Paper

CM Paper

Neues in Subversion 1.6

Tree Conflicts

File-Externals

Und mehr

Neues in Subversion 1.6

Das um die Jahrtausendwende gestartete *Subversion*-Projekt ist längst über den ursprünglichen Anspruch, einfach nur ein besseres *CVS* zu werden, hinausgewachsen. *Subversion* ist zu einem professionellen *Software-Configuration-Management*-Werkzeug gereift, wird von immer mehr großen internationalen Unternehmen eingesetzt und löst zunehmend auch kommerzielle Systeme ab.

Hohe Anforderungen aus dem professionellen Nutzerkreis sorgen dafür, dass die *Open-Source*-Entwickler sich nicht auf dem Stand der mittlerweile sehr stabilen 1.5-er Linie ausruhen können.

Die neuen Features in *Subversion 1.6* sind umfangreich und zum größten Teil darauf ausgerichtet, Probleme zu beheben, die anspruchsvollen Nutzern bei der täglichen Arbeit mit *Subversion 1.5* aufgefallen sind.

Neben einer ganzen Reihe von Verbesserungen [1] ist den folgenden Neuerungen besondere Aufmerksamkeit zu schenken:

Tree Conflicts

Konflikte tauchen auf, wenn widersprüchliche Änderungen verschmolzen werden sollen -- sei es weil Entwickler gleichzeitig arbeiten (`update` und `commit`) oder weil Entwicklungslinien zusammengeführt werden (`merge`).

Subversion unterschied bisher zwischen zwei möglichen Konflikttypen: Den Text-Konflikten und den Property-Konflikten. Letztere sind im Grunde wie Text-Konflikte, die jedoch innerhalb der Metadaten auftauchen statt innerhalb der versionierten Dateien.

Ein weiterer Konflikttyp wurde bislang nur selten erkannt: der strukturelle Konflikt, oder *Tree Conflict*. Dieser taucht auf, wenn widersprüchliche Änderungen an der *Baumstruktur* der Verzeichnisse und Dateien verschmolzen werden sollen.

Ein Beispiel: Entwickler A löscht eine Datei, während Entwickler B derselben Datei einige neue Zeilen hinzufügt. Es ist also nicht möglich, automatisch zu entscheiden, welche der beiden Seiten sich durchsetzen sollte.

Bisher verhielt sich *Subversion* bei strukturellen Konflikten parteiisch, störrisch oder gar vergesslich – ein „Show Stopper" für die Kombination von extensivem *Branching/Merging* mit *Refactoring*. Es war bisher zum Beispiel möglich, dass lokal modifizierte Dateien aus der Versionskontrolle fielen, wenn sie durch einen `update`-Vorgang „gelöscht" wurden. Die lokalen Änderungen wurden unterschlagen.

Subversion 1.6 erkennt diese und ähnliche Situationen nun ausdrücklich als *Tree Conflicts*, und hält Entwickler davon ab, folgenschwere Fehler zu übersehen.

Das Spektrum möglicher *Tree Conflicts* besteht aus [2]:

- *hier* modifiziert, *dort* gelöscht/umbenannt (und umgekehrt)
- *hier* gelöscht/umbenannt, *dort* gelöscht/umbenannt
- *hier* etwas neu erstellt, *dort* mit gleichem Namen etwas neu erstellt

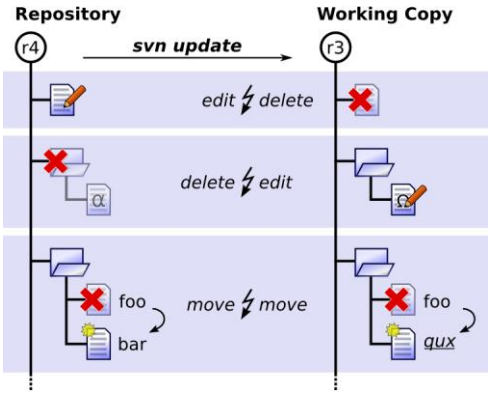

Abb. 1: Einige Beispiele struktureller Konflikte (*Tree Conflicts*)

Auf Dateiebene funktioniert die Erkennung struktureller Konflikte nun zuverlässig. Hier gibt es schlimmstenfalls falsche Positive: da *Subversion* seit jeher eine `move` Operation in je eine `delete` und eine `add` Operation trennt, kann heute ein „echtes" `delete` nicht von einem `move`-bedingten `delete` unterschieden werden -- bei allen überschneidenden `delete` Operationen muss also ein Konflikt angenommen werden. Die Implementierung eines „echten" `move` wird unterdessen diskutiert [3].

