Praktische Software-Entwicklung: Versionsverwaltung mit dem Concurrent Versions System

Institut für Informatik
Mustererkennung und Bioinformatik

Angewandte Bildverarbeitung, SS 2007

Überblick

Versionskontroll - Wozu das?

- Scenario:
  (viele) Dateien in einem Verzeichnis werden über längere Zeit immer mal wieder bearbeitet...
  - ...Welches ist die aktuelle Version?
  - ...Warum läuft der Code plötzlich nicht mehr?!
  - ...Was wurde zuletzt geändert? Wer war das?!
  - ...Wo ist die Sicherungskopie?!
- Aufgaben einer Versionskontrolle:
  - Dateien sichern
  - Änderungen tracken
  - Änderungen mehrerer Nutzer konsistent halten
- Software:
  - CVS - Concurrent Versions System
  - Subversion
  - ...

Das Concurrent Versions System (CVS)

- Grundideen:
  - CVS Verwaltet Dateien in einem zentralen *Repository*
  - jeder Nutzer hat eine eigene Kopie der Files *(Sandbox)*, die er beliebig verändern kann
- man kann jederzeit:
  - jede Version einer Datei zurückbekommen (!)
  - Änderungen zwischen beliebigen Versionen ansehen
  - herausfinden, welcher Nutzer welche Änderung gemacht hat (und Kommentare dazu ansehen)
  - Änderungen in seinen Dateien ins Repository einchecken
  - Änderungen anderer Nutzer aus dem Repository holen

Das Concurrent Versions System (CVS)

- das CVS verwaltet *Module*, die unabhängig voneinander sind
- jedes Modul besteht aus einer Verzeichnisstruktur
- jede Datei (und jedes Verzeichnis) eines Moduls erhält jeweils eine Versionsnummer (*Revision*)
  $\Rightarrow$ Aktualisierung beim Einchecken (oder manuell)
- Dateien können jederzeit...
  - neu hinzugefügt werden
  - entfernt werden (Archivkopie bleibt erhalten!)
  - verschoben werden
- mit Verzeichnissen ist das etwas schwieriger...

Sandboxes

- Annahme: irgendwo gibt es ein Repository
- Sandbox aus Repository auschecken:

  `cvs -d 'cvsroot' co 'module' | 'directory'`
- jedes Verzeichnis der Sandbox enthält ein Unterverzeichnis *CVS* mit wichtigen Daten (z.B. Ort des Repositories)
  $\Rightarrow$ Pfad nur beim ersten Checkout angeben
- eine neue Datei zum Repository hinzufügen:

  `cvs add 'filename'`
- Änderungen einchecken:

  `cvs commit -m "Kommentar" 'filename'`
- Unterschiede anzeigen:

  `cvs diff 'filename'`

```
> cvs -d /tmp/bspcvsroot co Prj/bspProjekt

cvs checkout: Updating Prj/bspProjekt
U Prj/bspProjekt/file1
cvs checkout: Updating Prj/bspProjekt/unterverzeichnis
U Prj/bspProjekt/unterverzeichnis/file2
```

```
Prj/
|-- CVS
`-- bspProjekt
    |-- CVS ...
    |-- file1
    `-- unterverzeichnis
        |-- CVS ...
        `-- file2
```

```
> cvs add file3

cvs add: scheduling file `file3' for addition
cvs add: use 'cvs commit' to add this file permanently
```

```
> cvs commit -m "mein 3. tolles file"

cvs commit: Examining .
cvs commit: Examining unterverzeichnis
RCS file: /tmp/bspcvsroot/Prj/bspProjekt/file3,v
done
Checking in file3;
/tmp/bspcvsroot/Prj/bspProjekt/file3,v  <--  file3
initial revision: 1.1
done
```

file3 bei commit:

Test File3

danach aendern zu:

Dies ist
Test File3

> cvs diff file3

```
Index: file3
===================================================================
RCS file: /tmp/bspcvsroot/Prj/bspProjekt/file3,v
retrieving revision 1.1
diff -r1.1 file3
0a1
> Dies ist
```

Änderungen übernehmen

- Geänderte Dateien im Repository anzeigen (-n = nichts verändern):

```
cvs -n update
```

- Änderungen aus Repository übernehmen:

```
cvs update 'filename'
```


```
> cvs -n update

cvs update: Updating .
U file1
M file2
cvs update: Updating unterverzeichnis
```

```
| file1  von jemand anderem geaendert von
|   Das hier  ist  flasch  geschrieben!
|
| zu
|   Das hier  ist  richtig  geschrieben!
```

```
> cvs update file1

U file1
```

```
| file1  danach:
|   Das hier  ist  richtig  geschrieben!
```

