendors/baker/bagel.txt
_U  trunk/vendors/baker/croissant.txt
UU  trunk/vendors/baker/pretzel.txt
D   trunk/vendors/baker/baguette.txt
```

Hier ist ein Beispiel, das eine Revision zeigt, in der eine Datei umbenannt wurde:

```
$ svnlook changed -r 64 /var/svn/repos
A   trunk/vendors/baker/toast.txt
D   trunk/vendors/baker/bread.txt
```

Unglücklicherweise wird in der vorangegangenen Ausgabe nicht deutlich, dass eine Beziehung zwischen der gelöschten und hinzugefügten Datei besteht. Verwenden Sie die Option `--copy-info`, um diese Beziehung zu verdeutlichen:

```
$ svnlook changed -r 64 --copy-info /var/svn/repos

A + trunk/vendors/baker/toast.txt
    (von trunk/vendors/baker/bread.txt:r63)
D   trunk/vendors/baker/bread.txt
```

Name

svnlook date — Ausgabe des Zeitstempels.

Aufruf

```
svnlook date REPOS_PATH
```

Beschreibung

Gibt den Zeitstempel einer Revision oder Transaktion im Projektarchiv aus.

Optionen

```
--revision (-r) REV
--transaction (-t) TXN
```

Beispiele

Dies zeigt das Datum von Revision 40 eines Test-Projektarchivs:

```
$ svnlook date -r 40 /var/svn/repos/
2003-02-22 17:44:49 -0600 (Sat, 22 Feb 2003)
```

Name

svnlook diff — Ausgabe der Unterschiede geänderter Dateien und Eigenschaften.

Aufruf

```
svnlook diff REPOS_PATH
```

Beschreibung

Ausgabe der Unterschiede geänderter Dateien und Eigenschaften in einem Projektarchiv im Stil von GNU.

Optionen

```
--diff-copy-from
--no-diff-added
--no-diff-deleted
--revision (-r) REV
--transaction (-t) TXN
--extensions (-x) ARG
```

Beispiele

Dies zeigt eine frisch hinzugefügte (leere) und eine kopierte Datei sowie je zwei gelöschte und geänderte Dateien:

```
$ svnlook diff -r 40 /var/svn/repos/

Kopiert: egg.txt (von Rev 39, trunk/vendors/deli/pickle.txt)

Hinzugefügt: trunk/vendors/deli/soda.txt
===================================================================

Geändert: trunk/vendors/deli/sandwich.txt
===================================================================
--- trunk/vendors/deli/sandwich.txt (original)
+++ trunk/vendors/deli/sandwich.txt 2003-02-22 17:45:04.000000000 -0600
@@ -0,0 +1 @@
+Don't forget the mayo!

Geändert: trunk/vendors/deli/logo.jpg
===================================================================
(Binärdateien sind unterschiedlich)

Gelöscht: trunk/vendors/deli/chips.txt
===================================================================

Gelöscht: trunk/vendors/deli/pickle.txt
===================================================================
```

Falls eine Datei eine Eigenschaft `svn:mime-type` besitzt das nicht textuell ist, werden die Unterschiede nicht explizit aufgezeigt.

Name

svnlook dirs-changed — Ausgabe der Verzeichnisse, die für sich geändert wurden.

Aufruf

```
svnlook dirs-changed REPOS_PATH
```

Beschreibung

Gibt die Verzeichnisse aus, die für sich (Änderungen an Eigenschaften) oder deren Kinder geändert wurden.

Optionen

```
--revision (-r) REV
--transaction (-t) TXN
```

Beispiele

Dies gibt die Verzeichnisse aus, die in Revision 40 unseres Beispiel-Projektarchivs geändert wurden:

```
$ svnlook dirs-changed -r 40 /var/svn/repos
trunk/vendors/deli/
```

Name

svnlook help (h, ?) — Hilfe!

Aufruf

```
Auch svnlook -h und svnlook -?.
```

Beschreibung

Zeigt die Hilfemeldung für **svnlook** an. Auch dieser Befehl ist, wie sein Bruder **svn help**, Ihr Freund, auch falls Sie ihn nicht mehr aufrufen und es vergessen haben, ihn zu Ihrer letzten Fete einzuladen.

Optionen

Keine

Name

svnlook history — Gibt die Geschichte eines Projektarchiv-Pfades aus (oder des Wurzelverzeichnisses, falls kein Pfad angegeben ist).

Aufruf

```
svnlook history REPOS_PATH [PATH_IN_REPOS]
```

Beschreibung

Gibt die Geschichte eines Projektarchiv-Pfades aus (oder des Wurzelverzeichnisses, falls kein Pfad angegeben ist).

Optionen

```
--limit (-l) NUM
--revision (-r) REV
--show-ids
```

Beispiele

Dies gibt die Geschichte für den Pfad `/branches/bookstore` ab Revision 13 unseres Beispiel-Projektarchivs aus:

```
$ svnlook history -r 13 /var/svn/repos /branches/bookstore --show-ids

REVISION   PFAD <ID>
--------   ---------
      13   /branches/bookstore <1.1.r13/390>
      12   /branches/bookstore <1.1.r12/413>
      11   /branches/bookstore <1.1.r11/0>
       9   /trunk <1.0.r9/551>
       8   /trunk <1.0.r8/131357096>
       7   /trunk <1.0.r7/294>
       6   /trunk <1.0.r6/353>
       5   /trunk <1.0.r5/349>
       4   /trunk <1.0.r4/332>
       3   /trunk <1.0.r3/335>
       2   /trunk <1.0.r2/295>
       1   /trunk <1.0.r1/532>
```

Name

svnlook info — Ausgabe des Autors, des Zeitstempels, der Größe der Protokollnachricht und der Protokollnachricht.

Aufruf

```
svnlook info REPOS_PATH
```

Beschreibung

Gibt den Autor, den Zeitstempel, die Größe der Protokollnachricht (in Bytes) und die Protokollnachricht aus, gefolgt von einem Zeilenvorschubzeichen.

Optionen

```
--revision (-r) REV
--transaction (-t) TXN
```

Beispiele

Dies zeigt die Informationen für Revision 40 unseres Beispiel-Projektarchivs:

```
$ svnlook info -r 40 /var/svn/repos
sally
2003-02-22 17:44:49 -0600 (Sat, 22 Feb 2003)
16
Rearrange lunch.
```

Name

svnlook lock — Falls eine Sperre für einen Pfad im Projektarchiv besteht, wird sie beschrieben.

Aufruf

```
svnlook lock REPOS_PATH PATH_IN_REPOS
```

Beschreibung

Zeigt alle verfügbaren Informationen zur Sperre bei *`PATH_IN_REPOS`* an. Falls *`PATH_IN_REPOS`* nicht gesperrt ist, wird nichts ausgegeben.

Optionen

Keine

Beispiele

Dies beschreibt die Sperre auf Datei `tree.jpg`:

```
$ svnlook lock /var/svn/repos tree.jpg

UUID Marke: opaquelocktoken:ab00ddf0-6afb-0310-9cd0-dda813329753
Eigentümer: harry
Erstellt: 2005-07-08 17:27:36 -0500 (Fri, 08 Jul 2005)
Läut ab:
Kommentar (1 Zeilen):
Rework the uppermost branches on the bald cypress in the foreground.
```

Name

svnlook log — Ausgabe der Protokollnachricht mit einem Zeilenvorschub.

Aufruf

```
svnlook log REPOS_PATH
```

Beschreibung

Gibt die Protokollnachricht aus.

Optionen

```
--revision (-r) REV
--transaction (-t) TXN
```

Beispiele

Dies zeigt die Protokollnachricht von Revision 40 unseres Beispiel-Projektarchivs:

```
$ svnlook log /var/svn/repos/
Rearrange lunch.
```

Name

svnlook propget (pget, pg) — Ausgabe des unveränderten Wertes einer Eigenschaft auf einem Pfad im Projektarchiv.

Aufruf

```
svnlook propget REPOS_PATH PROPNAME [PATH_IN_REPOS]
```

Beschreibung

Zeigt den Wert einer Eigenschaft auf einem Pfad im Projektarchiv an.

Optionen

```
--revision (-r) REV
--revprop
--transaction (-t) TXN
```

Beispiele

Dies zeigt den Wert der Eigenschaft „seasonings“ auf der Datei `/trunk/sandwich` der Revision `HEAD` an:

```
$ svnlook pg /var/svn/repos seasonings /trunk/sandwich
mustard
```

Name

svnlook proplist (plist, pl) — Ausgabe der Namen und Werte von Eigenschaften versionierter Dateien und Verzeichnisse.

Aufruf

```
svnlook proplist REPOS_PATH [PATH_IN_REPOS]
```

Beschreibung

Gibt die Eigenschaften eines Pfades im Projektarchiv aus. Mit `--verbose` (`-v`) werden auch die Werte angezeigt.

Optionen

```
--revision (-r) REV
--revprop
--transaction (-t) TXN
--verbose (-v)
--xml
```

Beispiele

Dies zeigt die Namen der Eigenschaften, die auf die Datei `/trunk/README` in der Revision `HEAD` gesetzt sind:

```
$ svnlook proplist /var/svn/repos /trunk/README
  original-author
  svn:mime-type
```

Dies ist der gleiche Befehl wie im vorhergehenden Beispiel, doch diesmal mit der Ausgabe der Werte:

```
$ svnlook -v proplist /var/svn/repos /trunk/README
  original-author : harry
  svn:mime-type : text/plain
```

Name

svnlook tree — Ausgabe des Baums.

Aufruf

```
svnlook tree REPOS_PATH [PATH_IN_REPOS]
```

Beschreibung

Gibt den Baum beginnend bei *`PATH_IN_REPOS`* aus (falls angegeben; sonst bei der Wurzel des Baums); optional werden Revisions-IDs der Knoten angezeigt.

Optionen

```
--full-paths
--non-recursive (-N)
--revision (-r) REV
--show-ids
--transaction (-t) TXN
```

Example

Dies gibt den Baum für Revision 13 unseres Beispiel-Projektarchivs aus (mit Knoten-IDs):

```
$ svnlook tree -r 13 /var/svn/repos --show-ids
/ <0.0.r13/811>
 trunk/ <1.0.r9/551>
  button.c <2.0.r9/238>
  Makefile <3.0.r7/41>
  integer.c <4.0.r6/98>
 branches/ <5.0.r13/593>
  bookstore/ <1.1.r13/390>
   button.c <2.1.r12/85>
   Makefile <3.0.r7/41>
   integer.c <4.1.r13/109>
```

Name

svnlook uuid — Ausgabe der `UUID` des Projektarchivs.

Aufruf

```
svnlook uuid REPOS_PATH
```

Beschreibung

Gibt die `UUID` des Projektarchivs aus. `UUID` bedeutet *u*niversal *u*nique *id*entifier (universelle, eindeutige ID). Der Client von Subversion verwendet diese Identifizierung um zwischen Projektarchiven zu unterscheiden.

Optionen

Keine

Beispiele

```
$ svnlook uuid /var/svn/repos
e7fe1b91-8cd5-0310-98dd-2f12e793c5e8
```

Name

svnlook youngest — Ausgabe der letzten Revisionsnummer.

Aufruf

```
svnlook youngest REPOS_PATH
```

Beschreibung

Gibt die letzte Revisionsnummer eines Projektarchivs aus.

Optionen

Keine

Beispiele

Dies gibt die letzte Revision unseres Beispiel-Projektarchivs aus:

```
$ svnlook youngest /var/svn/repos/
42
```

svnsync – Subversion Projektarchiv-Spiegelung

svnsync ist das Werkzeug von Subversion zum entfernten Spiegeln von Projektarchiven. Einfach gesagt, gestattet es Ihnen, die Revisionen eines Projektarchivs in ein anderes zu überspielen.

In allen Spiegelszenarios gibt es zwei Projektarchive: das Quell-Projektarchiv und das Spiegel- (oder „Senken-“) Projektarchiv. Das Quell-Projektarchiv ist das Projektarchiv, aus dem **svnsync** Revisionen herauszieht. Das Spiegel-Projektarchiv ist das Ziel für diese Revisionen. Jedes dieser Projektarchive kann lokal oder entfernt sein – sie werden immer nur durch ihre URLs adressiert.

Der Prozess **svnsync** benötigt lediglich Lesezugriff auf das Quell-Projektarchiv; er wird nie versuchen, es zu verändern. Offensichtlich benötigt **svnsync** jedoch sowohl Lese- als auch Schreibzugriff auf das Spiegel-Projektarchiv.

svnsync ist sehr empfindlich gegenüber Änderungen im Spiegel-Projektarchiv, die nicht im Zuge einer Spiegelung vorgenommen wurden. Um das zu vermeiden, sollte der Prozess **svnsync** der einzige Prozess sein, der das Spiegel-Projektarchiv verändern darf.

svnsync Optionen

Optionen für **svnlook** sind global, genauso wie bei **svn** und **svnadmin**:

`--config-dir` *`DIR`*
: Weist Subversion an, Konfigurationsinformationen aus dem angegebenen Verzeichnis zu lesen, statt aus dem Standardverzeichnis (`.subversion` im Heimatverzeichnis des Benutzers).

`--no-auth-cache`
: Verhindert die Zwischenspeicherung von Authentisierungsinformationen (z.B. Anwendername und Passwort) in den Laufzeitkonfigurationsverzeichnissen von Subversion.

`--non-interactive`
: Im Fall einer fehlgeschlagenen Zugangsüberprüfung oder mangelnder Berechtigungen, verhindert diese Option die

Nachfrage nach Zugangsdaten (z.B. Anwendername oder Passwort). Dies ist nützlich, falls Sie Subversion innerhalb eines automatisierten Skriptes aufrufen und somit ein Abbruch mit Fehlermeldung angebrachter ist als eine Nachfrage.

`--quiet` (`-q`)
Fordert den Client auf, nur die wichtigsten Informationen beim Ausführen einer Operation auszugeben.

`--source-password` *`PASSWD`*
Gibt das Passwort für den Subversion-Server an, von dem Sie synchronisieren. Falls es nicht mitgegeben wird oder falsch ist, fragt Subversion bei Bedarf nach.

`--source-username` *`NAME`*
Gibt den Anwendernamen für den Subversion-Server an, von dem Sie synchronisieren. Falls es nicht mitgegeben wird oder falsch ist, fragt Subversion bei Bedarf nach.

`--sync-password` *`PASSWD`*
Gibt das Passwort für den Subversion-Server an, zu dem Sie synchronisieren. Falls es nicht mitgegeben wird oder falsch ist, fragt Subversion bei Bedarf nach.

`--sync-username` *`NAME`*
Gibt den Anwendernamen für den Subversion-Server an, zu dem Sie synchronisieren. Falls es nicht mitgegeben wird oder falsch ist, fragt Subversion bei Bedarf nach.

`--trust-server-cert`
In Verbindung mit `--non-interactive` zum Akzeptieren eines unbekannten SSL-Server-Zertifikates ohne Nachfrage.

svnsync-Unterbefehle

Hier sind die verschiedenen Unterbefehle für das Programm **svnsync**.

Name

svnsync copy-revprops — Alle Revisions-Eigenschaften einer bestimmten Revision (oder eines Revisionsbereiches) vom Quell-Projektarchiv ins Spiegel-Projektarchiv kopieren.

Aufruf

```
svnsync copy-revprops DEST_URL [REV[:REV2]]
```

Beschreibung

Da Revisions-Eigenschaften von Subversion jederzeit geändert werden können, ist es möglich, das die Eigenschaften einer Revision geändert werden, nachdem diese Revision bereits mit einem anderen Projektarchiv synchronisiert wurde. Da der Befehl **svnsync synchronize** nur auf einem Bereich von Revisionen arbeitet, die noch nicht synchronisiert worden sind, würde er eine Änderung einer Revisions-Eigenschaft außerhalb dieses Bereiches nicht erkennen. Dies würde zu einer Abweichung der Werte dieser Revisions-Eigenschaft zwischen dem Quell-Projektarchiv und dem Spiegel-Projektarchiv führen. **svnsync copy-revprops** ist die Lösung für dieses Problem. Verwenden Sie es, um die Revisions-Eigenschaften einer bestimmten Revision oder eines Revisionsbereiches erneut zu synchronisieren.

Optionen

```
--config-dir DIR
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
--trust-server-cert
```

Beispiele

Revisions-Eigenschaften einer einzelnen Revision erneut synchronisieren:

```
$ svnsync copy-revprops file:///var/svn/repos-mirror 6

Eigenschaften für Revision 6 kopiert.
$
```

Name

svnsync help — Hilfe!

Aufruf

```
svnsync help
```

Beschreibung

Dieser Unterbefehl ist nützlich, falls Sie ohne Netzverbindung oder einem Exemplar dieses Buches in einem ausländischen Gefängnis festsitzen, jedoch über eine lokale WLAN-Verbindung verfügen und eine Kopie Ihres Projektarchivs zum Sicherungsserver synchronisieren möchten, den Ira das Messer in Zellenblock D betreibt.

Optionen

Keine

Name

svnsync info — Informationen über die Synchronisierung eines Ziel-Projektarchivs ausgeben.

Aufruf

```
svnsync info DEST_URL
```

Beschreibung

Ausgabe der Synchronisierungs-Quell-URL, der UUID des Quell-Projektarchivs sowie der letzten Revision die aus der Quelle mit dem Ziel-Projektarchiv bei *`DEST_URL`* zusammengeführt wurde.

Optionen