Strukturelle Konflikte mit Ordnern können wie bei den Dateien „flach" sein und verhalten sich dann ebenso. Insbesondere können aber auch Änderungen an allen Dateien und Ordnern *innerhalb* eines Ordners einen Konflikt verursachen. Das ist der Fall wenn eine Seite einen Ordner löscht, die andere Seite jedoch eine Datei oder einen Ordner tief im Innern des betreffenden Ordners modifiziert hat. Es muss also Ordner für Ordner und Datei für Datei überprüft werden, ob der 'dort' gelöschte Verzeichnisbaum mit seinem Gegenstück 'hier' identisch ist. Leider ist der interne Vergleichsmechanismus (`diff`) historisch bedingt noch nicht in der Lage, eine Working Copy mit einer beliebigen Repository URL, also z.B. mit einem anderen Branch, zu vergleichen. So müssen einige dieser innerhalb von Ordnern verborgenen strukturellen Konflikte auch in *Subversion 1.6* leider noch unter den Tisch fallen [4].

Wie die altbekannten Konflikttypen muss auch ein einmal aufgetauchter *Tree Conflict* ausdrücklich vom Benutzer aufgelöst werden, bevor ein `commit` zugelassen wird. Es gilt, die unter Umständen sehr komplexe Konfliktsituation sinnvoll zu klären. Abhilfe schaffen hier `svn status` und `svn info`, die genaue Information über die Konflikte ausgeben. `svn resolve`, `remove --force` oder `revert` können dann je nach Situation den Konflikt lösen. Beispiele hierfür gibt es unter [5] und [6].

Man beachte, dass Konflikte ausschließlich in der Working Copy auftauchen und das Repository nicht beeinflussen. Im Notfall können also die nötigen Änderungen in eine neue Working Copy übertragen werden.

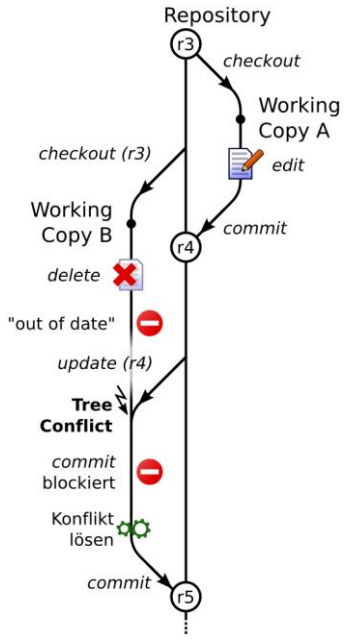

Abb. 2: Zeitlinie: Beispiel eines strukturellen Konflikts beim update

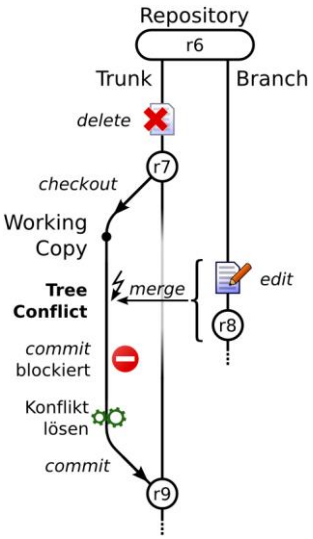

Abb. 3: Zeitlinie: Beispiel eines strukturellen Konflikts beim merge

Zwar sind die Erkennung von *Tree Conflicts* und ihre Auflösungsstrategien noch nicht perfekt, *Subversion 1.6* macht aber in der sinnvollen Behandlung struktureller Konflikte gegenüber seinen Vorgängern einen großen Schritt vorwärts. Der größte Teil möglicher Konfliktsituationen in der Baumstruktur der Ordner und Dateien wird nun erkannt und damit entschärft.

Verschlüsselter Passwort-Cache

Damit ein *Subversion* Passwort nicht bei jedem Serverzugriff erneut eingegeben werden muss, gibt es einen lokalen Passwort-Cache [7]. Seit *Subversion 1.2* kann hierfür die *Windows-CryptoAPI* verwendet werden, und seit *Subversion 1.4* auch die *OS X Keychain*. Auf allen anderen Systemen werden bisher stattdessen die Passwörter in „Plaintext" (unverschlüsselt) in

einer Textdatei abgelegt - ein rotes Tuch für viele Benutzer und Administratoren [8].

