Einführung in Subversion

Middleware-Übungen

Michael Gernoth

Informatik 4

28. Oktober 2006

Repository anlegen

- Das Repository ist der zentrale Datenspeicher des Revisionskontrollsystems
- Subversion bietet mehrere Repositoryformate an: `fsfs` und `bdb`
  - `fsfs` Mehrere kleine Dateien im Filesystem (Standard ab SVN 1.2)
  - `bdb` Speicherung in einer Datenbankdatei, nicht im NFS benutzbar

Erstellen eines neuen Repositories:

```
svnadmin create [--fs-type {fsfs|bdb}] /pfad/zum/Repository
> svnadmin create --fs-type fsfs /proj/i4mw/gernoth/aufgabe1
```


Checkout

- Dateien werden nicht im Repository editiert, sondern in einer eigenen ,,Working-Copy", welche ausgecheked wird
- Es können mehrere gleichzeitige Working-Copies vorhanden sein und (z.B. von verschiedenen Personen) bearbeitet werden

Auschecken aus einem lokalen Repository:

```
> svn co file:///proj/i4mw/gernoth/aufgabe1
```

Auschecken aus einem entfernet Repository per `ssh`:

```
> svn co svn+ssh://gernoth@faui01/proj/i4mw/gernoth/aufgabe1
```

Änderungen übernehmen

- Jede Änderung (z.B. editieren, anlegen, löschen, ... einer Datei) an der Working-Copy muss dem zentralen Repository mitgeteilt werden
- Jede Änderung bekommt eine eigene Log-Message zur Dokumentation

Änderungen an der Working-Copy ins Repository übernehmen:

```
> svn commit
<Hier öffnet sich ein Editor für die Log-Message>
Sending        aufgabe1.java
Transmitting file data .
Committed revision 10.
```


Working-Copy aktualisieren

- Um die im Repository gespeicherten Änderungen (z.b. einer anderen Person) in die eigene Working-Copy zu übernehmen, muss diese aktualisiert werden.
- Hierbei werden die Änderungen im Repository mit den lokalen Änderungen vereinigt (merging)
- Sollte das automatische mergen nicht erfolgreich sein, so entsteht ein Konflikt, welcher per Hand behoben werden muss.

Working-Copy aktualisieren:

```
> svn up
U    aufgabe1.java
Updated to revision 10.
```


Dateien hinzufügen

- Dateien, die in der Working-Copy angelegt werden, fallen nicht automatisch unter die Kontrolle der Versionsverwaltung
- Sie muessen dem System einzeln bekannt gemacht werden
- Nicht alle Dateien sollten versionsverwaltet werden (z.B. `*.class`, `*.o`, ...)

Hinzufügen der Datei `aufgabe1.java` zum Repository:

```
> svn add aufgabe1.java
A         aufgabe1.java
> svn commit
...
```


Dateien entfernen

- Wenn eine Datei aus dem Repository entfernt werden soll, bringt es nichts, diese in der Working-Copy zu löschen
- Das entfernen muss dem Versionsverwaltungssystem explizit mitgeteilt werden

Löschen der Datei `aufgabe1.java` aus dem Repository:

```
> svn rm aufgabe1.java
D         aufgabe1.java
> svn commit
...
```

Änderungen anzeigen

- Änderungen an einer Datei zwischen zwei Versionen bzw. den Unterschied zur Working-Copy herausfinden
- Sinnvoll, um vor dem `commit` nochmals alle Änderungen anzusehen

Unterschiede der Working-Copy zum Repository ausgeben:

```
> svn diff
Index: aufgabe1.java
===================================================================
--- aufgabe1.java       (revision 9)
+++ aufgabe1.java       (working copy)
@@ -1,5 +1,6 @@
 public class aufgabe1 {
        public static void main(String[] argv) {
+               //Gebe ,,Hallo Welt!'' aus
                System.out.println("Hallo Welt!");
        }
 }
```


Geschichte einer Datei anzeigen

- Die während des `commit` geschriebenen Log-Nachrichten zu einer Datei können abgerufen werden
- Hierdurch ist ein schneller Überblick über den Wandel einer Datei oder des gesamten Repositories möglich

Geschichte der Datei `aufgabe1.java` anzeigen:

```
> svn log aufgabe1.java
------------------------------------------------------------------------
r10 | simigern | 2006-10-23 14:53:03 +0200 (Mon, 23 Oct 2006) | 2 lines

comment added

...
```


,,Verantwortlichen" feststellen

- Möchte man den Verantwortlichen für eine bestimmte Codeänderung herausfinden, kann der jeweils letzte Autor einer Codezeile bei einer bestimmten Revision herausgefunden werden

Ausgabe der Verantwortlichen aller Änderungen an `aufgabe1.java` bis Revision 5:

```
> svn blame -r 5 aufgabe1.java
     4   simigern public class aufgabe1 {
     4   simigern       public static void main(String[] argv) {
     5    gernoth               System.out.println("Hallo Welt!");
     4   simigern       }
     4   simigern }
```


(Online-)Hilfe

Subversion bietet eine ausführliche Online-Hilfe auf der Kommandozeile:

```
> svn --help
usage: svn <subcommand> [options] [args]
...
Available subcommands:
   add
...

> svn add --help
add: Put files and directories under version control, scheduling
them for addition to repository.  They will be added in next commit.
...
```

Weitere Informationen zu Subversion sind auch hier zu finden:

- Das Subversion-Buch: `http://svnbook.red-bean.com/`
- Subversion-Homepage: `http://subversion.tigris.org/`

Dateisystem-ACLs

- Sollen mehrere Leute auf ein Verzeichnis zugreifen können, bieten sich Dateisystem-ACLs an
- Mit diesen können die Berechtigungen feingranular vergeben werden

Den Benutzern `gernoth` und `simigern` vollen Zugriff auf alle Dateien unterhalb von `/proj/i4mw/gernoth/aufgabe1` geben und diese auch für neue Dateien gewähren:

```
> setfacl -R -m default:user:gernoth:rwx /proj/i4mw/gernoth/aufgabe1
> setfacl -R -m default:user:simigern:rwx /proj/i4mw/gernoth/aufgabe1
> setfacl -R -m user:gernoth:rwx /proj/i4mw/gernoth/aufgabe1
> setfacl -R -m user:simigern:rwx /proj/i4mw/gernoth/aufgabe1
> setfacl -m user:simigern:r-x /proj/i4mw/gernoth
```

Weitere Informationen in `setfacl(1)` und `acl(5)`.

Bearbeitungshinweise zu den Übungsaufgaben

- Bearbeitung der Aufgaben in Zweiergruppen
- Für jede Aufgabe ein neues Subversion-Repository in `/proj/i4mw/<loginname>/aufgabeX` erstellen und (mindestens) die Lösung dorthin committen
- Die Liste der Bearbeiter (Loginnamen) in der Datei `/proj/i4mw/<loginname>/aufgabeX/bearbeiter` ablegen