```
--config-dir DIR
--no-auth-cache
--non-interactive
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
--trust-server-cert
```

Beispiele

Ausgabe der Synchronisierungs-Informationen eines gespiegelten Projektarchivs:

```
$ svnsync info file:///var/svn/repos-mirror

Quell-URL: http://svn.example.com/repos
UUID des Quellprojektarchivs: e7fe1b91-8cd5-0310-98dd-2f12e793c5e8
Letzte zusammengeführte Revision: 47
$
```

Name

svnsync initialize (init) — Ein Spiegel-Projektarchiv für die Synchronisierung aus dem Quell-Projektarchiv initialisieren.

Aufruf

```
svnsync initialize MIRROR_URL SOURCE_URL
```

Beschreibung

svnsync initialize überprüft, ob ein Projektarchiv die Voraussetzungen eines neuen Spiegel-Projektarchivs erfüllt – dass es keine bereits bestehende Versionsgeschichte hat und dass es Änderungen an Revisions-Eigenschaften zulässt – und legt die anfängliche Verwaltungsinformation an, welche das Spiegel-Projektarchiv mit dem Quell-Projektarchiv (durch *SOURCE_URL* angegeben) verknüpft. Dies ist der erste **svnsync**-Befehl, den Sie auf einem künftigen Spiegel-Projektarchiv laufen lassen.

Normalerweise ist *SOURCE_URL* der URL des Wurzelverzeichnisses des Subversion Projektarchivs, das Sie spiegeln möchten. Subversion 1.5 und neuer erlauben jedoch auch, **svnsync** für teilweise Projektarchiv-Spiegelungen zu verwenden; geben Sie einfach mit *SOURCE_URL* den URL des Unterverzeichnisses des Quell-Projektarchivs an, das Sie spiegeln möchten.

Optionen

```
--config-dir DIR
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
--trust-server-cert
```

Beispiele

Fehlgeschlagene Initialisierung eines Spiegel-Projektarchivs, da sich Revisions-Eigenschaften nicht ändern lassen:

```
$ svnsync initialize file:///var/svn/repos-mirror http://svn.example.com/repos

Das Projektarchiv lässt keine Änderungen an Revisionseigenschaften zu.
Bitten Sie den Administrator darum, eine Aktion »pre-revprop-change«
einzurichten."
$
```

Initialisierung eines Projektarchivs als Spiegel, wobei bereits ein `pre-revprop-change`-Hook bereitgestellt wurde, der alle Änderungen an Revisions-Eigenschaften zulässt:

```
$ svnsync initialize file:///var/svn/repos-mirror http://svn.example.com/repos

Eigenschaften für Revision 0 kopiert.
$
```

Name

svnsync synchronize (sync) — Alle ausstehenden Revisionen vom Quell-Projektarchiv zum Spiegel-Projektarchiv übertragen.

Aufruf

```
svnsync synchronize DEST_URL
```

Beschreibung

Der Befehl **svnsync synchronize** übernimmt die gesamte Schwerarbeit einer Projektarchiv-Spiegelung. Nachdem die Informationen über bereits kopierte Revisionen aus dem Spiegel-Projektarchiv geholt worden sind, wird damit begonnen, alle noch nicht gespiegelten Revisionen aus dem Quell-Projektarchiv zu kopieren.

svnsync synchronize kann sauber abgebrochen und erneut gestartet werden.

Optionen

```
--config-dir DIR
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
--trust-server-cert
```

Beispiele

Unsynchronisierte Revisionen aus dem Quell- ins Spiegel-Projektarchiv kopieren.

```
$ svnsync synchronize file:///var/svn/repos-mirror

Revision 1 übertragen.
Eigenschaften für Revision 1 kopiert.
Revision 2 übertragen.
Eigenschaften für Revision 2 kopiert.
Revision 3 übertragen.
Eigenschaften für Revision 3 kopiert.
…
Revision 45 übertragen.
Eigenschaften für Revision 45 kopiert.
Revision 46 übertragen.
Eigenschaften für Revision 46 kopiert.
Revision 47 übertragen.
Eigenschaften für Revision 47 kopiert.
$
```

svnserve – Maßgeschneideter Subversion-Server

svnserve gestattet den Zugriff auf Subversion-Projektarchive über das spezielle Netzprotokoll von Subversion.

Sie können **svnserve** als selbständigen Server-Prozess laufen lassen (für Clients, die das Zugriffsverfahren `svn://` verwenden); Sie können ihn bei Bedarf von einem Dämon wie **inetd** oder **xinetd** starten lassen (ebenfalls für `svn://`), oder Sie können ihn bei Bedarf durch **sshd** für das Zugriffsverfahren `svn+ssh://` starten lassen.

Egal, welches Zugriffsverfahren verwendet wird: sobald der Client ein Projektarchiv ausgewählt hat, indem er dessen URL überträgt, liest **svnserve** eine Datei namens `conf/svnserve.conf` im Projektarchiv-Verzeichnis, um spezifische Einstellungen wie etwa die zu verwendende Authentifizierungs-Datenbank und die anzuwendenden Autorisierungsrichtlinien zu ermitteln. Siehe „svnserve, ein maßgefertigter Server“ zu Details der Datei `svnserve.conf`.

svnserve-Optionen

Im Gegensatz zu den bisher beschriebenen Kommandos besitzt **svnserve** keine Unterbefehle – es wird ausschließlich über Optionen gesteuert.

`--daemon` (`-d`)
: Veranlasst **svnserve**, im Dämon-Modus zu laufen. **svnserve** verschwindet in den Hintergrund und akzeptiert und bedient TCP/IP-Verbindungen über den `svn`-Port (standardmäßig 3690).

`--foreground`
: In Verbindung mit `-d` wird **svnserve** veranlasst, im Vordergrund zu bleiben. Dies ist hauptsächlich nützlich für die Fehlersuche.

`--inetd` (`-i`)
: Veranlasst **svnserve**, die Dateideskriptoren `stdin` und `stdout` zu verwenden, was passend ist, falls der Dämon von **inetd** gestartet wird.

`--help` (`-h`)
: Gibt eine Zusammenfassung der Aufrufsyntax und beendet sich anschließend.

`--listen-host` *`HOST`*
: Veranlasst **svnserve**, an der durch *`HOST`* spezifizierten Schnittstelle auf Verbindungen zu warten; das kann entweder ein Rechnername oder eine IP-Adresse sein.

`--listen-once` (`-X`)
: Veranlasst **svnserve**, eine Verbindung über den `svn`-Port anzunehmen, sie zu bedienen und sich dann zu beenden. Dies ist hauptsächlich nützlich für die Fehlersuche.

`--listen-port` *`PORT`*
: Veranlasst **svnserve**, Verbindungen über *`PORT`* anzunehmen, falls es im Dämon-Modus läuft. (Dämonen unter FreeBSD horchen standardmäßig nur auf tcp6 – dies teilt ihnen mit, dass sie auch auf tcp4 horchen sollen.)

`--log-file` *`FILENAME`*
: Veranlasst **svnserve** die Datei *`FILENAME`* (falls nötig) anzulegen und für die operative Protokollausgaben von Subversion nach Art von **mod_dav_svn** zu verwenden. Für Einzelheiten, siehe „Protokollierung auf hohem Niveau“.

`--pid-file` *`FILENAME`*
: Veranlasst **svnserve**, seine Prozess-ID in die Datei *`FILENAME`* zu schreiben, die schreibbar sein muss für die Benutzerkennung unter der **svnserve** läuft.

`--root` (`-r`) *`ROOT`*
: Stellt das virtuelle Wurzelverzeichnis für Projektarchive ein, die von **svnserve** bedient werden. Der Pfadname in von Clients übergebenen URLs wird relativ zu dieser Wurzel interpretiert und kann dieses Wurzelverzeichnis auch nicht verlassen.

`--threads` (`-T`)
: Veranlasst **svnserve** wenn es im Dämon-Modus läuft, für jede Verbindung einen Thread statt einen Prozess zu starten (z.B., falls es unter Windows läuft). Der Prozess **svnserve** geht beim Start dennoch in den Hintergrund.

`--tunnel` (`-t`)
: Veranlasst **svnserve**, im Tunnelmodus zu laufen, der wie der **inetd**-Modus funktioniert (beide Modi bedienen eine Verbindung über `stdin`/`stdout` und beenden sich dann), mit der Ausnahme, dass angenommen wird, dass die Verbindung im Voraus mit dem Anwendernamen der aktuellen UID authentifiziert worden ist. Diese Option wird für Sie automatisch vom Client übergeben, wenn ein Tunnel-Agent wie **ssh** verwendet wird. Das bedeutet, dass für *Sie* kaum die Notwendigkeit besteht, jemals diese Option an **svnserve** zu übergeben. Falls Sie sich einmal dabei erwischen, auf der Kommandozeile `svnserve --tunnel` zu tippen und sich fragen, was als nächstes zu tun ist, siehe „Tunneln über SSH“.

`--tunnel-user NAME`
In Verbindung mit der Option `--tunnel` teilt diese Option **svnserve** mit, dass es davon ausgehen soll, dass *`NAME`* der authentifizierte Benutzer ist statt der UID des **svnserve**-Prozesses. Dies ist nützlich für Benutzer, die zwar ein einzelnes Systemkonto über SSH verwenden möchten, dabei aber unterschiedliche Identitäten für die Übergabe beibehalten möchten.

`--version`
Gibt Versionsinformationen und eine Auflistung aller vom Client unterstützten Module für den Zugriff auf ein Subversion-Projektarchiv aus und beendet sich.

svndumpfilter—Subversion History Filtering

svndumpfilter ist ein Kommandozeilenwerkzeug zum Entfernen von Geschichtsdaten aus einer Subversion-Auszugsdatei, indem Pfade mit einem oder mehreren Präfixen entweder ein- oder ausgeschlossen werden. Für Details siehe „svndumpfilter“.

svndumpfilter-Optionen

Optionen für **svndumpfilter** sind global, genauso wie bei **svn** und **svnadmin**:

`--drop-empty-revs`
Falls das Filtern dazu führen sollte, dass irgendwelche Revisionen leer sein sollten (d.h., zu keinerlei Änderungen im Projektarchiv führen), werden diese Revisionen aus der endgültigen Auszugsdatei entfernt.

`--renumber-revs`
Nach dem Filtern verbliebene Revisionen neu nummerieren.

`--skip-missing-merge-sources`
Überspringt Quellen von Zusammenführungen, die im Zuge der Filterung entfernt wurden. Ohne diese Option wird sich **svndumpfilter** mit einer Fehlermeldung beenden, falls die Quelle einer Zusammenführung auf einen beibehaltenen Pfad durch das Filtern entfernt worden ist.

`--preserve-revprops`
Falls alle Knoten einer Revision durch das Filtern entfernt werden und die Option `--drop-empty-revs` nicht übergeben wird, verhält sich **svndumpfilter** standardmäßig so, dass alle Revisions-Eigenschaften außer dem Datum und der Protokollnachricht entfernt werden (was lediglich darauf hindeutet, dass die Revision leer ist). Wird diese Option übergeben, werden bestehende Revisions-Eigenschaften beibehalten (was mehr oder weniger sinnvoll sein kann, da der hiermit verbundene Inhalt sich nicht mehr in der Auszugsdatei befindet).

`--targets` *`FILENAME`*
Veranlasst **svndumpfilter**, zusätzliche Pfad-Präfixe – einen pro Zeile – aus der Datei *`FILENAME`* zu lesen. Das ist vor allem nützlich für komplexe Filteroperationen, die mehr Präfixe erfordern als das Betriebssystem auf einer einzelnen Kommandozeile zulässt.

`--quiet`
Keine Filterstatistiken anzeigen.

svndumpfilter-Unterbefehle

Hier sind die verschiedenen Unterbefehle für das Programm **svndumpfilter**.

Name

svndumpfilter exclude — Knoten mit gegebenen Präfixen aus dem Auszugsstrom herausfiltern.

Aufruf

```
svndumpfilter exclude PATH_PREFIX...
```

Beschreibung

Dies kann verwendet werden, um Knoten aus einer gefilterten Auszugsdatei zu verwerfen, die mit einem oder mehreren *`PATH_PREFIX`*en beginnen.

Optionen

```
--drop-empty-revs
--preserve-revprops
--quiet
--renumber-revs
--skip-missing-merge-sources
--targets ARG
```

Beispiel

Falls wir eine Auszugsdatei eines Projektarchivs haben, das über eine Anzahl verschiedener Verzeichnisse mit Bezug zum Picknicken verfügt, wir jedoch alles *außer* dem Teil im Projektarchiv mit `sandwiches` haben möchten, werden wir nur diesen Pfad verwerfen:

```
$ svndumpfilter exclude sandwiches < dumpfile > filtered-dumpfile

Präfixe ausschließen:
   '/sandwiches'

Revision 0 als 0 übertragen.
Revision 1 als 1 übertragen.
Revision 2 als 2 übertragen.
Revision 3 als 3 übertragen.
Revision 4 als 4 übertragen.

1 Knoten verworfen:
   '/sandwiches'
```

Name

svndumpfilter include — Knoten ohne gegebene Präfixe aus dem Auszugsstrom herausfiltern.

Aufruf

```
svndumpfilter include PATH_PREFIX...
```

Beschreibung

Kann verwendet werden, um Knoten aus einer gefilterten Auszugsdatei einzuschließen, die mit einem oder mehreren *`PATH_PREFIX`*en beginnen.

Optionen

```
--drop-empty-revs
--preserve-revprops
--quiet
--renumber-revs
--skip-missing-merge-sources
--targets ARG
```

Beispiel

Falls wir eine Auszugsdatei eines Projektarchivs haben, das über eine Anzahl verschiedener Verzeichnisse mit Bezug zum Picknicken verfügt, wir jedoch nur den Teil im Projektarchiv mit `sandwiches` haben möchten, werden wir nur diesen Pfad einschließen:

```
$ svndumpfilter include sandwiches < dumpfile > filtered-dumpfile

Präfixe einschließen:
   '/sandwiches'

Revision 0 als 0 übertragen.
Revision 1 als 1 übertragen.
Revision 2 als 2 übertragen.
Revision 3 als 3 übertragen.
Revision 4 als 4 übertragen.

3 Knoten verworfen:
   '/drinks'
   '/snacks'
   '/supplies'
```

Name

svndumpfilter help — Hilfe!

Aufruf

```
svndumpfilter help [SUBCOMMAND...]
```

Beschreibung

Zeigt die Hilfenachricht für **svndumpfilter** an. Anders als bei anderen in diesem Kapitel dokumentierten Hilfebefehlen, gibt es für diesen Hilfebefehl keinen geistreichen Kommentar. Die Autoren dieses Buchs bedauern diese Auslassung zutiefst.

Optionen

Keine

svnversion – Subversion Arbeitskkopie-Versions-Information

Name

svnversion — Die lokale(n) Revision(en) einer Arbeitskopie zusammenfassen.

Aufruf

```
svnversion [OPTIONS] [WC_PATH [TRAIL_URL]]
```

Beschreibung

svnversion ist ein Programm, um die Revisionsmischung einer Arbeitskopie zusammenzufassen. Als Ergebnis wird die Revisionsnummer oder ein Bereich von Revisionen in die Standardausgabe geschrieben.

Gewöhnlich wird diese Ausgabe in Ihrem Build-Prozess verwendet, um die Versionsnummer Ihres Programms zu definieren.

Falls angegeben, ist *TRAIL_URL* der hintere Teil des URL, der verwendet wird, um festzustellen, ob *WC_PATH* selbst umgestellt ist (die Erkennung von Umstellungen innerhalb von *WC_PATH* hängt nicht von *TRAIL_URL* ab).

Wenn *WC_PATH* nicht definiert ist, wird das aktuelle Verzeichnis als Arbeitskopiepfad herangezogen. *TRAIL_URL* kann nicht definiert werden, ohne dass *WC_PATH* explizit angegeben ist.

Optionen

Ebenso wie **svnserve** besitzt **svnversion** keine Unterbefehle – lediglich Optionen:

`--no-newline` (`-n`)
: Der sonst übliche Zeilenvorschub in der Ausgabe wird weggelassen.

`--committed` (`-c`)
: Verwendet die zuletzt geänderten Revisionen statt der aktuellen (d.h., der höchsten lokal verfügbaren) Revisionen.

`--help` (`-h`)
: Gibt eine zusammenfassende Hilfe aus.

`--version`
: Gibt die Version von **svnversion** aus und beendet sich ohne Fehler.

Beispiele

Falls die gesamter Arbeitskopie dieselbe Revision hat (etwa unmittelbar nach einer Aktualisierung), wird diese Revision ausgegeben:

```
$ svnversion
4168
```

Sie können *TRAIL_URL* hinzufügen, um sicherzustellen, dass die Arbeitskopie nicht wider Erwarten umgestellt wurde. Beachten Sie, dass *WC_PATH* für diesen Befehl erforderlich ist:

```
$ svnversion . /var/svn/trunk
4168
```

Für eine Arbeitskopie mit gemischten Revisionen wird der Bereich der vorhandenen Revisionen ausgegeben:

```
$ svnversion
4123:4168
```

Falls die Arbeitskopie Änderungen enthält, wird ein 'M' angefügt:

```
$ svnversion
4168M
```

Falls die Arbeitskopie umgestellt ist, wird ein 'S' angefügt:

```
$ svnversion
4168S
```

svnversion teilt Ihnen auch mit, falls es sich bei der Ziel-Arbeitskopie um einen Verzeichnis-Teilbaum handelt (siehe „Verzeichnis-Teilbäume“), indem der Code 'P' angehängt wird:

```
$ svnversion
4168P
```

Hier ist also eine Arbeitskopie als Verzeichnis-Teilbaum mit gemischten Revisionen, die umgestellt wurde und Änderungen enthält:

```
$ svnversion
4123:4168MSP
```

Falls es in einem Verzeichnis aufgerufen wird, das keine Arbeitskopie ist, nimmt **svnversion** an, es sei eine exportierte Arbeitskopie und gibt „exported“ aus:

```
$ svnversion
exported
```

mod_dav_svn – Subversion Apache HTTP-Server-Modul

Name

mod_dav_svn-Konfigurationsanweisungen — Apache Konfigurationsanweisungen, um Subversion-Projektarchive über den Apache-HTTP-Server bereitzustellen.

Beschreibung

Dieser Abschnitt beschreibt kurz jede Apache-Konfigurationsanweisung für Subversion. Für eine tiefer gehende Beschreibung zur Konfigurierung von Apache für Subversion, siehe „httpd, der Apache HTTP-Server“.

Anweisungen

Das sind die Anweisungen für `httpd.conf`, die sich auf **mod_dav_svn** beziehen:

`DAV svn`
: Muss in jedem `Directory`- oder `Location`-Abschnitt für ein Subversion-Projektarchiv enthalten sein. Sie fordert **httpd** auf, das Subversion-Backend von `mod_dav` zur Auftragsabwicklung zu verwenden.

`SVNActivitiesDB` *`directory-path`*
: Bestimmt den Ort im Dateisystem, an dem die Datenbank für Aktivitäten abgelegt werden soll. Standardmäßig erzeugt und verwendet **mod_dav_svn** ein Verzeichnis im Projektarchiv namens `dav/activities.d`. Der durch diese Option angegebene Pfad muss absolut sein.

  Falls ein `SVNParentPath`-Bereich angegeben wurde, hängt **mod_dav_svn** den Basisnamen des Projektarchivs an diesen Pfad an, beispielsweise:

```
<Location /svn>
  DAV svn

  # jeder "/svn/foo" URL wird auf ein Projektarchiv in
  # /net/svn.nfs/repositories/foo abgebildet
  SVNParentPath         "/net/svn.nfs/repositories"

  # jeder "/svn/foo" URL wird auf eine Aktivitäten-Datenbank in
  #  /var/db/svn/activities/foo abgebildet
  SVNActivitiesDB       "/var/db/svn/activities"
</Location>
```

`SVNAllowBulkUpdates On|Off`
: Ändert die Unterstützung für vollständige Antworten auf `REPORT`-Anfragen im Stil von Aktualisierungen. Subversion-Clients verwenden `REPORT`-Anfragen, um von **mod_dav_svn** Informationen über Checkouts und Aktualisierungen von Verzeichnisbäumen zu erhalten. Dabei kann vom Server verlangt werden, diese Information auf zwei mögliche Weisen zu senden: entweder mit den Informationen zum gesamten Teilbaum in einer umfangreichen Antwort oder als ein *Skelta* (eine skelettierte Repräsentation eines Baum-Deltas), das dem Client gerade genug Informationen liefert, so dass er weiß, welche *zusätzlichen* Daten er vom Server abfragen muss. Wird diese Direktive mit dem Wert `Off` versehen, werden `REPORT`-Anfragen von **mod_dav_svn** ausschließlich mit Skeltas beantwortet, egal welche Art der Antwort vom Client verlangt wurde.

  Die Meisten werden diese Direktive überhaupt nicht benötigen. Sie existiert hauptsächlich für Administratoren, die – aus Gründen der Sicherheit oder Nachprüfbarkeit – Subversion-Clients dazu zwingen möchten, alle für Checkouts oder Aktualisierungen benötigten Dateien und Verzeichnisse individuell abzurufen, um somit eine Spur aus `GET`- und `PROPFIND`-Anfragen in den Protokolldateien von Apache zu hinterlassen. Der Standardwert dieser Direktive ist `On`.

`SVNAutoversioning On|Off`
: Wenn der Wert `On` ist, führen Schreibanfragen von WebDAV-Clients zu automatischen Übergaben ins Projektarchiv. Eine automatisch erzeugte, generische Protokollnachricht wird mit jeder Revision verknüpft. Falls Sie automatische

Versionierung ermöglichen, werden Sie sicherlich auch `ModMimeUsePathInfo On` setzen wollen, so dass `mod_mime` `svn:mime-type` automatisch auf den richtigen MIME-Typen setzen kann (natürlich nur so gut, wie es `mod_mime` kann). Für weitere Informationen, siehe Anhang C, *WebDAV und Autoversionierung*. Der Standardwert dieser Direktive ist `Off`.

`SVNIndexXSLT` *`directory-path`*
Spezifiziert den URI einer XSL-Transformation für Verzeichnisindexe. Diese Direktive ist optional.

`SVNListParentPath On|Off`
Wenn sie auf `On` gesetzt ist, wird ein `GET` von `SVNParentPath` erlaubt, was zu einer Auflistung aller Projektarchive unter diesem Pfad führt. Der Standardwert ist `Off`.

`SVNMasterURI` *`url`*
Der URI des Master-Subversion-Projektarchivs (verwendet für einen Proxy, über den geschrieben wird).

`SVNParentPath` *`directory-path`*
Spezifiziert den Ort im Dateisystem, an dem ein Elternverzeichnis liegt, dessen Kindverzeichnisse Subversion-Projektarchive sind. In einem Konfigurationsblock für ein Subversion-Projektarchiv muss entweder diese Direktive oder `SVNPath` vorhanden sein, jedoch nicht beide.

`SVNPath` *`directory-path`*
Gibt den Ort im Dateisystem an, an dem die Dateien eines Subversion-Projektarchivs liegen. In einem Konfigurationsblock für ein Subversion-Projektarchiv muss entweder diese Direktive oder `SVNParentPath` vorhanden sein, jedoch nicht beide.

`SVNPathAuthz On|Off|short_circuit`
Kontrolliert pfadbasierte Autorisierung, indem Unteranfragen ermöglicht (`On`) oder abgeschaltet (`Off`; siehe „Abstellen pfadbasierter Prüfungen“) werden oder bei **mod_authz_svn** direkt nachgefragt wird (`short_circuit`). Der Standardwert dieser Direktive ist `On`.

`SVNReposName` *`name`*
Spezifiziert den Namen eines Subversion-Projektarchivs zur Verwendung für `HTTP GET`-Antworten. Dieser Wert wird dem Titel aller Verzeichnisauflistungen vorangestellt (die übertragen werden, wenn Sie mit einem Browser zu einem Subversion-Projektarchiv navigieren). Diese Direktive ist optional.

`SVNSpecialURI` *`component`*
Spezifiziert die URI-Komponente (Namensraum) für besondere Subversion-Ressourcen. Der Standardwert ist `!svn`, und die meisten Administratoren werden diese Direktive nie verwenden. Setzen Sie sie nur, falls die dringende Notwendigkeit besteht, eine Datei namens `!svn` in Ihrem Projektarchiv zu haben. Falls Sie diese Direktive auf einem Server ändern, der bereits in Gebrauch ist, werden alle offenstehenden Arbeitskopien unbrauchbar gemacht, und Ihre Benutzer werden Sie mit Mistgabeln und Fackeln zur Strecke bringen.

mod_authz_svn – Subversion Apache HTTP-Autorisierungs-Modul

Name

mod_authz_svn-Konfigurationsanweisungen — Apache Konfigurationsanweisungen für die Konfigurierung pfadbasierter Autorisierung für Subversion-Projektarchive, die über den Apache HTTP-Server bedient werden.

Beschreibung

Dieser Abschnitt beschreibt kurz jede Apache-Konfigurationsanweisung, die **mod_authz_svn** bietet. Für eine tiefgehende Beschreibung zur Verwendung pfadbasierter Autorisierung in Subversion, siehe „Pfadbasierte Autorisierung“.

Anweisungen

Dies sind die `httpd.conf`-Direktiven, die für **mod_authz_svn** gültig sind:

`AuthzForceUsernameCase Upper|Lower`
: Entweder `Upper` oder `Lower`, um die entsprechende Umwandlung des authentifizierten Anwendernamens vor der Autorisierungsprüfung in Klein- oder Großschreibung vorzunehmen. Während Anwendernamen unter Beachtung der Klein- oder Großschreibung mit denjenigen in der Autorisierungs-Regel-Datei verglichen werden, kann diese Direktive zumindest gemischt geschriebene Anwendernamen in irgendetwas konsistentes normalisieren.

`AuthzSVNAccessFile` *`file-path`*
: Für Zugriffsregeln, die Rechte auf Pfaden im Subversion-Projektarchiv beschreiben, soll in *`file-path`* nachgesehen werden.

`AuthzSVNAnonymous On|Off`
: Um zwei besondere Verhaltensweisen dieses Moduls zu unterbinden, sollte diese Direktive auf `Off` gesetzt werden: Die Interaktion mit der `Satisfy Any`-Anweisung und das Durchsetzen des Autorisierungsverfahrens auch dann, wenn keine `Require`-Direktiven vorhanden sind. Der Standardwert dieser Direktive ist `On`.

`AuthzSVNAuthoritative On|Off`
: Auf `Off` gesetzt, wird die Weiterleitung der Zugriffskontrolle an tiefere Module erlaubt. Der Standardwert dieser Direktive ist `On`.

`AuthzSVNNoAuthWhenAnonymousAllowed On|Off`
: Auf `On` setzen, um die Authentifizierung und Autorisierung von Anfragen zu unterbinden, die anonyme Benutzer durchführen dürfen. Der Standardwert dieser Direktive ist `On`.

Subversion-Eigenschaften

Subversion gestattet seinen Benutzern, beliebig benannte, versionierte Eigenschaften für Dateien und Verzeichnisse sowie unversionierte Eigenschaften für Revisionen zu erfinden. Die einzige Einschränkung gilt für Eigenschaften, die mit `svn:` beginnen (diese sind für die Benutzung durch Subversion reserviert). Obwohl diese Eigenschaften durch Benutzer geändert werden dürfen, um das Verhalten von Subversion zu steuern, dürfen Benutzer keine neuen `svn:`-Eigenschaften erfinden.

Versionierte Eigenschaften

Dies sind die versionierten Eigenschaften, die Subversion für die eigene Verwendung reserviert:

`svn:executable`
: Falls diese Eigenschaft für eine Datei vergeben ist, wird der Client die Datei in Arbeitskopien auf Unix-Wirtssystemen ausführbar machen. Siehe „Ausführbarkeit von Dateien“.

`svn:mime-type`
: Falls diese Eigenschaft für eine Datei vergeben ist, zeigt der Wert den MIME-Typen der Datei an. Das erlaubt dem Client die Entscheidung, ob es während einer Aktualisierung sicher ist, eine zeilenbasierte Zusammenführung durchzuführen; es kann auch Auswirkungen auf das Verhalten der Datei haben, wenn sie über einen Web-Browser geladen wird. Siehe „Datei-Inhalts-Typ“.

`svn:ignore`
Falls diese Eigenschaft für ein Verzeichnis vergeben ist, enthält der Wert eine Liste von Namensmustern *unversionierter* Dateien, die von **svn status** und anderen Unterbefehlen zu ignorieren sind. Siehe „Ignorieren unversionierter Objekte“.

`svn:keywords`
Falls diese Eigenschaft für eine Datei vergeben ist, teilt dessen Wert dem Client mit, wie bestimmte Schlüsselworte in der Datei zu expandieren sind. Siehe „Ersetzung von Schlüsselworten“.

`svn:eol-style`
Falls diese Eigenschaft für eine Datei vergeben ist, teilt dessen Wert dem Client mit, wie die Zeilenenden der Datei in der Arbeitskopie und in exportierten Bäumen zu behandeln sind. Siehe „Zeichenfolgen zur Zeilenende-Kennzeichnung“ und svn export weiter oben in diesem Kapitel.

`svn:externals`
Falls diese Eigenschaft für ein Verzeichnis vergeben ist, besteht dessen Wert aus einer Liste anderer Pfade und URLs, die der Client auschecken soll. Siehe „Externals-Definitionen“.

`svn:special`
Falls diese Eigenschaft für eine Datei vergeben ist, zeigt es an, dass die Datei keine gewöhnliche Datei ist, sondern ein symbolischer Link oder ein anderes besonderes Objekt.[1]

`svn:needs-lock`
Falls diese Eigenschaft für eine Datei vergeben ist, teilt es dem Client mit, die Datei in der Arbeitskopie schreibgeschützt abzulegen, um daran zu erinnern, die Datei vor der Bearbeitung zu sperren. Siehe „Kommunikation über Sperren“.

`svn:mergeinfo`
Wird von Subversion verwendet, um Zusammenführungsdaten festzuhalten. Siehe „Zusammenführungsinformation und Vorschauen“ für Details, jedoch sollten Sie diese Eigenschaft nie bearbeiten, es sei denn, Sie wissen *wirklich* was Sie machen.

Unversionierte Eigenschaften

Dies sind die unversionierten Eigenschaften, die Subversion für die eigene Verwendung reserviert:

`svn:author`
Falls vorhanden, beinhaltet es den authentifizierten Anwendernamen der Person, die die Revision erzeugt hat. (Falls nicht vorhanden, wurde die Revision anonym übergeben.)

`svn:autoversioned`
Falls vorhanden, wurde die Revision durch die Autoversionierungs-Funktion erzeugt. Siehe „Autoversionierung“.

`svn:date`
Beinhaltet den Erstellungszeitpunkt der Revision in UTC-Zeit im ISO-8601-Format. Der Wert kommt von der Uhr der *Server*-Maschine, nicht der des Clients.

`svn:log`
Beinhaltet die Protokollnachricht, die die Revision beschreibt.

`svn:sync-currently-copying`
Beinhaltet die Revisionsnummer des Quell-Projektarchivs, das momentan mit dem Programm **svnsync** in dieses gespiegelt wird. (Diese Eigenschaft ist nur relevant, wenn sie bei Revision 0 gesetzt ist.)

`svn:sync-from-uuid`
Beinhaltet die UUID des Projektarchivs, durch das dieses Projektarchiv mit dem Programm **svnsync** als Spiegel initialisiert wurde. (Diese Eigenschaft ist nur relevant, wenn sie bei Revision 0 gesetzt ist.)

`svn:sync-from-url`
Beinhaltet den URL des Projektarchivs, durch das dieses Projektarchiv mit dem Programm **svnsync** als Spiegel initialisiert

[1] Zum Zeitpunkt der Drucklegung sind symbolische Links tatsächlich die einzigen „besonderen“ Objekte. Allerdings könnten in künftigen Ausgaben von Subversion mehr davon hinzukommen.

wurde. (Diese Eigenschaft ist nur relevant, wenn sie bei Revision 0 gesetzt ist.)

`svn:sync-last-merged-rev`
: Beinhaltet die Revision des Projektarchivs, das zuletzt erfolgreich in dieses gespiegelt wurde. (Diese Eigenschaft ist nur relevant, wenn sie bei Revision 0 gesetzt ist.)

`svn:sync-lock`
: Wird verwendet, um vorübergehend einen gegenseitigen Ausschluss für den Zugriff auf das Projektarchiv durch Spiegelungsoperationen von **svnsync** zu erzwingen. Diese Eigenschaft wird im Allgemeinen nur dann beachtet, falls eine solche Aktion aktiv ist, oder falls ein **svnsync**-Befehl es nicht schaffte, sich sauber vom Projektarchiv abzumelden. (Diese Eigenschaft ist nur relevant, wenn sie bei Revision 0 gesetzt ist.)

Projektarchiv-Hooks

Dies sind die von Subversion zur Verfügung gestellten Projektarchiv-Hooks:

Name

start-commit — Ankündigung des Beginns einer Übergabe.

Beschreibung

Der Hook start-commit wird ausgeführt, bevor überhaupt eine Übergabetransaktion erzeugt wird. Er wird üblicherweise verwendet, um zu entscheiden, ob der Benutzer überhaupt die Rechte zur Übergabe hat.

Falls der Hook start-commit einen Rückgabewert ungleich Null zurückgibt, wird die Übergabe gestoppt, bevor die Übergabetransaktion überhaupt erzeugt wird, und alles, was nach `stderr` ausgegeben wird, wird zurück zum Client umgeleitet.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Authentifizierter Name des Benutzers, der die Übergabe versucht
3. Eine durch Doppelpunkte getrennte Liste von Fähigkeiten, die der Client an den Server übergibt; dazu gehören `depth`, `mergeinfo` und `log-revprops` (neu in Subversion 1.5).

Übliche Verwendung

Zugriffskontrolle (d.h., für das vorübergehende Sperren von Übergaben aus bestimmten Gründen).

Eine Methode, um den Zugriff nur Clients mit bestimmten Fähigkeiten zu ermöglichen.

Name

pre-commit — Ankündigung kurz vor Abschluss der Übergabe.

Beschreibung

Der Hook `pre-commit` wird ausgeführt, kurz bevor eine Übergabetransaktion zu einer neuen Revision wird. Üblicherweise wird dieser Hook dazu verwendet, um Übergaben abzuwenden, die aufgrund ihres Inhalts oder des Ortes nicht zulässig sind (z.B., könnte Ihr Standort verlangen, dass alle Übergaben auf einen bestimmten Zweig eine Ticketnummer des Fehlerverfolgungssystems beinhalten, oder dass die Protokollnachricht nicht leer ist).

Falls das Hook-Programm `pre-commit` einen Rückgabewert ungleich Null zurückgibt, wird die Übergabe abgebrochen, die Übergabetransaktion entfernt und alles, was nach `stderr` ausgegeben wird, zum Client umgeleitet.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Name der Übergabetransaktion

Darüber hinaus leitet Subversion alle vom übertragenden Client eingereichten Sperrmarken über die Standard-Eingabe an das Hook-Skript. Falls vorhanden, werden diese als eine Zeile formatiert, die die Zeichenkette `LOCK-TOKENS:` beinhaltet, gefolgt von zusätzlichen Zeilen, eine pro Sperrmarke, die die Informationen zur Sperrmarke enthält. Jede Zeile mit Sperrmarken-Informationen besteht aus dem mit der Sperre verbundenen, URI-maskierten Projektarchiv-Pfad, gefolgt vom senkrechten Strich (`|`) als Trennzeichen und schließlich der Zeichenkette der Sperrmarke.

Übliche Verwendung

Validierung und Kontrolle von Änderungen

Name

post-commit — Ankündigung einer erfolgreichen Übergabe.

Beschreibung

Der Hook `post-commit` wird ausgeführt, nachdem die Transaktion abgeschlossen wurde und eine neue Revision erzeugt worden ist. Die meisten Leute verwenden diesen Hook, um E-Mails zu versenden, die diese Übergabe beschreiben oder um irgendein Werkzeug von der Übergabe in Kenntnis zu setzen (etwa ein Fehlerverfolgungssystem). Manche Konfigurationen verwenden diesen Hook auch für Sicherungsprozesse.

Falls der Hook `post-commit` einen Rückgabewert ungleich Null zurückliefert, wird die Übergabe *nicht* abgebrochen, da sie bereits abgeschlossen ist. Trotzdem wird alles, was der Hook über `stderr` ausgibt, zum Client umgeleitet, um die Fehlersuche zu erleichtern.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Die durch die Übergabe erzeugte Revisionsnummer

Übliche Verwendung

Übergabebenachrichtigung; Werkzeugintegration

Name

pre-revprop-change — Ankündigung des Versuchs einer Änderung eines Revisions-Eigenschaften.

Beschreibung

Der Hook `pre-revprop-change` wird unmittelbar vor der Änderung einer Revisions-Eigenschaft außerhalb einer gewöhnlichen Übergabe ausgeführt. Anders als bei den anderen Hooks ist das Standardvorgehen dieses Hooks, die beabsichtigte Aktion zu verhindern. Der Hook muss wirklich vorhanden sein und einen Rückgabewert von Null zurückgeben, bevor eine Änderung einer Revisions-Eigenschaft stattfinden kann.

Falls der Hook `pre-revprop-change` nicht vorhanden ist, nicht ausführbar ist oder einen Rückgabewert ungleich Null liefert, wird keine Änderung an der Eigenschaft durchgeführt, und alles, was über `stderr` ausgegeben wird, zurück zum Client umgeleitet.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Revision, deren Eigenschaft geändert werden soll
3. Authentifizierter Name des Benutzers, der die Änderung an der Eigenschaft vornehmen will
4. Name der geänderten Eigenschaft
5. Beschreibung der Änderung: `A` (hinzugefügt), `D` (gelöscht) oder `M` (geändert)

Darüber hinaus übergibt Subversion den beabsichtigten neuen Wert der Eigenschaft über Standardeingabe an das Hook-Programm.

Übliche Verwendung

Zugriffskontrolle; Validierung und Kontrolle von Änderungen

Name

post-revprop-change — Ankündigung einer erfolgreichen Änderung einer Revisions-Eigenschaft.

Beschreibung

Der Hook `post-revprop-change` wird unmittelbar nach der Änderung einer Revisions-Eigenschaft außerhalb einer normalen Übergabe ausgeführt. Wie Sie aus der Beschreibung seines Gegenstücks, dem Hook `pre-revprop-change`, ableiten können, wird dieser Hook ohne einen implementierten Hook `pre-revprop-change` überhaupt nicht ausgeführt. Er wird gewöhnlich verwendet, um Benachrichtigungen über die Eigenschafts-Änderung per E-Mail zu versenden.

Falls der Hook `post-revprop-change` einen Rückgabewert ungleich Null liefert, wird die Änderung *nicht* abgebrochen, da sie bereits abgeschlossen ist. Allerdings wird alles, was der Hook über `stderr` ausgibt, zum Client umgeleitet, um die Fehlersuche zu erleichtern.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Revision, deren Eigenschaft geändert wurde
3. Authentifizierter Name des Benutzers, der die Änderung vorgenommen hat
4. Name der geänderten Eigenschaft
5. Beschreibung der Änderung: `A` (hinzugefügt), `D` (gelöscht), or `M` (geändert)

Darüberhinaus übergibt Subversion den beabsichtigten neuen Wert der Eigenschaft über Standardeingabe an das Hook-Programm.

Übliche Verwendung

Benachrichtigung über Eigenschafts-Änderung

Name

pre-lock — Ankündigung des Versuchs einer Pfadsperrung.

Beschreibung

Der Hook `pre-lock` wird ausgeführt, sobald jemand versucht, einen Pfad zu sperren. Er kann verwendet werden, um Sperren ganz zu verhindern oder eine kompliziertere Vorgehensweise festzulegen, bei der genau spezifiziert wird, welche Benutzer bestimmte Pfade sperren dürfen. Falls der Hook eine bereits bestehende Sperre bemerkt, kann er auch entscheiden, ob der Benutzer die bestehende Sperre „stehlen“ darf.

Falls das Hook-Programm `pre-lock` einen Rückgabewert ungleich Null liefert, wird der Sperrvorgang abgebrochen und alles, was über `stderr` ausgegeben wird, zum Client umgeleitet.

Das Hook-Programm darf optional die Sperrmarke bestimmen, die der Sperre zugewiesen wird, indem die gewünschte Sperrmarke zur Standard-Ausgabe geschickt wird. Daher sollten Implementierungen dieses Hooks sorgfältig darauf achten, keine unerwarteten Ausgaben an die Standard-Ausgabe zu schicken.

Falls das `pre-lock`-Skript von der Möglichkeit der Bestimmung der Sperrmarke Gebrauch macht, liegt die Verantwortung zur Erzeugung einer *eindeutigen* Sperrmarke beim Skript selbst. Die Erstellung einer nicht eindeutigen Sperrmarke kann zu undefinierten – und sehr wahrscheinlich unerwünschten – Verhalten führen.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Versionierter Pfad, der gesperrt werden soll
3. Authentifizierter Name des Benutzers, der sperren möchte
4. Kommentar bei Erstellung der Sperre
5. `1` falls der Anwender versucht, eine bestehende Sperre zu stehlen, sonst `0`

Übliche Verwendung

Zugriffskontrolle

Name

post-lock — Ankündigung einer erfolgreichen Pfadsperrung.

Beschreibung

Der Hook `post-lock` wird ausgeführt, nachdem ein oder mehrere Pfade gesperrt worden sind. Er wird üblicherweise verwendet, um Benachrichtigungen über die Sperre per E-Mail zu versenden.

Falls der Hook `post-lock` einen Rückgabewert ungleich Null liefert, wird die Sperre *nicht* abgebrochen, da sie bereits besteht. Allerdings wird alles, was der Hook über `stderr` ausgibt, zum Client umgeleitet, um die Fehlersuche zu erleichtern.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Authentifizierter Name des Benutzers, der die Pfade gesperrt hat

Darüber hinaus übergibt Subversion die Liste der gesperrten Pfade über Standardeingabe an das Hook-Programm.

Übliche Verwendung

Sperrbenachrichtigung

Name

pre-unlock — Ankündigung der Aufhebung einer Pfadsperre.

Beschreibung

Der Hook `pre-unlock` wird immer dann ausgeführt, wenn jemand versucht, eine Dateisperre aufzuheben. Er kann verwendet werden, um Vorgehensweisen festzulegen, die es Benutzern erlauben, bestimmte Pfade zu entsperren. Er ist besonders wichtig, um Strategien festzulegen, wenn es darum geht, Sperren aufzubrechen. Falls Benutzer A eine Datei sperrt, soll dann Benutzer B diese Sperre aufbrechen dürfen? Was ist, wenn die Sperre älter als eine Woche ist? Diese Dinge können von diesem Hook entschieden und durchgesetzt werden.

Falls das Hook-Programm `pre-unlock` einen Rückgabewert ungleich Null liefert, wird die Aufhebung der Sperre abgebrochen, und alles, was über `stderr` ausgegeben wird, zum Client umgeleitet.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Versionierter Pfad, dessen Sperre aufgehoben werden soll
3. Authentifizierter Name des Benutzers, der die Sperre aufheben will
4. Die mit der aufzuhebenden Sperre verbundene Sperrmarke
5. `1` falls der Anwender versucht, die Entsperrung einer bestehende Sperre zu erzwingen, sonst `0`

Übliche Verwendung

Zugriffskontrolle

Name

post-unlock — Ankündigung einer erfolgreichen Aufhebung einer Pfadsperre.

Beschreibung

Der Hook `post-unlock` wird ausgeführt, nachdem ein oder mehrere Pfade entsperrt worden sind. Er wird üblicherweise verwendet, um Benachrichtigungen über die Aufhebung der Sperre per E-Mail zu versenden.

Falls der Hook `post-unlock` einen Rückgabewert ungleich Null liefert, wird die Aufhebung der Sperre *nicht* abgebrochen, da sie bereits aufgehoben ist. Allerdings wird alles, was der Hook über `stderr` ausgibt, zum Client umgeleitet, um die Fehlersuche zu erleichtern.

Eingabeparameter

Die Kommandozeilenparameter, die an das Hook-Programm übergeben werden, sind in der gegebenen Reihenfolge:

1. Projektarchiv-Pfad
2. Authentifizierter Name der Person, die die Pfadsperre aufgehoben hat

Darüber hinaus übergibt Subversion die Liste der entsperrten Pfade über Standardeingabe an das Hook-Programm.

Übliche Verwendung

Benachrichtigung über die Aufhebung von Sperren

Anhang A. Subversion-Einführung für einen schnellen Start

Falls Sie Subversion unbedingt zum Laufen bringen wollen (und Sie der Typ sind, der durch Experimentieren lernt), erklärt Ihnen dieser Anhang, wie ein Projektarchiv erstellt, Code importiert und wieder als Arbeitskopie ausgecheckt wird. Dabei zeigen wir Verbindungen zu den relevanten Kapiteln in diesem Buch auf.

Falls das gesamte Konzept der Versionskontrolle oder das von CVS und Subversion verwendete „Copy-Modify-Merge“-Modell völliges Neuland für Sie sein sollte, empfiehlt sich vor dem Weiterlesen die Lektüre von Kapitel 1, *Grundlegende Konzepte*.

Subversion installieren

Subversion ist auf einer Portabilitätsschicht namens APR – die Bibliothek Apache Portable Runtime – aufgebaut. Die APR-Bibliothek liefert alle Schnittstellen, die Subversion benötigt, um auf verschiedenen Betriebssystemen lauffähig zu sein: Plattenzugriff, Netzzugriff, Speicherverwaltung usw. Obwohl Subversion Apache HTTP-Server (oder **httpd**) als eines seiner Netz-Server-Programme verwenden kann, bedeutet seine Abhängigkeit von APR *nicht*, dass **httpd** eine notwendige Komponente ist. APR ist eine selbständige Bibliothek, die von jeder Anwendung verwendet werden kann. Es bedeutet allerdings, dass Subversion-Clients und -Server auf allen Betriebssystemen lauffähig sind, auf denen **httpd**-Server läuft: Windows, Linux, alle BSD-Varianten, Mac OS X, NetWare und andere.

Der einfachste Weg, sich Subversion zu beschaffen, ist ein Binärpaket für Ihr Betriebssystem herunterzuladen. Oft sind diese Pakete über die Webpräsenz von Subversion (http://subversion.apache.org) erhältlich, wo sie von Freiwilligen hinterlegt wurden. Dort finden Benutzer von Microsoft-Betriebssystemen normalerweise graphische Installationspakete. Wenn Sie ein Unix-ähnliches Betriebssystem verwenden, können Sie das eigene Paketverteilungssystem Ihres Betriebssystems verwenden (RPMs, DEBs, Ports Tree usw.), um an Subversion zu gelangen.

Alternativ können Sie Subversion direkt aus dem Quelltext bauen, obwohl es nicht immer ein leichtes Unterfangen ist. (Falls Sie keine Erfahrung mit dem Bauen von Open-Source-Softwarepaketen haben, ist es wahrscheinlich besser für Sie, stattdessen ein Binärpaket herunterzuladen!) Laden Sie die neueste Ausgabe des Quelltextes von der Subversion-Webpräsenz. Folgen Sie nach dem Entpacken zum Bauen den Anweisungen in der Datei `INSTALL`. Beachten Sie, dass ein herausgegebenes Quelltextpaket unter Umständen nicht alles enthält, was Sie benötigen, um einen Kommandozeilen-Client zu bauen, der mit einem entfernten Projektarchiv kommunizieren kann. Ab Subversion 1.4 werden die Bibliotheken, von denen Subversion abhängt (apr, apr-util und neon) als eigene Quelltextpakete mit dem Suffix `-deps` verteilt. Diese Bibliotheken sind heutzutage verbreitet genug, um vielleicht bereits auf Ihrem System installiert zu sein. Falls nicht, müssen Sie diese Pakete in dasselbe Verzeichnis entpacken, in das Sie den Hauptquelltext von Subversion entpackt haben. Trotzdem ist es möglich, dass Sie sich weitere optionale abhängige Pakete besorgen möchten, wie etwa Berkeley DB und möglicherweise Apache **httpd**. Falls Sie Subversion vollständig erstellen möchten, stellen Sie sicher, dass Sie alle in der Datei `INSTALL` dokumentierten Pakete haben.

Falls Sie einer der Zeitgenossen sind, die gerne die allerneueste Software verwenden, können Sie auch den Quelltext von Subversion aus dem Subversion-Projektarchiv bekommen, in dem er verwaltet wird. Offensichtlich benötigen Sie hierfür bereits einen Subversion-Client. Doch sobald Sie einen haben, können Sie eine Arbeitskopie von http://svn.apache.org/repos/asf/subversion auschecken:[1]