Subversion 1.6 weitet die nahtlose Einbindung in Benutzer-Desktops auf alle gängigen Plattformen aus: *GNOME Keyring* und *KDE Wallet* können nun ebenfalls mit *Subversion* integriert werden. Sind sie beim Kompilieren des `svn` Programms mit `--with-gnome-keyring` bzw. `--with-kwallet` angeschaltet worden, werden sie ohne weiteren Aufwand automatisch verwendet: bei Bedarf wird ein Benutzer auf die jeweils übliche Weise aufgefordert, sein Hauptpasswort anzugeben. Sollte es Verwirrung geben, kann mit der ebenfalls neuen Konfigurationsoption `password-stores` ein bestimmter Kryptographiedienst bevorzugt werden [9]. Es ist zudem nicht zwingend notwendig, einen grafischen Desktop zu verwenden: beispielsweise lässt sich der *GNOME-Keyring* auch im Textmodus starten [10].

Abb. 4: *Subversion* fragt nach der *GNOME Passphrase*, wenn der *Keyring* noch gesperrt ist.

Fehlen entsprechende Dienste oder gibt es Probleme mit ihnen, fällt *Subversion* weiterhin auf den alten *Plaintext*-Passwort-Cache zurück. Um aber Versehen zu vermeiden bittet *Subversion 1.6* nun ausdrücklich um Erlaubnis bevor ein Passwort unverschlüsselt gespeichert wird.

Repository-Root-Relative URLs

Aus der Notation von Externals sind seit *Subversion 1.5* einige URL-Kurzformen bekannt [11], allen voran das "`^/`" anstelle der Repository Root URL (die *caret notation* [12]). Dieses Kürzel steht ab *Subversion 1.6* auch auf der Kommandozeile zur Verfügung, und bezieht sich hierbei auf das Repository der gegenwärtigen Working Copy. Beim ersten `checkout` macht es deshalb keinen Sinn, ist aber bei Arbeiten innerhalb einer Working Copy sehr hilfreich.

Hier einige Beispiele:

Kurzform	Bedeutung
`svn info ^/`	Zeigt die neueste eingecheckte Revision im Repository, z.B. *http://beispiel.com/mein/repos/*
`svn list ^/tags`	Listet alle Tags in *http://beispiel.com/mein/repos/tags*
`svn copy ^/trunk ^/branches/y3k`	Legt einen Branch an als *http://beispiel.com/mein/repos/branches/y3k*
`svn switch ^/branches/1.6.x`	Schaltet die Working Copy auf den Branch *http://beispiel.com/mein/repos/branches/1.6.x*
`svn merge ^/branches/y3k .`	Holt Änderungen von einem Branch

Man beachte, dass mit der Repository URL auch das URL-Schema der Working Copy übernommen wird (z.B. *https://*, *svn+ssh://*, *file:///*, ...).

Andere aus den *Externals* bekannte Kürzel ("`../`", "`/`" und "`//`") stehen allerdings auf der Kommandozeile *nicht* zur Verfügung, da sie identisch mit Unix-typischen lokalen Pfaden sind.

File-Externals

Visual SourceSafe [13] Benutzer mögen bisher ein *Pinning* Feature [14] in *Subversion* vermisst haben, mit dem eine Datei an einer bestimmten, typischerweise stabilen Version „festgesteckt" werden kann. *Subversion* bietet mit *Externals* [15] eine ganz ähnliche Funktionalität, jedoch konnten bisher nur Ordner als *Externals* angelegt werden (*Directory-Externals*). *Pinning* Freunde können sich in *Subversion 1.6* über die neuen *File-Externals* freuen.

Um Ordner- und nun auch Datei-*Externals* zu erstellen gibt man in Form des *Property* `svn:externals` für einen beliebigen übergeordneten Ordner eine Liste von *Subversion URLs* mit zugehörigen lokalen Pfaden an, die dann mit dem nächsten `update` oder `checkout` automatisch in jede Working Copy eingefügt werden. Insbesondere können dabei die „externen" Versionsnummern fixiert werden.