Mehrere Nutzer und Konflikte

- Konflikte entstehen (nur), wenn mehrere Nutzer an denselben Stellen ändern
- ”winner-takes-all”-Prinzip:
  nur der erste kann seine Daten einchecken (*commit*)
- wurde eine Datei im Repository geändert,
  muss erst die Sanbox auf den neuesten
  Stand gebracht werden (*update*)

file3 :	Aenderung von Nutzer1 eingecheckt:	Aenderung von Nutzer2:
1 Hier wird	1 Hier wird	1 Hier wird
2 ein Konflikt	2 ein von Nutzer1 Konflikt	2 ein Konflikt
3 produziert	3 produziert	3 produziert
		4 von Nutzer2

Nutzer2> cvs update

cvs update: Updating .
RCS file : / tmp/bspcvsroot/Prj/bspProjekt/ file3 ,v
retrieving revision 1.1
retrieving revision 1.2
Merging differences between 1.1 and 1.2 into file3
M file3
cvs update: Updating unterverzeichnis

file3 von Nutzer2 danach:
1 Hier wird
2 ein von Nutzer1 Konflikt
3 produziert
4 von Nutzer2

Neue Aenderung von Nutzer1	Aenderung, von Nutzer2 eingecheckt:
1 Hier wird	1 Hier wird
2 ein von Nutzer1 Konflikt	2 ein von Nutzer1 Konflikt
3 produziert von Nutzer1	3 produziert von Nutzer2
	4 von Nutzer2

Nutzer1> cvs update

```
cvs update: Updating .
RCS file : / tmp/bspcvsroot/Prj/bspProjekt/ file3 ,v
 retrieving  revision 1.2
 retrieving  revision 1.3
Merging differences between 1.2 and 1.3 into file3
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in file3
C file3
cvs update: Updating unterverzeichnis

 file3  von Nutzer1 danach: KONFLIKT!

1 Hier wird
2 ein von Nutzer1 Konflikt
<<<<<<< file3
3 produziert von Nutzer1
=======
3 produziert von Nutzer2
4 von Nutzer2
>>>>>>> 1.3
```

Repositories & Sandboxes

- Repository:
  - CVS speichert hier die Dateien und Zusatzdaten
  - beliebig viele anlegbar
    (auch auf anderen Rechnern im Netz)
  - wird normalerweise nie(!) direkt bearbeitet
- jedes Verzeichnis einer Sandbox
  - enthält ein Verzeichnis CVS mit Verwaltungsdateien
  - gehört zu genau einem Repository
    (Pfad steht in CVS/Root)
- Anlegen von Repositories:

```
cvs -d 'cvsroot' init
```

$\Rightarrow$ legt in *cvsroot* ein Verzeichnis CVSROOT an mit administrativen Dateien an

Repositories & Sandboxes

- Import einer Dateistruktur:

```
cvs -d 'cvsroot' import
-m''Kommentar'' dir vendor-tag release-tag
```


```
> cvs -d /tmp/bspcvsroot import -m "Import des Beispiels" Prj/bspProjekt autor initial

N Prj/bspProjekt/file1
cvs import: Importing /tmp/bspcvsroot/Prj/bspProjekt/unterverzeichnis
N Prj/bspProjekt/unterverzeichnis/file2

No conflicts created by this import
```

Daten im CVS

- in ein CVS darf *fast* alles:
  Text-Dateien, Programmcode, Bilder ...
- Was ist im CVS gut aufgehoben?
  - Dateien, die direkt verändert werden
  - Dateien, die *nicht* automatisch generiert werden
- Was gehört nicht in ein CVS?
  - grosse Menge statischer Daten (z.B. Testbilder)
  - Programmdateien, Objektfiles
  - lokale Daten, die für ein Projekt nicht relevant sind

Niemals. . .

- . . . die CVS-Verzeichnisse direkt bearbeiten!
- . . . die CVS-Verzeichnisse in der Sandbox löschen!
- . . . das CVS-Repository verschieben!

CVS in dieser Veranstaltung

- zentrales Projektverzeichnis:
  - auf den Rechnern der AG:

`sol:/vol/export/angewandteBV_SS07/`*`project`*`/CVSROOT`

  - im Pool:

`/lehre/agprbio/angewandteBV_SS07/`*`project`*`/CVSROOT`

- externer Zugriff per ssh:

```
cvs -d
login@sol.informatik.uni-halle.de:/vol/export/...
```