```
$ svn checkout http://svn.apache.org/repos/asf/subversion/trunk subversion
A    subversion/HACKING
A    subversion/INSTALL
A    subversion/README
A    subversion/autogen.sh
A    subversion/build.conf
…
```

[1]Beachten Sie, dass der ausgecheckte URL im Beispiel nicht auf `subversion` sondern auf eine Unterverzeichnis hiervon namens `trunk` endet. Der Grund hierfür findet sich bei der Erörterung des Verzweigungs- und Etikettierungsmodells von Subversion.

Der obige Befehl erzeugt eine Arbeitskopie des neuesten (unveröffentlichten) Quelltextes von Subversion im Unterverzeichnis `subversion` Ihres aktuellen Arbeitsverzeichnisses. Sie können das letzte Argument ganz nach Ihren Bedürfnissen anpassen. Egal wie Sie das Verzeichnis mit der neuen Arbeitskopie nennen, nach Abschluss der Operation haben Sie den Quelltext von Subversion. Natürlich müssen Sie noch ein paar Hilfsbibliotheken besorgen (apr, apr-util, etc.) – Näheres dazu in der Datei `INSTALL` im obersten Verzeichnis der Arbeitskopie.

Schnellstart-Lehrgang

> „Bitte stellen Sie sicher, dass Ihre Lehnen aufrecht stehen und die Tabletts eingeklappt und verriegelt sind! Flugbegleiter, bitte für den Start vorbereiten…."

Es folgt eine schnelle Einführung, die Sie durch einige grundlegende Einstellungen und Funktionen von Subversion führt. Nach Abschluss sollten Sie ein allgemeines Verständnis über die Verwendung von Subversion haben.

Die Beispiele in diesem Anhang gehen davon aus, dass Ihnen **svn**, der Subversion-Kommandozeilen-Client, und **svnadmin**, das Verwaltungswerkzeug, auf einer Unix-ähnlichen Umgebung zur Verfügung stehen. (Dieser Lehrgang funktioniert auch auf der Windows-Kommandozeile, sofern Sie einige offensichtliche Anpassungen vornehmen.) Wir gehen auch davon aus, dass Sie Subversion 1.2 oder neuer verwenden (rufen Sie **`svn --version`** zur Überprüfung auf).

Subversion speichert alle versionierten Daten in einem zentralen Projektarchiv. Um zu beginnen, erstellen Sie ein neues Projektarchiv:

```
$ cd /var/svn
$ svnadmin create repos
$ ls repos
conf/  dav/  db/  format  hooks/  locks/  README.txt
$
```

Dieser Befehl erzeugt ein Subversion-Projektarchiv im Verzeichnis `/var/svn/repos`, wobei das Verzeichnis `repos` selbst angelegt wird, sofern es noch nicht vorhanden ist. Dieses neue Verzeichnis beinhaltet (neben anderen Dingen) eine Sammlung von Datenbankdateien. Wenn Sie hineinschauen, werden Sie Ihre versionierten Dateien nicht sehen. Weitere Informationen zur Erstellung und Wartung von Projektarchiven finden Sie in Kapitel 5, *Verwaltung des Projektarchivs*.

Subversion kennt kein Konzept „Projekt". Das Projektarchiv ist lediglich ein virtuelles, versioniertes Dateisystem, ein großer Baum, der alles aufnehmen kann, was Sie wollen. Manche Administratoren bevorzugen es, nur ein Projekt in einem Projektarchiv zu speichern, wohingegen andere mehrere Projekte in einem Projektarchiv unterbringen, indem sie sie in getrennten Unterverzeichnissen ablegen. Wir erörtern die Vorteile jedes Ansatzes in „Planung der Organisation Ihres Projektarchivs". So oder so, das Projektarchiv verwaltet nur Dateien und Verzeichnisse, so dass es ganz allein bei den Menschen liegt, bestimmte Verzeichnisse als „Projekte" anzusehen. Auch wenn Sie in diesem Buch Bezüge auf Projekte sehen sollten, denken Sie daran, dass wir dabei nur über irgendein Verzeichnis (oder eine Sammlung von Verzeichnissen) im Projektarchiv sprechen.

In diesem Beispiel gehen wir davon aus, dass Sie bereits so etwas wie ein Projekt (eine Sammlung aus Dateien und Verzeichnissen) haben, die Sie in Ihr frisch erstelltes Subversion-Projektarchiv importieren möchten. Fangen Sie damit an, Ihre Daten innerhalb eines einzelnen Verzeichnisses namens `myproject` (oder ein anderer Wunschname) zu organisieren. Aus Gründen, die in Kapitel 4, *Verzweigen und Zusammenführen* erklärt werden, sollte die Struktur Ihres Projektbaums drei oberste Verzeichnisse namens `branches`, `tags` und `trunk` haben. Das Verzeichnis `trunk` sollte alle Ihre Daten beinhalten, und die Verzeichnisse `branches` und `tags` sollten leer sein:

```
/tmp/
  myproject/
    branches/
    tags/
```

```
trunk/
  foo.c
  bar.c
  Makefile
  …
```

Die Unterverzeichnisse `branches`, `tags` und `trunk` werden von Subversion nicht tatsächlich benötigt. Sie sind eher eine verbreitete Konvention, die sehr wahrscheinlich auch Sie später verwenden wollen.

Sobald Sie Ihre Daten vorbereitet haben, importieren Sie sie mit dem Befehl **svn import** in das Projektarchiv (siehe „Wie Sie Daten in Ihr Projektarchiv bekommen“):

```
$ svn import /tmp/myproject file:///var/svn/repos/myproject \
      -m "initial import"

Hinzufügen     /tmp/myproject/branches
Hinzufügen     /tmp/myproject/tags
Hinzufügen     /tmp/myproject/trunk
Hinzufügen     /tmp/myproject/trunk/foo.c
Hinzufügen     /tmp/myproject/trunk/bar.c
Hinzufügen     /tmp/myproject/trunk/Makefile
…
Revision 1 übertragen.
$
```

Nun enthält das Projektarchiv diesen Baum von Daten. Wie bereits erwähnt, werden Sie Ihre Dateien nicht sehen, wenn Sie direkt in das Projektarchiv schauen; sie werden alle in einer Datenbank abgelegt. Das imaginäre Dateisystem des Projektarchivs jedoch enthält nun ein Verzeichnis namens `myproject`, welches wiederum Ihre Daten enthält.

Beachten Sie, dass das ursprüngliche Verzeichnis `/tmp/myproject` unverändert bleibt; für Subversion bedeutet es nichts. (Sie können das Verzeichnis eigentlich löschen, wenn Sie möchten.) Um damit zu beginnen, Projektarchiv-Daten zu bearbeiten, müssen Sie eine neue „Arbeitskopie“ der Daten anlegen, eine Art privater Arbeitsbereich. Fordern Sie Subversion dazu auf, eine Arbeitskopie des Projektarchiv-Verzeichnisses `myproject/trunk` „auszuchecken“:

```
$ svn checkout file:///var/svn/repos/myproject/trunk myproject
A    myproject/foo.c
A    myproject/bar.c
A    myproject/Makefile
…

Ausgecheckt, Revision 1.
```

Nun haben Sie eine persönliche Kopie eines Teils des Projektarchivs in einem Verzeichnis namens `myproject`. Sie können die Dateien in Ihrer Arbeitskopie bearbeiten und dann diese Änderungen zurück an das Projektarchiv übertragen.

- Gehen Sie in Ihre Arbeitskopie und bearbeiten Sie den Inhalt einer Datei.
- Lassen Sie **`svn diff`** laufen, um eine vereinheitlichte Diff-Ausgabe Ihrer Änderungen zu sehen.
- Rufen Sie **`svn commit`** auf, um die neue Version Ihrer Datei an das Projektarchiv zu übergeben.
- Rufen Sie **`svn update`** auf, um Ihre Arbeitskopie bezüglich des Projektarchivs zu „aktualisieren“.

Eine vollständige Führung durch alles, was Sie mit Ihrer Arbeitskopie machen können, finden Sie in Kapitel 2, *Grundlegende Benutzung*.

An dieser Stelle haben Sie die Möglichkeit, Ihr Projektarchiv für andere über das Netz erreichbar zu machen. Siehe Kapitel 6, *Konfiguration des Servers*, um mehr über die verschiedenen Arten von verfügbaren Server-Prozessen zu erfahren und wie sie konfiguriert werden.

Anhang B. Subversion für CVS-Benutzer

Dieser Anhang ist eine Anleitung für CVS-Benutzer, die in Subversion einsteigen möchten. Im Wesentlichen ist er eine Liste von Unterschieden zwischen den beiden Systemen „aus der Vogelperspektive betrachtet“. Sofern dies möglich ist, geben wir in jedem Abschnitt Verweise auf die entsprechenden Kapitel an.

Obwohl es das Ziel von Subversion ist, den bestehenden und künftigen Benutzerstamm von CVS zu übernehmen, war es nötig, einige neue Funktionen und Änderungen im Entwurf vorzunehmen, um bestimmte „fehlerhafte“ Verhaltensweisen von CVS zu beheben. Das bedeutet, dass Sie sich als CVS-Benutzer einige Angewohnheiten abgewöhnen müssen – einige, von denen Sie vergessen haben, dass sie vor allem merkwürdig waren.

Revisionsnummern sind jetzt anders

In CVS werden Revisionsnummern pro Datei vergeben. Das liegt daran, dass CVS seine Daten in RCS-Dateien speichert; für jede Datei gibt es eine entsprechende RCS-Datei im Projektarchiv, und die Struktur des Projektarchivs entspricht grob der Struktur Ihres Projektbaums.

In Subversion sieht das Projektarchiv aus wie ein einzelnes Dateisystem. Jede Übergabe verursacht einen völlig neuen Dateibaum; im Wesentlichen ist das Projektarchiv eine Liste aus Bäumen. Jeder dieser Bäume wird mit einer einzelnen Revisionsnummer gekennzeichnet. Wenn jemand von „Revision 54“ redet, ist damit ein bestimmter Baum gemeint (und indirekt, wie das Dateisystem nach der 54. Übergabe aussah).

Technisch ist es nicht zulässig, von „Revision 5 von `foo.c`“ zu sprechen. Stattdessen sollte man sagen, „`foo.c` wie es in Revision 5 aussieht“. Seien Sie ebenfalls sorgfältig, wenn Sie Annahmen über die Entwicklung einer Datei machen. In CVS sind die Revisionen 5 und 6 von `foo.c` immer unterschiedlich. In Subversion ist es sehr wahrscheinlich, dass `foo.c` sich zwischen den Revisionen 5 und 6 *nicht* geändert hat.

Auf ähnliche Weise ist in CVS ein Tag oder ein Zweig eine Anmerkung zu der Datei oder zu der Versionsinformation dieser individuellen Datei, wohingegen ein Tag oder ein Zweig in Subversion eine Kopie des gesamten Baums ist (konventionell in die Verzeichnisse `/branches` oder `/tags`, die auf der obersten Ebene des Projektarchivs neben `/trunk` liegen). Im Projektarchiv in seiner Gesamtheit können viele Versionen einer Datei sichtbar sein: die letzte Version jedes Zweigs, jede mit einem Tag versehene Version und natürlich die letzte Version auf dem Stamm. Um den Ausdruck also noch weiter zu präzisieren, würde man sagen „`foo.c` wie es in `/branches/REL1` in Revision 5“ aussieht.

Für weitere Einzelheiten zu diesem Thema, siehe „Revisionen“.

Verzeichnisversionen

Subversion verfolgt Baumstrukturen, nicht nur Dateiinhalte. Dies ist einer der Hauptgründe, warum Subversion geschrieben wurde, um CVS zu ersetzen.

Für Sie als ehemaligen CVS-Benutzer bedeutet das:

- Die Befehle **svn add** und **svn delete** arbeiten nun auf Verzeichnissen wie auf Dateien; ebenso **svn copy** und **svn move**. Jedoch bewirken diese Befehle *keine* sofortige Änderung im Projektarchiv. Stattdessen werden die Objekte einfach zum Hinzufügen oder Löschen „vorgemerkt“. Es findet keine Änderung im Projektarchiv statt, bevor Sie **`svn commit`** aufrufen.

- Verzeichnisse sind nicht mehr dumme Behälter, sondern sie haben Versionsnummern wie Dateien. (Obwohl es genauer ist, von „Verzeichnis `foo/` in Revision 5“ zu sprechen.)

Lassen Sie uns den letzten Punkt etwas genauer erörtern. Die Versionierung von Verzeichnissen ist ein ernstes Problem; da wir Arbeitskopien aus gemischten Revisionen zulassen möchten, gibt es einige Einschränkungen beim Ausreizen dieses Modells.

Vom theoretischen Standpunkt definieren wir „Revision 5 des Verzeichnisses `foo`“ als eine bestimmte Ansammlung von Verzeichniseinträgen und Eigenschaften. Angenommen, wir beginnen nun damit, Dateien dem Verzeichnis `foo` hinzuzufügen und wegzunehmen und diese Änderungen dann zu übergeben. Es wäre eine Lüge, zu behaupten, dass wir immer noch Revision 5 von `foo` hätten. Wenn wir allerdings die Revisionsnummer von `foo` nach der Übergabe erhöht hätten, wäre das auch eine

Lüge; es könnten noch weitere Änderungen an `foo` vorliegen, die wir aber nicht mitbekommen haben, da wir noch nicht aktualisiert haben.

Subversion behandelt dieses Problem, indem es stillschweigend übergebene Hinzufügungen sowie Löschungen im `.svn`-Bereich mitverfolgt. Wenn Sie schließlich **`svn update`** aufrufen, wird alles in Bezug auf das Projektarchiv glattgezogen und die neue Revisionsnummer des Verzeichnisses korrekt vergeben. *Daher kann erst nach einer Aktualisierung gesagt werden, dass es eine „vollständige“ Verzeichnisrevision gibt.* Meist wird Ihre Arbeitskopie „unvollständige“ Verzeichnisrevisionen enthalten.

Auf ähnliche Art ergibt sich ein Problem, wenn Sie versuchen, Eigenschafts-Änderungen an einem Verzeichnis zu übergeben. Normalerweise würde die Übergabe die lokale Revisionsnummer erhöhen, was jedoch eine Lüge wäre, da Hinzufügungen oder Löschungen vorhanden sein könnten, die das Verzeichnis noch nicht mitbekommen hat, da es nicht aktualisiert worden ist. *Deshalb dürfen Sie Änderungen an Verzeichnis-Eigenschaften nicht übergeben, bevor Sie das Verzeichnis aktualisiert haben.*

Für eine weitergehende Erörterung der Einschränkungen der Verzeichnisversionierung, siehe „Arbeitskopien mit gemischten Revisionen“.

Mehr Operationen ohne Verbindung

Während der letzten Jahre ist Plattenplatz saubillig und im Überfluss verfügbar geworden, die Bandbreite des Netzes jedoch nicht. Deshalb wurde die Arbeitskopie von Subversion hinsichtlich der knapperen Ressource optimiert.

Das Verwaltungsverzeichnis `.svn` dient demselben Zweck wie das Verzeichnis `CVS`, außer dass es zusätzlich schreibgeschützte „unveränderte“ Kopien Ihrer Dateien speichert. Das erlaubt es Ihnen, viele Dinge ohne Verbindung zu erledigen:

svn status
: Zeigt Ihnen alle lokalen Änderungen, die Sie vorgenommen haben (siehe „Verschaffen Sie sich einen Überblick über Ihre Änderungen“)

svn diff
: Zeigt Ihnen die Details Ihrer Änderungen (siehe „Untersuchen Sie die Details Ihrer lokalen Änderungen“)

svn revert
: Macht Ihre lokalen Änderungen rückgängig (siehe „Beheben Sie Ihre Fehler“)

Des weiteren erlauben die zwischengespeicherten unveränderten Dateien dem Subversion-Client bei der Übergabe Unterschiede zu senden, was CVS nicht kann.

Der letzte Unterbefehl in der Liste – **svn revert** – ist neu. Er entfernt nicht nur lokale Änderungen, sondern beseitigt auch vorgemerkte Operationen wie Hinzufügungen und Löschungen. Auch wenn das Löschen einer Datei und der folgende Aufruf von **`svn update`** immer noch funktioniert, verzerrt dies den wahren Zweck einer Aktualisierung. Und, wo wir gerade beim Thema sind…

Unterscheidung zwischen Status und Update

Subversion versucht, einen großen Teil der Verwirrung zu beseitigen, die hinsichtlich der Befehle **cvs status** und **cvs update** besteht.

Der Befehl **cvs status** erfüllt zwei Zwecke: dem Benutzer alle lokalen Änderungen in der Arbeitskopie zu zeigen und welche Dateien nicht mehr aktuell sind. Bedingt durch die schwer lesbare Ausgabe des CVS-Befehls status, verzichten viele CVS-Benutzer ganz auf die Vorteile dieses Befehls. Stattdessen haben sie sich angewöhnt, **`cvs update`** oder **`cvs -n update`** aufzurufen, um schnell ihre Änderungen zu sehen. Falls Benutzer die Option `-n` vergessen, hat das den Nebeneffekt, dass Projektarchiv-Änderungen hineingebracht werden könnten, um die sie sich momentan noch nicht kümmern können.

Subversion räumt mit diesem Durcheinander auf, indem es die Ausgabe von **svn status** leicht lesbar für Menschen und Parser macht. Außerdem gibt **svn update** nur Informationen über Dateien aus, die aktualisiert werden, jedoch *nicht* über lokale Änderungen.

Status

svn status gibt alle Dateien aus, an denen lokale Änderungen vorgenommen wurden. Standardmäßig wird kein Kontakt zum Projektarchiv hergestellt. Obwohl dieser Unterbefehl eine stattliche Anzahl an Optionen versteht, sind die folgenden die meistbenutzten:

`-u`
: Verbindung zum Projektarchiv aufnehmen, um Informationen über nicht mehr Aktuelles zu ermitteln und dann anzuzeigen.

`-v`
: *Alle* versionskontrollierten Einträge anzeigen.

`-N`
: Nicht rekursiv ausführen (nicht in Unterverzeichnisse gehen).

Der Befehl **svn status** besitzt zwei Ausgabeformate. Im standardmäßigen „Kurzformat“ sehen lokale Änderungen so aus:

```
$ svn status
M       foo.c
M       bar/baz.c
```

Falls Sie die Option `--show-updates` (`-u`) angeben, wird ein umfangreicheres Ausgabeformat verwendet:

```
$ svn status -u
M             1047   foo.c
        *     1045   faces.html
        *            bloo.png
M             1050   bar/baz.c