Da dieses *Property* wie jedes andere versioniert wird, lassen sich jederzeit vorige Zustände einsehen und wiederherstellen - ein grundlegender Vorteil gegenüber dem *Pinning* in *SourceSafe*.

Leider sollte man für einen ernsthaften Einsatz von Datei-*Externals* noch das eine oder andere Patch-Release abwarten (oder sich an der Fehlerbehebung beteiligen [16]), da auch 1.6.1 hier noch lästige Kinderkrankheiten aufweist:

- Ein wieder gelöschtes *File-External* verschwindet erst mit einem frischen `checkout` der Working Copy [17].
- Binäre Dateien als *File-Externals* funktionieren noch nicht [18].
- Ein *External* mit fixierter Revisionsnummer ist erst dann für einen `commit` gesperrt, wenn diese Revision ungleich `HEAD` ist [19].

Auch verhalten sich die beiden Typen von *Externals* unterschiedlich:

- Während externe Ordner auch von anderen Repositories stammen können, müssen *File-Externals* immer vom selben Repository stammen.
- Während bei *Directory-Externals*, die einige Unterverzeichnisse tiefer angelegt werden sollen, eventuell fehlende Verzeichnisglieder automatisch angelegt werden, muss man bei *File-Externals* bisher selbst darauf achten.
- *Directory-Externals* werden bei einem rekursiven `commit` wie gehabt übersprungen, die neuen *File-Externals* dagegen sind hier mit einbezogen.

In der *Subversion-Community* wird allgemein davon ausgegangen, dass die in Arbeit befindliche neue Working-Copy-Struktur [20] es recht einfach machen wird, den *Externals* ein einheitliches Verhalten zu geben.

Shard Packing

Das neue *Shard Packing* in *Subversion 1.6* macht Administratoren großer *FSFS* Repositorien [21] in Zukunft das Leben leichter. Bei Bedarf kann nun auf einfachste Weise das Dateisystem effizienter genutzt werden.

In einem *FSFS* Repository wird für jede eingehende Revision eine neue Datei angelegt. Insbesondere bei sehr vielen kleinen Revisionen können verschiedene Flaschenhälse durch unverhältnismäßigen Verwaltungs-Overhead auftreten:

- Das erste Problem wachsender Zugriffszeiten bei zu vielen Dateieinträgen in einem einzelnen Ordner wurde bereits in *Subversion 1.5* mit den sogenannten *Shards* (Scherben) gelöst [22]. *Shards* sind nichts anderes als mehrere Ordner, auf die alle Revisionsdateien verteilt werden, um die Anzahl der Dateieinträge pro Ordner klein zu halten.
- Das „Inode Problem" tritt auf, wenn die insgesamt verfügbare Anzahl von Dateieinträgen im Dateisystem (*Inodes*) erschöpft ist. Als Folge kann ein Datenträger „voll" sein, obwohl physikalisch noch Platz wäre.
- Durch Sektorengrenzen wird abhängig vom verwendeten Dateisystem für jede Datei im Schnitt 2 bis 16 kB mehr Platz belegt als Daten vorhanden sind, was insbesondere bei Anhäufungen kleiner Dateien sehr ins Gewicht fallen kann.
- Ebenso beeinträchtigen Anhäufungen kleiner Dateien die Performanz von Betriebssystemen durch häufige Systemaufrufe und fragmenthaftes Caching.

All diese Probleme können nun durch *Shard Packing* vermieden werden. Der Befehl `svnadmin pack` [23] erlaubt es, ganze *Shards* in jeweils eine einzelne große Datei zu packen. Je eine zweite Datei enthält einen Index von Offsets im gepackten *Shard*. Dies geschieht transparent auf dem Server und macht für die Nutzer keinerlei Unterschied. Zudem werden nur solche *Shards* gepackt, die bereits voll sind [24], sodass für einmal gepackte Dateien Unveränderbarkeit gewährleistet ist.

Shard Packing geschieht *nicht* automatisch, da es von der Situation abhängig ist, ob dadurch tatsächlich die Performanz steigt und Platz gespart wird. Auch weil es gegenwärtig nicht möglich ist, ein gepacktes *Shard* wieder zu *entpacken*, wird man erst bei sichtbaren Problemen zum *Shard Packing* greifen. Bis dahin kann man sich nunmehr unbeschwert dem Anhäufen von kleinsten Revisionen widmen.

Referenzen

[1] siehe komplette Liste unter http://subversion.tigris.org/svn_1.6_releasenotes.html

[2] http://svn.collab.net/viewvc/svn/trunk/notes/tree-conflicts/use-cases.txt

[3] http://subversion.tigris.org/issues/show_bug.cgi?id=898

[4] http://subversion.tigris.org/issues/show_bug.cgi?id=3210

[5] http://svn.collab.net/viewvc/svn/trunk/notes/tree-conflicts/use-cases-resolution.txt

[6] http://svnbook.red-bean.com/nightly/en/svn.tour.treeconflicts.html

[7] http://svnbook.red-bean.com/nightly/en/svn.serverconfig.netmodel.html#svn.serverconfig.netmodel.credcache

[8] Es gilt als unsicher. Auch wenn betreffende Dateien durch Zugriffsrechte gesichert sind können diese z.B. durch physikalischen Zugriff auf Hardware oder Backups umgangen werden.

[9] Siehe [auth] Sektion der Beispielskonfiguration in z.B. $HOME/.subversion/config: password-stores = gnome-keyring,kwallet.

[10] Der *Shell*-Befehl hierfür lautet export `gnome-keyring-daemon` (mit *single-backward-quotes* `).

[11] http://svnbook.red-bean.com/nightly/en/svn.advanced.externals.html#svn.advanced.externals.urlmagic

[12] http://svnbook.red-bean.com/nightly/en/svn.basic.in-action.html#svn.advanced.reposurls

[13] http://msdn.microsoft.com/en-us/library/aa302175.aspx

[14] http://msdn.microsoft.com/en-us/library/xa4cy26x(VS.80).aspx

[15] http://svnbook.red-bean.com/nightly/en/svn.advanced.externals.html

[16] http://subversion.tigris.org/hacking.html

[17] Issue #3368: http://subversion.tigris.org/issues/show_bug.cgi?id=3368

[18] Issue #3351: http://subversion.tigris.org/issues/show_bug.cgi?id=3351

[19] Issue #3401: http://subversion.tigris.org/issues/show_bug.cgi?id=3401

[20] *wc-ng* („*Working Copy - Next Generation*“)

[21] http://svnbook.red-bean.com/nightly/en/svn.reposadmin.planning.html#svn.reposadmin.basics.backends

[22] http://svnbook.red-bean.com/nightly/en/svn.reposadmin.planning.html#svn.reposadmin.basics.backends.fsfs.revfiles

[23] http://svnbook.red-bean.com/nightly/en/svn.reposadmin.maint.html#svn.reposadmin.maint.diskspace.fsfspacking

[24] Siehe auch *fsfs-reshard.py*: http://svnbook.red-bean.com/nightly/en/svn.reposadmin.maint.html#svn.reposadmin.maint.tk.fsfsreshard

Über die Autoren

Neels Hofmeyr studiert Technische Informatik an der Technischen Universität Berlin. Einen Teil seines Studiums absolvierte er an der University of Stellenbosch, Südafrika.

Stefan Sperling hat sein Informatikstudium an der Freien Universität Berlin abgeschlossen und ein ergänzendes Auslandsstudium am UCC in Cork, Irland. Abgeschlossen. Derzeit ist er für elego als Developer und Consultant tätig.

Stephen Butler arbeitet seit 2007 bei *elego* als Software-Entwickler und CM Consultant. Er war zuvor für Software-Firmen in den USA und Europa tätig. Bei *UBS* in der Schweiz arbeitete er an einem großen Migrationsprojekt von *CM Synergy* nach *Subversion*.

Alle drei sind über den Deutschen *Subversion*-Sponsor *elego Software Solutions GmbH* aktiv an der Entwicklung von *Subversion* beteiligt - Stefan seit Anfang 2007, Neels und Stephen seit Mitte 2008. Hinsichtlich *Subversion 1.6* wirkten sie an einigen der neuen Features direkt mit.

Die *elego Software Solutions GmbH* unterstützt aktuell mit vier Committern die Weiterentwicklung von *Subversion*. *elego* bietet in Kooperation mit *CollabNet*, dem Hauptsponsor der *Subversion*-Entwicklung, *Subversion*-Support für den europäischen Raum. D.h. Unternehmen können professionelle Unterstützung bei Installation, Migration und Integration, sowie der Administration und des Betriebs von *Subversion* in Anspruch nehmen.