Status bezogen auf Revision:   1066
```

In diesem Fall tauchen zwei neue Spalten auf. Die zweite Spalte beinhaltet ein Sternchen, falls die Datei oder das Verzeichnis nicht mehr aktuell ist. Die dritte Spalte zeigt die Revisionsnummer des Objektes in der Arbeitskopie an. Im vorangegangenen Beispiel zeigt das Sternchen an, dass das Objekt `faces.html` beim Aktualisieren verändert würde, und dass `bloo.png` eine dem Projektarchiv hinzugefügte neue Datei ist. (Die Abwesenheit einer Revisionsnummer neben `bloo.png` bedeutet, dass die Datei noch nicht in der Arbeitskopie vorhanden ist.)

Für eine tiefer gehende Erörterung von **svn status**, darunter eine Erklärung der Status-Codes aus dem obigen Beispiel, siehe „Verschaffen Sie sich einen Überblick über Ihre Änderungen“.

Update

svn update aktualisiert Ihre Arbeitskopie und gibt nur Informationen über Dateien aus, die aktualisiert werden.

Subversion hat den Code `P` und `U` von CVS zu `U` vereinfacht. Wenn eine Zusammenführung stattfindet oder ein Konflikt auftritt, gibt Subversion einfach `G` oder `C` aus, statt einen ganzen Satz darüber zu verlieren.

Für eine detailliertere Erörterung von **svn update**, siehe „Aktualisieren Sie Ihre Arbeitskopie“.

Zweige und Tags

Subversion unterscheidet nicht zwischen Dateisystem- und „Zweig“-Raum; Zweige und Tags sind gewöhnliche Verzeichnisse im Dateisystem. Das ist wahrscheinlich die einzige, größte mentale Hürde, die ein CVS-Benutzer überwinden muss. Lesen Sie

alles hierüber in Kapitel 4, *Verzweigen und Zusammenführen*.

Da Subversion Zweige und Tags wie normale Verzeichnisse behandelt, befinden sich die verschiedenen Entwicklungsstränge Ihres Projektes wahrscheinlich in Unterverzeichnissen des Projekthauptverzeichnisses. Denken Sie also daran, beim Auschecken den URL des Unterverzeichnisses anzugeben, das den bestimmten Entwicklungsstrang enthält, den Sie benötigen, nicht den URL des Projektwurzelverzeichnisses. Falls Sie den Fehler begehen, das Projektwurzelverzeichnis auszuchecken, kann es passieren, dass dabei eine Arbeitskopie entsteht, die eine vollständige Kopie Ihres Projektinhaltes von allen Zweigen und Tags enthält. [1]

Eigenschafts-Metadaten

Ein neues Merkmal von Subversion ist, dass nun beliebige Metadaten (oder „Eigenschaften“) an Dateien und Verzeichnisse geheftet werden können. Eigenschaften sind beliebige Name-Wert-Paare, die mit Dateien und Verzeichnissen in Ihrer Arbeitskopie verbunden sind.

Verwenden Sie zum Setzen oder Abfragen eines Eigenschafts-Namens die Unterbefehle **svn propset** und **svn propget**. Um eine Liste aller Eigenschaften eines Objektes zu erhalten, rufen Sie **svn proplist** auf.

Für weitere Informationen siehe „Eigenschaften“.

Konfliktauflösung

CVS markiert Konflikte mit „Konfliktmarkierungen“ im Text und gibt anschließend während einer Aktualisierung oder einer Zusammenführung ein `C` aus. Historisch hat das zu Problemen geführt, da CVS nicht genug macht. Viele Benutzer vergessen das `C` nachdem es über ihr Terminal geschossen ist (oder sehen es nicht). Oftmals vergessen sie, dass überhaupt Konfliktmarkierungen vorhanden sind und übergeben versehentlich Dateien, die diese Konfliktmarkierungen enthalten.

Subversion löst dieses Problem mit einem Maßnahmenpaar. Zunächst merkt sich Subversion beim Auftreten eines Konfliktes in einer Datei den Konfliktzustand und erlaubt Ihnen die Übergabe der Datei erst dann, wenn Sie explizit die Konflikte aufgelöst haben. Des weiteren bietet Subversion eine interaktive Konfliktauflösung an, so dass Sie Konflikte lösen können sobald sie auftreten, anstatt später nach Abschluss der Aktualisierung oder Übergabe zurückgehen zu müssen. Mehr zur Konfliktauflösung in Subversion lesen Sie in „Lösen Sie etwaige Konflikte auf“.

Binärdateien und Umwandlung

Im Großen und Ganzen geht Subversion mit Binärdateien eleganter um als CVS. Da CVS RCS verwendet, kann es nur aufeinanderfolgende vollständige Kopien einer sich ändernden Binärdatei abspeichern. Subversion jedoch stellt Unterschiede zwischen Dateien mithilfe eines binären Differenzalgorithmus dar. Das bedeutet, dass alle Dateien als (komprimierte) Differenzen im Projektarchiv abgespeichert werden.

Benutzer von CVS müssen binäre Dateien mit der Option `-kb` kennzeichnen, um zu verhindern, dass die Daten verfälscht werden (aufgrund von Schlüsselwortersetzung und der Umwandlung von Zeilenenden). Manchmal vergessen sie es.

Subversion schlägt den paranoideren Weg ein. Erstens macht es keinerlei Schlüsselwortersetzung oder Zeilenendenumwandlung, es sei denn, Sie fordern es ausdrücklich dazu auf (Einzelheiten unter „Ersetzung von Schlüsselworten“ und „Zeichenfolgen zur Zeilenende-Kennzeichnung“). Standardmäßig behandelt Subversion alle Dateiinhalte als Byteketten, und Dateien werden stets ohne Umwandlung im Projektarchiv gespeichert.

Zweitens besitzt Subversion eine interne Auffassung, ob eine Datei „textuellen“ oder „binären“ Inhalt hat, doch besteht diese Auffassung *nur* in der Arbeitskopie. Während eines **svn update** unternimmt Subversion für lokal veränderte Textdateien eine kontextabhängige Zusammenführung, versucht das allerdings nicht bei Binärdateien.

Um festzustellen, ob eine kontextabhängige Zusammenführung möglich ist, überprüft Subversion die Eigenschaft `svn:mime-type`. Falls die Datei keine Eigenschaft `svn:mime-type` besitzt, oder ein textueller MIME-Typ ist (z.B. `text/*`), nimmt Subversion an, dass es sich um eine Textdatei handelt. Anderenfalls nimmt Subversion an, dass die Datei binär ist. Subversion hilft Benutzern auch, indem es einen Algorithmus zur Erkennung von Binärdaten bei den Befehlen **svn import** und **svn add** ausführt. Diese Befehle werden eine gute Schätzung machen und (möglicherweise) eine binäre

[1] Dass heißt, wenn Ihnen vor Abschluss des Checkouts nicht der Plattenplatz zu Ende geht.

Eigenschaft `svn:mime-type` auf die hinzuzufügende Datei setzen. (Falls sich Subversion verschätzt, kann der Benutzer stets die Eigenschaft entfernen oder manuell bearbeiten.)

Versionierte Module

Anders als bei CVS weiß eine Arbeitskopie von Subversion, dass hier ein Modul ausgecheckt ist. Das heißt, dass, falls jemand die Definition eines Moduls ändert (z.B. Komponenten hinzufügt oder entfernt), ein Aufruf von **svn update** die Arbeitskopie entsprechend aktualisiert, indem Komponenten hinzugefügt oder entfernt werden.

Subversion definiert Module als eine Liste von Verzeichnissen innerhalb einer Verzeichnis-Eigenschaft; siehe „Externals-Definitionen“.

Authentifizierung

Bei Verwendung des pservers von CVS müssen Sie sich beim Server anmelden (mit dem Befehl **cvs login**), bevor Sie irgendeine Lese- oder Schreiboperation vornehmen – manchmal müssen Sie sich sogar für anonyme Vorgänge anmelden. Mit einem Subversion-Projektarchiv, das Apache **httpd** oder **svnserve** als Server verwendet, übergeben Sie Zugangsdaten nicht von vornherein – falls eine von Ihnen durchgeführte Tätigkeit eine Authentifizierung notwendig macht, fordert der Server Ihre Zugangsdaten an (egal, ob es sich um Anwendernamen und Passwort, ein Client-Zertifikat oder beides handelt). Falls Ihr Projektarchiv also der ganzen Welt Lesezugriff einräumt, brauchen Sie sich für Lesevorgänge überhaupt nicht zu legitimieren.

Wie bei CVS speichert Subversion Ihre Zugangsdaten immer noch auf Platte (in Ihrem Verzeichnis `~/.subversion/auth/`), es sei denn, Sie untersagen es ihm mit der Option `--no-auth-cache`.

Eine Ausnahme für dieses Verhalten gibt es jedoch beim Zugriff auf einen **svnserve**-Server über einen SSH-Tunnel bei Verwendung des URL-Schemas `svn+ssh://`. In diesem Fall verlangt das Programm **ssh** unbedingt eine Authentifizierung, allein um den Tunnel zu starten.

Ein Projektarchiv von CVS nach Subversion überführen

Die vielleicht wichtigste Methode, CVS-Benutzer mit Subversion vertraut zu machen, ist es, sie ihre Arbeit an den Projekten mit dem neuen System fortführen zu lassen. Obwohl sich das mit einem flachen Import eines exportierten CVS-Projektarchivs in ein Subversion-Projektarchiv bewerkstelligen lässt, umfasst die gründlichere Lösung nicht nur die Übertragung einer Momentaufnahme des aktuellen Datenbestands, sondern der gesamten Historie, die sich dahinter verbirgt. Dabei gilt es, ein äußerst schwieriges Problem zu lösen: neben anderen Komplikationen bedingt dies, dass Änderungsmengen aufgrund der fehlenden Atomizität hergeleitet werden und die vollständig orthogonalen Verzweigungsstrategien übertragen werden müssen. Trotzdem behaupten eine handvoll Werkzeuge, dass sie zumindest teilweise die Fähigkeit mitbringen, bestehende CVS- in Subversion-Projektarchivs umwandeln zu können.

Das meistverbreitete (und ausgereifteste) Werkzeug zur Umwandlung ist cvs2svn (http://cvs2svn.tigris.org/), ein Python-Programm, das ursprünglich von Mitgliedern der Subversion-Entwicklergemeinde erstellt wurde. Dieses Werkzeug soll genau einmal aufgerufen werden: es durchsucht mehrfach Ihr CVS-Projektarchiv und versucht so gut es eben geht, Übergaben, Zweige und Tags herzuleiten. Wenn es fertig ist, liegt als Ergebnis entweder ein Subversion-Projektarchiv oder eine portable Subversion-Auszugsdatei vor, die die Historie Ihres Codes repräsentiert. Auf der Web-Präsenz finden Sie detaillierte Anleitungen und Warnungen.

Anhang C. WebDAV und Autoversionierung

WebDAV ist eine Erweiterung zum HTTP, und es wird als Standard für Dateifreigaben immer beliebter. Heutige Betriebssysteme unterstützen das Web immer besser, und viele haben nun eingebaute Unterstützung zum Einhängen von „Dateifreigaben“, die von WebDAV-Servern exportiert werden.

Falls Sie Apache als Ihren Subversion-Netz-Server verwenden, betreiben Sie auch bis zu einem gewissen Grad einen WebDAV-Server. Dieser Anhang erklärt etwas zum Hintergrund dieses Protokolls, wie es Subversion benutzt und wie Subversion mit Software zusammenarbeitet, die WebDAV unterstützt.

Was ist WebDAV?

DAV steht für „Distributed Authoring and Versioning“ (Verteilte Erstellung und Versionierung). RFC 2518 definiert eine Reihe Konzepte und begleitende Erweiterungsmethoden für HTTP 1.1, die aus dem Web ein universelles Lese- und Schreibmedium machen. Die grundlegende Idee ist, dass ein WebDAV-konformer Web-Server sich wie ein gewöhnlicher Datei-Server verhalten kann; Clients können freigegebene Ordner über HTTP „einhängen“, die sich völlig wie andere vernetzte Dateisysteme verhalten (wie etwa NFS oder SMB).

Das Trauerspiel jedoch ist, dass trotz des Akronyms die RFC-Spezifikation keinerlei Versionskontrolle beschreibt. Einfache WebDAV-Clients und Server gehen davon aus, dass lediglich eine Version jeder Datei oder jedes Verzeichnisses existiert und sie wiederholt überschrieben werden kann.

Da RFC 2518 Versionierungskonzepte ausließ, war einige Jahre später ein anderes Komitee dafür verantwortlich, RFC 3253 zu schreiben. Der neue RFC fügt WebDAV Versionierungskonzepte hinzu und packt das „V“ wieder zurück in „DAV“ – daher der Begriff „DeltaV“. WebDAV/DeltaV-Clients und -Server werden oft nur „DeltaV“-Programme genannt, da DeltaV das Vorhandensein des grundlegenden WebDAV voraussetzt.

Der ursprüngliche WebDAV-Standard war auf breiter Linie erfolgreich. Jedes moderne Betriebssystem verfügt über einen eingebauten WebDAV-Client (Einzelheiten später), und eine Anzahl verbreiteter eigenständiger Anwendungen sprechen ebenfalls WebDAV – Microsoft Office, Dreamweaver und Photoshop, um nur ein paar aufzuzählen. Serverseitig war der Apache HTTP Server in der Lage, seit 1998 WebDAV-Dienste anzubieten und wird als der quelloffene de facto Standard angesehen. Mehrere andere kommerzielle WebDAV-Server sind erhältlich, u.a. IIS von Microsoft.

Leider war DeltaV nicht so erfolgreich. Es ist sehr schwierig, irgendwelche DeltaV-Clients oder -Server zu finden. Die wenigen vorhandenen sind relativ unbekannte kommerzielle Produkte, und somit ist es sehr schwierig, die Interoperabilität zu testen. Es ist noch nicht völlig geklärt, warum DeltaV so stockend voran kam. Manche sind der Meinung, dass die Spezifikation zu komplex sei. Andere wiederum sagen, dass, während sich die Möglichkeiten von WebDAV an die Masse wenden (selbst minder technikaffine Benutzer wissen vernetzte Dateisysteme zu schätzen), seine Versionskontrollfähigkeiten dagegen für die meisten Benutzer nicht interessant oder notwendig seien. Schließlich glauben manche, dass DeltaV unbeliebt bleibe, weil es noch kein quelloffenes Server-Produkt mit einer guten Implementierung gibt.

Als sich Subversion noch in der Entwurfsphase befand, schien es eine großartige Idee zu sein, Apache als einen Netz-Server zu verwenden. Es gab bereits ein Modul, das WebDAV-Dienste anbot. DeltaV war eine relativ neue Spezifikation. Es bestand die Hoffnung, dass sich das Server-Modul von Subversion (**mod_dav_svn**) schließlich zu einer quelloffenen Referenzimplementierung von DeltaV entwickeln würde. Unglücklicherweise besitzt DeltaV ein sehr spezifisches Versionierungsmodell, das sich nicht ganz mit dem Modell von Subversion deckt. Einige Konzepte ließen sich abbilden, andere nicht.

Was bedeutet das nun?

Zunächst ist der Subversion-Client kein vollständig implementierter DeltaV-Client. Er benötigt bestimmte Dinge vom Server, die DeltaV von sich aus nicht bieten kann, und ist deshalb zu einem großen Teil abhängig von einer Anzahl Subversion-spezifischer HTTP `REPORT` Anfragen, die nur **mod_dav_svn** versteht.

Zweitens ist **mod_dav_svn** kein vollständig umgesetzter DeltaV-Server. Viele Teile der DeltaV-Spezifikation waren für Subversion irrelevant und wurden nicht implementiert.

Eine lang anhaltende Debatte innerhalb der Entwicklergemeinde von Subversion, ob es den Aufwand lohne, irgendeine dieser

Situationen zu beheben, wurde letztendlich beigelegt, indem die Subversion-Entwickler offiziell beschlossen, die Pläne zur vollständigen Unterstützung von DeltaV aufzugeben. Neuere Versionen von Subversion werden selbstverständlich weiterhin die bereits in älteren Releases vorhandene DeltaV-Funktionalität unterstützen, jedoch wird keine Arbeit aufgewendet, um die Spezifikation weiter abzudecken; tatsächlich würde Subversion absichtlich beginnen, sich vom strengen DeltaV als sein primäres HTTP-basiertes Protokoll zu entfernen.

Autoversionierung

Obwohl der Subversion-Client kein vollständiger DeltaV-Client und der Subversion-Server kein vollständiger DeltaV-Server ist, gibt es trotzdem noch ein Schimmer an WebDAV-Interoperabilität, über den man sich freuen kann: *Autoversionierung*.

Autoversionierung ist ein optionales Merkmal, das im DeltaV-Standard definiert wird. Ein typischer DeltaV-Server wird einen unwissenden WebDAV-Client ablehnen, der ein `PUT` auf eine Datei unter Versionskontrolle anwenden möchte. Um eine versionskontrollierte Datei zu ändern, erwartet der Server eine Reihe exakter Versionierungsanfragen, so etwas wie `MKACTIVITY`, `CHECKOUT`, `PUT`, `CHECKIN`. Unterstützt der DeltaV-Server jedoch Autoversionierung, werden Schreibanforderungen von einfachen WebDAV-Clients angenommen. Der Server verhält sich, als *habe* der Client die entsprechenden Versionierungsbefehle erteilt und vollzieht die Übergabe im Verborgenen. Mit anderen Worten: es wird einem DeltaV-Server erlaubt, mit gewöhnlichen WebDAV-Clients zusammenzuarbeiten, die keine Versionierung verstehen.

Da so viele Betriebssysteme bereits über integrierte WebDAV-Clients verfügen, kann der Anwendugsfall für dieses Merkmal unglaublich attraktiv für Administratoren sein, die mit technisch unbedarften Benutzern arbeiten müssen. Stellen Sie sich ein Büro vor, in dem gewöhnliche Benutzer auf Microsoft Windows oder Mac OS arbeiten. Jeder Benutzer „hängt“ das Subversion-Projektarchiv „ein“, das sich darstellt wie ein gewöhnlicher vernetzter Ordner. Sie verwenden den freigegebenen Ordner wie immer: öffnen Dateien, bearbeiten und sichern sie. In der Zwischenzeit versioniert der Server alles. Jeder Administrator (oder sachkundige Benutzer) kann immer noch einen Subversion-Client verwenden, um die Historie zu durchsuchen oder ältere Dateiversionen abzurufen.

Dieses Szenario ist keine Fiktion – es ist Wirklichkeit und es funktioniert. Um Autoversionierung in **mod_dav_svn** zu aktivieren, benutzen Sie die Anweisung `SVNAutoversioning` im Block `<Location>` der Datei `httpd.conf`:

```
<Location /repos>
  DAV svn
  SVNPath /var/svn/repository
  SVNAutoversioning on
</Location>
```

Wenn die Autoversionierung von Subversion aktiv ist, erzeugen Schreibanfragen von WebDAV-Clients automatische Übergaben. Eine allgemeine Protokollnachricht wird erzeugt und an jede Revision gehängt.

Bevor Sie dieses Merkmal aktivieren, sollten Sie verstehen, worauf Sie sich einlassen. WebDAV-Clients neigen dazu, *viele* Schreibanfragen abzusetzen, was zu einer riesigen Menge automatisch übergebener Revisionen führt. Beim Sichern von Daten beispielsweise geben viele Clients ein `PUT` einer 0-Byte-Datei aus (um somit einen Namen zu reservieren), gefolgt von einem weiteren `PUT` mit den eigentlichen Daten. Das einfache Schreiben der Datei zieht zwei einzelne Übergaben nach sich. Bedenken Sie auch, dass viele Anwendungen alle paar Minuten automatische Sicherungen machen, was zu noch mehr Übergaben führt.

Falls Sie ein Post-Commit-Hook-Programm verwenden, das E-Mail verschickt, empfiehlt es sich, die Erstellung der Mail zu unterbinden oder auf bestimmte Abschnitte des Projektarchivs zu beschränken; es hängt davon ab, ob Sie das Hereinprasseln der Mails immer noch als wertvolle Benachrichtigungen erachten oder nicht. Des weiteren kann ein kluges Post-Commit-Hook-Programm zwischen Transaktionen unterscheiden, die durch Autoversionierung erzeugt wurden und solchen, die von einer normalen Subversion Übergabeoperation ausgelöst wurden. Der Trick besteht darin, nach einer Revisions-Eigenschaft namens `svn:autoversioned` zu sehen. Falls es vorliegt, wurde die Übergabe durch einen gewöhnlichen WebDAV-Client ausgelöst..

Ein weiteres Merkmal, das eine sinnvolle Ergänzung zur Autoversionierung von Subversion sein kann, kommt aus dem Modul `mod_mime` von Apache. Falls ein WebDAV-Client dem Projektarchiv eine neue Datei hinzufügt, hat der Benutzer keine Gelegenheit, die Eigenschaft `svn:mime-type` zu setzen. Das könnte dazu führen, dass die Datei als allgemeines Icon angezeigt wird, wenn sie innerhalb eines Ordners über WebDAV-Freigabe betrachtet wird, und nicht mit einer bestimmten Anwendung verknüpft wird. Eine Abhilfe wäre ein Systemadministrator (oder eine andere Person mit Subversion-

Kenntnissen) zum Auschecken einer Arbeitskopie und dem manuellen Setzen der Eigenschaft `svn:mime-type`, wenn es für bestimmte Dateien notwendig ist. Doch möglicherweise würden solche Aufräumarbeiten zu keinem Ende führen. Stattdessen können Sie die Anweisung `ModMimeUsePathInfo` in Ihrem Subversion `<Location>`-Block verwenden:

```
<Location /repos>
  DAV svn
  SVNPath /var/svn/repository
  SVNAutoversioning on

  ModMimeUsePathInfo on

</Location>
```

Diese Anweisung erlaubt es, `mod_mime` zu versuchen, den MIME-Typen von durch Autoversionierung frisch hinzugefügten Dateien herzuleiten. Das Modul schaut auf die Dateinamenserweiterung und möglicherweise ebenfalls auf den Inhalt; wenn die Datei bestimmten Mustern entspricht, wird die Eigenschaft `svn:mime-type` der Datei automatisch gesetzt.

Interoperabilität von Clients

Alle WebDAV-Clients fallen in eine von drei Kategorien – eigenständige Anwendungen, Erweiterungen von Dateisystem-Browsern oder Dateisystem-Erweiterungen. Diese Kategorien definieren grob die Typen der WebDAV-Funktionen, die für Anwender verfügbar sind. Tabelle C.1, „Verbreitete WebDAV-Clients“ zeigt sowohl unsere Kategorisierung als auch eine kurze Beschreibung verbreiteter WebDAV-fähiger Software. Weitere Einzelheiten über diese Software und deren allgemeine Kategorie finden Sie in den folgenden Abschnitten.

Tabelle C.1. Verbreitete WebDAV-Clients

Software	Typ	Windows	Mac	Linux	Beschreibung
Adobe Photoshop	Eigenständige WebDAV-Anwendung	X			Bildbearbeitungs-Software, die das direkte Lesen und Schreiben über WebDAV-URLs erlaubt
cadaver	Eigenständige WebDAV-Anwendung		X	X	Kommandozeilen-WebDAV-Client, der Dateiübertragung, Baum- und Sperroperationen unterstützt
DAV Explorer	Eigenständige WebDAV-Anwendung	X	X	X	Java Werkzeug mit graphischer Benutzerschnittstelle zum Erkunden von WebDAV-Freigaben
Adobe Dreamweaver	Eigenständige WebDAV-Anwendung	X			Software zum Erstellen von Webseiten, die direkt über WebDAV-URLs lesen und schreiben kann
Microsoft Office	Eigenständige WebDAV-Anwendung	X			Büroanwendungspaket mit verschiedenen

Software	Typ	Windows	Mac	Linux	Beschreibung
					Komponenten, die direkt über WebDAV-URLs lesen und schreiben können
Microsoft Web Folders	WebDAV-Erweiterung für Dateisystem-Browser	X			Dateisystem-Browser mir graphischer Benutzerschnittstelle, der Baumoperationen auf WebDAV Freigaben durchführen kann
GNOME Nautilus	WebDAV-Erweiterung für Dateisystem-Browser			X	Dateisystem-Browser mir graphischer Benutzerschnittstelle, der Baumoperationen auf WebDAV Freigaben durchführen kann
KDE Konqueror	WebDAV-Erweiterung für Dateisystem-Browser			X	Dateisystem-Browser mir graphischer Benutzerschnittstelle, der Baumoperationen auf WebDAV Freigaben durchführen kann
Mac OS X	WebDAV-Dateisystemimplementierung		X		Betriebssystem mit eingebauter Unterstützung zum Einhängen von WebDAV-Freigaben
Novell NetDrive	WebDAV-Dateisystemimplementierung	X			Programm zum Zuweisen von Windows Laufwerksbuchstaben auf eingehängte WebDAV-Freigaben
SRT WebDrive	WebDAV-Dateisystemimplementierung	X			Dateiübertragungs-Software, die, neben anderen Dingen, die Zuordnung von Windows Laufwerksbuchstaben auf eingehängte WebDAV-Freigaben zulässt
davfs2	WebDAV-Dateisystemimplementierung			X	Linux Dateisystemtreiber, der das Einhängen einer WebDAV-

Software	Typ	Windows	Mac	Linux	Beschreibung
					Freigabe zulässt

Eigenständige WebDAV-Anwendungen

Eine WebDAV-Anwendung ist ein Programm, das über WebDAV-Protokolle mit einem WebDAV-Server spricht. Wir behandeln einige der am weitesten verbreiteten Programme mit dieser Art von WebDAV-Unterstützung.

Microsoft Office, Dreamweaver, Photoshop

Unter Windows besitzen mehrere wohlbekannte Anwendungen eine integrierte WebDAV-Client Funktionalität, so wie Microsofts Office[1], Adobes Photoshop und Dreamweaver. Sie sind in der Lage, direkt über URLs zu lesen und zu schreiben und neigen dazu, bei der Bearbeitung von Dateien regen Gebrauch von WebDAV-Sperren zu machen.

Beachten Sie, dass viele dieser Programme, obwohl sie auch für Mac OS X verfügbar sind, auf dieser Plattform jedoch WebDAV nicht direkt unterstützen. Tatsächlich erlaubt der Dialog Datei#Öffnen auf Mac OS X überhaupt nicht die Eingabe eines Pfades oder eines URL. Wahrscheinlich wurden die WebDAV-Fähigkeiten auf Macintosh-Versionen dieser Programme absichtlich weggelassen, da OS X bereits eine ausgezeichnete WebDAV-Unterstützung auf Dateisystemebene bietet.

cadaver, DAV Explorer

cadaver ist ein sich auf das Wesentliche beschränkendes Unix-Kommandozeilenwerkzeug zum Durchforsten und Ändern von WebDAV-Freigaben. So wie der Subversion-Client verwendet es die HTTP-Bibliothek neon – das ist nicht überraschend, da sowohl neon als auch cadaver vom selben Autor stammen. cadaver ist freie Software (GPL Lizenz) und über http://www.webdav.org/cadaver/ erhältlich.

Die Verwendung von cadaver ist ähnlich wie die Verwendung eines Kommandozeilen-FTP-Programms und eignet sich deshalb besonders für die einfache WebDAV-Fehlersuche. Zur Not kann es zum Herauf- oder Herunterladen von Dateien verwendet werden, um Eigenschaften zu untersuchen und zum Kopieren, Verschieben, Sperren und Entsperren von Dateien:

```
$ cadaver http://host/repos
dav:/repos/> ls
Listing collection `/repos/': succeeded.
Coll: > foobar                                0  May 10 16:19
      > playwright.el                      2864  May  4 16:18
      > proofbypoem.txt                    1461  May  5 15:09
      > westcoast.jpg                     66737  May  5 15:09

dav:/repos/> put README
Uploading README to `/repos/README':
Progress: [=============================>] 100.0% of 357 bytes succeeded.

dav:/repos/> get proofbypoem.txt
Downloading `/repos/proofbypoem.txt' to proofbypoem.txt:
Progress: [=============================>] 100.0% of 1461 bytes succeeded.
```

DAV Explorer ist ein weiterer selbständiger WebDAV-Client, der in Java geschrieben ist. Er steht unter einer freien Apache-ähnlichen Lizenz und ist erhältlich über http://www.ics.uci.edu/~webdav/. Er macht alles, was auch cadaver macht, hat jedoch den Vorteil, ein portables, benutzerfrundliches Programm mit einer graphischen Benutzeroberfläche zu sein. Er ist auch einer der ersten Clients, die das neue WebDAV-Zugriffskontrollprotokoll (RFC 3744) unterstützen.

Natürlich ist die ACL-Unterstützung von DAV Explorer in diesem Fall nutzlos, da sie nicht von **mod_dav_svn** unterstützt wird. Die Tatsache, dass sowohl cadaver als auch DAV Explorer einige eingeschränkte DeltaV-Befehle unterstützen, ist ebenfalls nicht sehr nützlich, da sie keine `MKACTIVITY`-Anfragen erlauben. Es ist jedenfalls belanglos; wir nehmen an, dass all diese Clients mit einem autoversionierenden Projektarchiv zusammenarbeiten.

[1] WebDAV-Unterstützung wurde aus bestimmten Gründen aus Microsoft Access entfernt, jedoch besteht diese im Rest des Pakets.

WebDAV-Erweiterungen von Dateisystem-Browsern

Einige verbreitete graphische Dateisystem-Browser unterstützen WebDAV-Erweiterungen, die es Benutzern erlauben, eine DAV-Freigabe zu durchstöbern als sei es ein gewöhnliches Verzeichnis auf dem lokalen Rechner und einfache Bearbeitungen der Dateisystemobjekte über diese Freigabe vorzunehmen. So ist beispielsweise der Windows Explorer in der Lage, einen WebDAV-Server als „Netzwerkumgebung“ zu durchstöbern. Benutzer können Dateien auf die oder von der Arbeitsoberfläche ziehen oder Dateien auf althergebrachte Weise umbenennen, kopieren oder löschen. Da es jedoch eine Fähigkeit des Explorers ist, ist die DAV-Freigabe für normale Anwendungen nicht sichtbar. Alle Zugriffe auf DAV müssen über die Schnittstelle des Explorers erfolgen.

Microsoft Webordner

Microsoft war unter den ursprünglichen Unterstützern der WebDAV-Spezifikation und begann, den ersten Client in Windows 98 auszuliefern, der als Webordner bekannt wurde. Dieser Client wurde auch mit Windows NT 4.0 und Windows 2000 ausgeliefert.

Der ursprüngliche Webordner-Client war eine Erweiterung des Explorers, dem graphischen Hauptprogramm zum Durchstöbern von Dateisystemen. Er arbeitet gut genug. In Windows 98 könnte es sein, das dieses Merkmal ausdrücklich installiert werden muss, falls Webordner noch nicht innerhalb von Mein Computer sichtbar sind. In Windows 2000 muss einfach eine neue „Netzwerkumgebung“ hizugefügt und ein URL eingegeben werden, und die WebDAV-Freigabe erscheint zum Durchstöbern.

Mit der Einführung von Windows XP begann Microsoft, eine neue Implementierung der Webordner herauszugeben, die unter dem Namen WebDAV-Mini-Redirector bekannt wurde. Die neue Implementierung ist ein Client auf Dateisystemebene, die es ermöglicht, WebDAV-Freigaben unter einem Laufwerksbuchstaben einzuhängen. Leider ist diese Implementierung unglaublich fehlerhaft. Für gewöhnlich versucht der Client, HTTP-URLs (`http://host/repos`) in UNC-Freigabe-Darstellung (`\\host\repos`) umzuwandeln; er versucht darüber hinaus, Windows-Domänen-Authentifizierung als Antwort auf Basic-Authentication HTTP-Anforderungen zu verwenden, wobei Anwendernamen als `RECHNER\anwendername` gesendet werden. Diese Probleme bei der Zusammenarbeit sind schwerwiegend und an zahlreichen Orten rund um die Welt dokumentiert, was viele Benutzer frustriert. Sogar Greg Stein, der ursprüngliche Autor von Apaches WebDAV-Modul, sagt ohne Umschweife, dass die Webordner von XP nicht mit einem Apache-Server zusammenarbeiten können.

Die erste Implementierung der Webordner in Windows Vista scheint fast die gleiche wie unter XP zu sein, so dass dieselben Probleme auftauchen. Mit etwas Glück wird Microsoft diese Probleme in einem Vista-Service-Pack beheben.

Jedoch scheint es provisorische Lösungen sowohl für XP und Vista zu geben, die es Webordner ermöglichen, mit Apache zusammenzuarbeiten. Benutzer haben meistens berichtet, dass diese Techniken erfolgreich waren, also werden wir sie an dieser Stelle weitergeben.

Unter Windows XP haben Sie zwei Optionen. Suchen Sie zunächst in der Webpräsenz von Microsoft nach dem Update KB907306, „Software Update for Web Folders“. Das könnte all Ihre Probleme lösen. Falls nicht, scheinen sich die originalen Webordner aus der Zeit vor XP in Ihrem System zu verbergen. Sie können sie ans Licht holen, indem Sie in Netzwerkumgebung gehen und eine neue Umgebung anlegen. Wenn Sie danach gefragt werden, tragen Sie den URL des Projektarchivs ein, aber *verwenden Sie dabei eine Portnummer* im URL. Beispielsweise sollten Sie statt **`http://host/repos`** **`http://host:80/repos`** eingeben. Antworten Sie auf irgendwelche Authentifizierungsanfragen mit Ihren Subversion-Zugangsdaten.

Unter Windows Vista könnte derselbe Update KB907306 alles bereinigen. Dort könnte es jedoch noch weitere Probleme geben. Einige Benutzer haben berichtet, dass Vista alle `http://`-Verbindungen als unsicher betrachtet, und deshalb alle Authentifizierungsanforderungen seitens Apache scheitern lässt, sofern sie nicht über `https://` erfolgen. Sollten Sie nicht in der Lage sein, sich über SSL mit dem Subversion-Projektarchiv zu verbinden, können Sie die System-Registratur anpassen, um dieses Verhalten zu unterbinden. Dazu ändern Sie einfach den Wert des Schlüssels `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WebClient\Parameters\BasicAuthLevel` von **`1`** auf **`2`**. Eine letzte Warnung: Stellen Sie sicher, den Webordner so einzurichten, dass er auf das Wurzelverzeichnis des Projektarchivs (`/`) zeigt, statt auf ein Unterverzeichnis wie etwa `/trunk`. Die Webordner von Vista scheinen nur mit Projektarchiv-Wurzelverzeichnissen zu funktionieren.

Generell sei gesagt, dass Sie, obwohl diese Zwischenlösungen für Sie funktionieren könnten, mehr Spaß bei der Verwendung eines WebDAV-Clients von einem Drittanbieter haben werden, wie etwa WebDrive oder NetDrive.

Nautilus, Konqueror

Nautilus ist der offizielle Dateimanager/Browser des GNOME-Desktops (http://www.gnome.org), und Konqueror ist der Manager/Browser für den KDE-Desktop (http://www.kde.org). Beide dieser Anwendungen besitzen einen eingebauten WebDAV-Client auf Dateisystem-Browser-Ebene und funktionieren prima mit einem autoversionierenden Projektarchiv.

In GNOMEs Nautilus wählen Sie den Menüeintrag File#Open location und tragen den URL in den angebotenen Dialog ein. Das Projektarchiv sollte dann wie ein anderes Dateisystem angezeigt werden.

Im Konqueror von KDE müssen Sie beim Eintragen des URL in den Zielleiste das Schema `webdav://` verwenden. Wenn Sie einen `http://`-URL eingeben, verhält sich Konqueror wie ein gewöhnlicher Web-Browser. Sie werden wahrscheinlich die allgemeine Verzeichnisauflistung von **mod_dav_svn** sehen. Falls Sie statt **`http://host/repos`** jedoch **`webdav://host/repos`** eingeben, wird Konqueror zum WebDAV-Client und zeigt das Projektarchiv als Dateisystem an.

WebDAV-Dateisystem-Implementation

Die WebDAV-Dateisystem-Implementation ist wohl die beste Art von WebDAV-Client. Sie ist als systemnahes Dateisystem-Modul implementiert, das sich normalerweise innerhalb des Betriebssystemkerns befindet. Das bedeutet, dass eine DAV-Freigabe wie irgendein anderes Dateisystem eingehängt werden kann, ähnlich dem Einhängen einer NFS-Freigabe unter Unix oder dem Sichtbarmachen einer SMB-Freigabe über einen Laufwerksbuchstaben unter Windows. Demzufolge bietet diese Art von Client für alle Programme einen vollständig transparenten WebDAV-Schreib- und Lesezugriff. Die Anwendungen merken nicht einmal, dass WebDAV-Anforderungen stattfinden.

WebDrive, NetDrive

Sowohl WebDrive als auch NetDrive sind ausgezeichnete kommerzielle Produkte, die es ermöglichen, WebDAV-Freigaben unter Windows als Laufwerksbuchstaben einzuhängen. Als Ergebnis können Sie mit den Inhalten dieser Pseudo-Laufwerke mit WebDAV im Hintergrund ebenso einfach und auf dieselbe Weise arbeiten wie mit lokalen Festplatten. Sie können WebDrive von South River Technologies (http://www.southrivertech.com) beziehen. NetDrive von Novell ist frei online verfügbar, die Benutzung erfordert jedoch eine NetWare Lizenz.

Mac OS X

Apples Betriebssystem OS X besitzt einen integrierten WebDAV-Client auf Dateisystemebene. Wählen Sie im Finder den Menüeintrag Go#Connect to Server aus. Geben Sie einen WebDAV-URL ein, und er erscheint als Laufwerk auf der Arbeitsoberfläche so wie jede andere eingehängte Platte. Sie können auch eine WebDAV-Freigabe aus dem Darwin-Terminal einhängen, indem Sie den Dateisystemtypen `webdav` beim Befehl **mount** angeben:

```
$ mount -t webdav http://svn.example.com/repos/project /some/mountpoint
$
```

Beachten Sie, dass OS X sich weigert, die Freigabe les- und schreibbar einzuhängen, sofern die Version Ihres **mod_dav_svn** älter ist als 1.2; in diesem Fall wird sie nur lesbar sein. Das liegt daran, das OS X auf der Unterstützung von Sperren bei les- und schreibbaren Freigaben besteht, und die Fähigkeit, Dateien zu sperren erst mit Subversion 1.2 erschien.

Des weiteren kann der WebDAV-Client von OS X manchmal überempfindlich auf HTTP-Weiterleitungen reagieren. Falls OS X das Projektarchiv überhaupt nicht einhängen kann, könnte es notwendig sein, die Direktive `BrowserMatch` in der Datei `httpd.conf` des Apache Servers zu aktivieren:

```
BrowserMatch "^WebDAVFS/1.[012]" redirect-carefully
```

Linux davfs2

Linux davfs2 ist ein Dateisystem-Modul für den Linux-Kernel, dessen Entwicklung bei http://dav.sourceforge.net/ betrieben wird. Sobald Sie davfs2 installiert haben, können Sie eine WebDAV-Netzfreigabe mit dem üblichen Linux mount-Befehl einhängen:

```
$ mount.davfs http://host/repos /mnt/dav
```

Anhang D. Copyright

b. Unter den „Lizenzelementen“ werden die folgenden Lizenzcharakteristika verstanden, die vom Lizenzgeber ausgewählt und in der Bezeichnung der Lizenz genannt werden: „Namensnennung“, „Nicht-kommerziell“, „Weitergabe unter gleichen Bedingungen“.

c. Unter dem „Lizenzgeber“ wird die natürliche oder juristische Person verstanden, die den Schutzgegenstand unter den Bedingungen dieser Lizenz anbietet.

d. Unter einem „Sammelwerk“ wird eine Sammlung von Werken, Daten oder anderen unabhängigen Elementen verstanden, die aufgrund der Auswahl oder Anordnung der Elemente eine persönliche geistige Schöpfung ist. Darunter fallen auch solche Sammelwerke, deren Elemente systematisch oder methodisch angeordnet und einzeln mit Hilfe elektronischer Mittel oder auf andere Weise zugänglich sind (Datenbankwerke). Ein Sammelwerk wird im Zusammenhang mit dieser Lizenz nicht als Bearbeitung (wie oben beschrieben) angesehen.

e. Mit „SIE“ und „Ihnen“ ist die natürliche oder juristische Person gemeint, die die durch diese Lizenz gewährten Nutzungsrechte ausübt und die zuvor die Bedingungen dieser Lizenz im Hinblick auf das Werk nicht verletzt hat, oder die die ausdrückliche Erlaubnis des Lizenzgebers erhalten hat, die durch diese Lizenz gewährten Nutzungsrechte trotz einer vorherigen Verletzung auszuüben.

f. Unter dem „Schutzgegenstand“wird das Werk oder Sammelwerk oder das Schutzobjekt eines verwandten Schutzrechts, das Ihnen unter den Bedingungen dieser Lizenz angeboten wird, verstanden

g. Unter dem „Urheber“ wird die natürliche Person verstanden, die das Werk geschaffen hat.

h. Unter einem „verwandten Schutzrecht“ wird das Recht an einem anderen urheberrechtlichen Schutzgegenstand als einem Werk verstanden, zum Beispiel einer wissenschaftlichen Ausgabe, einem nachgelassenen Werk, einem Lichtbild, einer Datenbank, einem Tonträger, einer Funksendung, einem Laufbild oder einer Darbietung eines ausübenden Künstlers.

i. Unter dem „Werk“ wird eine persönliche geistige Schöpfung verstanden, die Ihnen unter den Bedingungen dieser Lizenz angeboten wird.

2. Schranken des Urheberrechts. Diese Lizenz lässt sämtliche Befugnisse unberührt, die sich aus den Schranken des Urheberrechts,aus dem Erschöpfungsgrundsatz oder anderen Beschränkungen der Ausschließlichkeitsrechte des Rechtsinhabers ergeben.

3. Lizenzierung. Unter den Bedingungen dieses Lizenzvertrages räumt Ihnen der Lizenzgeber ein lizenzgebührenfreies, räumlich und zeitlich (für die Dauer des Urheberrechts oder verwandten Schutzrechts) unbeschränktes einfaches Nutzungsrecht ein, den Schutzgegenstand in der folgenden Art und Weise zu nutzen:

a. den Schutzgegenstand in körperlicher Form zu verwerten, insbesondere zu vervielfältigen, zu verbreiten und auszustellen;

b. den Schutzgegenstand in unkörperlicher Form öffentlich wiederzugeben, insbesondere vorzutragen, aufzuführen und vorzuführen, öffentlich zugänglich zu machen, zu senden, durch Bild- und Tonträger wiederzugeben sowie Funksendungen und öffentliche Zugänglichmachungen wiederzugeben;

c. den Schutzgegenstand auf Bild- oder Tonträger aufzunehmen, Lichtbilder davon herzustellen, weiterzusenden und in dem in a. und b. genannten Umfang zu verwerten;

d. den Schutzgegenstand zu bearbeiten oder in anderer Weise umzugestalten und die Bearbeitungen zu veröffentlichen und in dem in a. bis c. genannten Umfang zu verwerten;

Die genannten Nutzungsrechte können für alle bekannten Nutzungsarten ausgeübt werden. Die genannten Nutzungsrechte beinhalten das Recht, solche Veränderungen an dem Werk vorzunehmen, die technisch erforderlich sind, um die Nutzungsrechte für alle Nutzungsarten wahrzunehmen. Insbesondere sind davon die Anpassung an andere Medien und auf andere Dateiformate umfasst.

4. Beschränkungen. Die Einräumung der Nutzungsrechte gemäß Ziffer 3 erfolgt ausdrücklich nur unter den folgenden Bedingungen:

   a. Sie dürfen den Schutzgegenstand ausschließlich unter den Bedingungen dieser Lizenz vervielfältigen, verbreiten oder öffentlich wiedergeben, und Sie müssen stets eine Kopie oder die vollständige Internetadresse in Form des Uniform-Resource-Identifier (URI) dieser Lizenz beifügen, wenn Sie den Schutzgegenstandvervielfältigen, verbreiten oder öffentlich wiedergeben. Sie dürfen keine Vertragsbedingungen anbieten oder fordern, die die Bedingungen dieser Lizenz oder die durch sie gewährten Rechte ändern oder beschränken. Sie dürfen den Schutzgegenstand nicht unterlizenzieren. Sie müssen alle Hinweise unverändert lassen, die auf diese Lizenz und den Haftungsausschluss hinweisen. Sie dürfen den Schutzgegenstand mit keinen technischen Schutzmaßnahmen versehen, die den Zugang oder den Gebrauch des Schutzgegenstandes in einer Weise kontrollieren, die mit den Bedingungen dieser Lizenz im Widerspruch stehen. Die genannten Beschränkungen gelten auch für den Fall, dass der Schutzgegenstand einen Bestandteil eines Sammelwerkes bildet; sie verlangen aber nicht, dass das Sammelwerk insgesamt zum Gegenstand dieser Lizenz gemacht wird. Wenn Sie ein Sammelwerk erstellen, müssen Sie - soweit dies praktikabel ist - auf die Mitteilung eines Lizenzgebers oder Urhebers hin aus dem Sammelwerk jeglichen Hinweis auf diesen Lizenzgeber oder diesen Urheber entfernen. Wenn Sie den Schutzgegenstand bearbeiten, müssen Sie - soweit dies praktikabel ist- auf die Aufforderung eines Rechtsinhabers hin von der Bearbeitung jeglichen Hinweis auf diesen Rechtsinhaber entfernen.

   b. Wenn Sie den Schutzgegenstand oder eine Bearbeitung oder ein Sammelwerk vervielfältigen, verbreiten oder öffentlich wiedergeben, müssen Sie alle Urhebervermerke für den Schutzgegenstand unverändert lassen und die Urheberschaft oder Rechtsinhaberschaft in einer der von Ihnen vorgenommenen Nutzung angemessenen Form anerkennen, indem Sie den Namen (oder das Pseudonym, falls ein solches verwendet wird) des Urhebers oder Rechteinhabers nennen, wenn dieser angegeben ist. Dies gilt auch für den Titel des Schutzgegenstandes, wenn dieser angeben ist, sowie - in einem vernünftigerweise durchführbaren Umfang - für die mit dem Schutzgegenstand zu verbindende Internetadresse in Form des Uniform-Resource-Identifier (URI), wie sie der Lizenzgeber angegeben hat, sofern dies geschehen ist, es sei denn, diese Internetadresse verweist nicht auf den Urhebervermerk oder die Lizenzinformationen zu dem Schutzgegenstand. Bei einer Bearbeitung ist ein Hinweis darauf aufzuführen, in welcher Form der Schutzgegenstand in die Bearbeitung eingegangen ist (z.B. „Französische Übersetzung des ... (Werk) durch ... (Urheber)“ oder „Das Drehbuch beruht auf dem Werk des ... (Urheber)“). Ein solcher Hinweis kann in jeder angemessenen Weise erfolgen, wobei jedoch bei einer Bearbeitung, einer Datenbank oder einem Sammelwerk der Hinweis zumindest an gleicher Stelle und in ebenso auffälliger Weise zu erfolgen hat wie vergleichbare Hinweise auf andere Rechtsinhaber.

   c. Obwohl die gemäss Ziffer 3 gewährten Nutzungsrechte in umfassender Weise ausgeübt werden dürfen, findet diese Erlaubnis ihre gesetzliche Grenze in den Persönlichkeitsrechten der Urheber und ausübenden Künstler, deren berechtigte geistige und persönliche Interessen bzw. deren Ansehen oder Ruf nicht dadurch gefährdet werden dürfen, dass ein Schutzgegenstand über das gesetzlich zulässige Maß hinaus beeinträchtigt wird.

5. Gewährleistung. Sofern dies von den Vertragsparteien nicht anderweitig schriftlich vereinbart,, bietet der Lizenzgeber keine Gewährleistung für die erteilten Rechte, außer für den Fall, dass Mängel arglistig verschwiegen wurden. Für Mängel anderer Art, insbesondere bei der mangelhaften Lieferung von Verkörperungen des Schutzgegenstandes, richtet sich die Gewährleistung nach der Regelung, die die Person, die Ihnen den Schutzgegenstand zur Verfügung stellt, mit Ihnen außerhalb dieser Lizenz vereinbart, oder - wenn

eine solche Regelung nicht getroffen wurde - nach den gesetzlichen Vorschriften.

6. Haftung. Über die in Ziffer 5 genannte Gewährleistung hinaus haftet Ihnen der Lizenzgeber nur für Vorsatz und grobe Fahrlässigkeit.

7. Vertragsende

   a. Dieser Lizenzvertrag und die durch ihn eingeräumten Nutzungsrechte enden automatisch bei jeder Verletzung der Vertragsbedingungen durch Sie. Für natürliche und juristische Personen, die von Ihnen eine Bearbeitung, eine Datenbank oder ein Sammelwerk unter diesen Lizenzbedingungen erhalten haben, gilt die Lizenz jedoch weiter, vorausgesetzt, diese natürlichen oder juristischen Personen erfüllen sämtliche Vertragsbedingungen. Die Ziffern 1, 2, 5, 6, 7 und 8 gelten bei einer Vertragsbeendigung fort.

   b. Unter den oben genannten Bedingungen erfolgt die Lizenz auf unbegrenzte Zeit (für die Dauer des Schutzrechts). Dennoch behält sich der Lizenzgeber das Recht vor, den Schutzgegenstand unter anderen Lizenzbedingungen zu nutzen oder die eigene Weitergabe des Schutzgegenstandes jederzeit zu beenden, vorausgesetzt, dass solche Handlungen nicht dem Widerruf dieser Lizenz dienen (oder jeder anderen Lizenzierung, die auf Grundlage dieser Lizenz erfolgt ist oder erfolgen muss) und diese Lizenz wirksam bleibt, bis Sie unter den oben genannten Voraussetzungen endet.

8. Schlussbestimmungen

   a. Jedes Mal, wenn Sie den Schutzgegenstand vervielfältigen, verbreiten oder öffentlich wiedergeben, bietet der Lizenzgeber dem Erwerber eine Lizenz für den Schutzgegenstand unter denselben Vertragsbedingungen an, unter denen er Ihnen die Lizenz eingeräumt hat.

   b. Jedes Mal, wenn Sie eine Bearbeitung vervielfältigen, verbreiten oder öffentlich wiedergeben, bietet der Lizenzgeber dem Erwerber eine Lizenz für den ursprünglichen Schutzgegenstand unter denselben Vertragsbedingungen an, unter denen er Ihnen die Lizenz eingeräumt hat.

   c. Sollte eine Bestimmung dieses Lizenzvertrages unwirksam sein, so wird die Wirksamkeit der übrigen Lizenzbestimmungen dadurch nicht berührt, und an die Stelle der unwirksamen Bestimmung tritt eine Ersatzregelung, die dem mit der unwirksamen Bestimmung angestrebten Zweck am nächsten kommt.

   d. Nichts soll dahingehend ausgelegt werden, dass auf eine Bestimmung dieses Lizenzvertrages verzichtet oder einer Vertragsverletzung zugestimmt wird, so lange ein solcher Verzicht oder eine solche Zustimmung nicht schriftlich vorliegen und von der verzichtenden oder zustimmenden Vertragspartei unterschrieben sind

   e. Dieser Lizenzvertrag stellt die vollständige Vereinbarung zwischen den Vertragsparteien hinsichtlich des Schutzgegenstandes dar. Es gibt keine weiteren ergänzenden Vereinbarungen oder mündlichen Abreden im Hinblick auf den Schutzgegenstand. Der Lizenzgeber ist an keine zusätzlichen Abreden gebunden, die aus irgendeiner Absprache mit Ihnen entstehen könnten. Der Lizenzvertrag kann nicht ohne eine übereinstimmende schriftliche Vereinbarung zwischen dem Lizenzgeber und Ihnen abgeändert werden.

   f. Auf diesen Lizenzvertrag findet das Recht der Bundesrepublik Deutschland Anwendung.

CREATIVE COMMONS IST KEINE VERTRAGSPARTEI DIESES LIZENZVERTRAGES UND ÜBERNIMMT KEINERLEI GEWÄHRLEISTUNG FÜR DAS WERK. CREATIVE COMMONS IST IHNEN ODER DRITTEN GEGENÜBER NICHT HAFTBAR FÜR SCHÄDEN JEDWEDER ART. UNGEACHTET DER VORSTEHENDEN ZWEI (2) SÄTZE HAT CREATIVE COMMONS ALL RECHTE UND PFLICHTEN EINES LIZENSGEBERS, WENN SICH CREATIVE COMMONS AUSDRÜCKLICH ALS LIZENZGEBER BEZEICHNET.

AUSSER FÜR DEN BESCHRÄNKTEN ZWECK EINES HINWEISES AN DIE ÖFFENTLICHKEIT, DASS

Stichwortverzeichnis

T

U

V

Z