Das Concurrent Versions System (CVS)

Vortrag im Proseminar Unix Tools 1

Markus Sander
sander@in.tum.de

07.06.2005

Gliederung

Versionenkontrolle

1. Definition
   Versionenkontroll-Software verwaltet Aktualisierung von Dateien (i.A. Quellcode).
2. Motivation
   - Änderungen am Quellcode können neue Fehler einführen. Diese könnten lange unerkannt bleiben.
   - Wann wurde der Fehler eingeführt? Von wem?
   - Wie sah der korrekte Code aus?
   - Koordination vom gemeinsamen Arbeiten mehrerer Entwickler am gleichen Projekt ist schwierig.

Lösung: Concurrent Versions System (CVS)

https://www.cvshome.org

- CVS speichert Änderungen an Dateien.
- Die fehlerhafte Änderung kann gefunden werden. Der korrekte Code kann nachgeschlagen werden.
- CVS verwaltet auch die Distribution der Dateien. So, dass mehrere Entwickler an selber Datei arbeiten können.
- CVS kann Änderungen in disjunkten Teilen einer Datei automatisch verschmelzen.

Versionen, Revisionen, Releases

CVS-Terminologie

- Jede Datei (und nur Dateien) unter Versionenkontrolle besitzt eine Revisionsnummer.
- Jede vorgenommene Änderung an einer Datei erhöht ihre Revisionsnummer, z.B. von `1.1` auf `1.2`.
- Release: Software-Version. CVS hat kein Verständnis von Releases.
- Man spricht häufig von Version, wenn man eigentlich entweder Revision oder Release meint. CVS meidet mehrdeutiges ”Version”.

Aufruf

- Allgemeiner CVS-Aufruf:

```
cvs [global-opt] command
  [command-opt] [command-args]
```

- Beispiele für globale Optionen und Befehle werden wir in Kürze sehen. Die gebräuchlichsten finden sich auch in der Kurzreferenz im Handout.

Das CVS-Repositorium

bzw. engl: repository

- CVS-Repositorium: Verzeichnisbaum, in dem verwaltete Dateien und Verzeichnisse abgelegt werden.
- Alle Revisionen aller Dateien befinden sich dort, aber in kompakter Form. Es werden nicht komplette Dateien gespeichert, sondern Änderungen gegenüber der jeweils vorigen Revision.
- Somit kann jederzeit jede beliebige Revision synthetisiert werden.

$CVSROOT
Spezifi zieren eines CVS-Repositoriums

- Einige CVS-Befehle interagieren mit (genau) einem Repositorium.
- Dann muss CVS die Adresse mitgeteilt werden: Mit der globalen Option `-d`, oder über Umgebungsvariable $CVSROOT (wobei die Kommandozeile die höhere Priorität hat).

Pfad zum Repositorium

Spezifi zieren eines CVS-Repositoriums

- Ein Pfad zu einem Repositorium hat die Form

```
[:method:]
[[[user][:password]@]hostname[:port]]
  /absolute/path/to/repository
```

- `method` gibt die Zugriffsmethode an, z.B. `pserver` oder `ext`.
- CVS via SSH: Setze `CVS_RSH = /usr/bin/ssh` und `method = ext`

cvs init
Erstellen eines Repositoriums

- Ein Repositorium wird erstellt:

```
$ mkdir cvsroot
$ export CVSROOT=‘pwd‘/cvsroot
$ cvs init
$ cd cvsroot
$ ls -a
./ ../ CVSROOT/
```

- Im `CVSROOT/` Unterverzeichnis werden Administrationsdateien gespeichert. Für einfache Anwendungen sind die Standardwerte ausreichend.

Einfache Module

...werden durch lediglich Erstellen eines Verzeichnisses defi niert

- Ein Modul ist eine im Repositorium verwaltete Menge an Dateien und Verzeichnissen.
- Jedes (direkte) Unterverzeichnis des Repositoriums fungiert als Modul.
- Das gilt insbesondere auch für das `CVSROOT/` Unterverzeichnis. Die Konfigurationsdateien stehen also ebenfalls unter Versionenkontrolle.

`cvs checkout`
Erstellen einer Arbeitskopie

- Direktes Arbeiten am Repositorium gäbe genau die Probleme, die wir mit CVS lösen möchten.
- Deshalb muss eine Arbeitskopie (auch: `sandbox`) verwendet werden.
- Arbeitskopien dürfen nicht innerhalb des Repositoriums erstellt werden.

`cvs checkout`
Erstellen einer Arbeitskopie

- Eine Arbeitskopie des Moduls `mod` wird im aktuellen Verzeichnis erstellt:

```
$ cvs checkout mod
cvs checkout: Updating mod
U mod/file1
$ ls -a mod/
./ ../ CVS/ file1
```

- Im `CVS/` Unterverzeichnis ist der Pfad zum Repositorium gespeichert. Weitere Operationen auf `mod` sind nicht mehr auf `$CVSROOT` angewiesen.
- Das `U` vor einem Dateinamen zeigt an, dass die jeweilige Datei in der Arbeitskopie *jetzt* `Up-to-date` ist.

cvs add
Hinzufügen von Dateien und Verzeichnissen

- Eine Datei `file2` wird zum Hinzufügen vorgemerkt:

```
$ ls -a
./ ../ CVS/ file1
$ > file2
$ cvs add file2
cvs add: scheduling file ‘file2’ for addition
cvs add: use ’cvs commit’ to add this file
permanently
```

- Verzeichnisse werden analog zum Hinzufügen markiert.

cvs add
Hinzufügen von Dateien und Verzeichnissen

- Der hinzuzufügende Dateisystemeintrag muss existieren, darf aber nicht bereits unter Versionenkontrolle stehen.
- Binärdateien brauchen eine Sonderbehandlung: Die Befehlsoption `-kb`. Denn:
  - CVS konvertiert automatisch zwischen UNIX- und Windows-Newlines.
  - Binärdateien können Schlüsselworte enthalten (-> *Schlüsselwortsubstitution*).

cvs commit
Ablegen von Änderungen

- Vorgenommene Änderungen werden eingereicht:

```
$ cvs commit -m "Added: mod/file2"
cvs commit: Examining .
Checking in mod;
/path/to/cvsroot/mod/file2,v <- file2
initial revision: 1.1
done
```

- Dateien müssen vor dem Einreichen aktualisiert werden. Sonst könnten Änderungen anderer Entwickler überschrieben werden.

`cvs commit`
Ablegen von Änderungen

- Eine Lognachricht muss angegeben werden. Wenn nicht durch Option `-m`, dann wird `$CVSEDITOR`, `$EDITOR`, `$VISUAL` oder `vi` gestartet.

`cvs update`

Verschmelzen von Änderungen anderer Entwickler mit der Arbeitskopie

- Ein Arbeitsverzeichnis wird aktualisiert:
  ```
  $ cvs update -dP
  cvs update: Updating .
  ```
- `file1` könnte seit dem letzten `update` verändert worden sein. Dann wird eine Verschmelzung versucht. Lokale Änderungen gehen dabei nicht verloren.

`cvs update`

Verschmelzen von Änderungen anderer Entwickler mit der Arbeitskopie

- Erster Fall:

  ```
  U file1
  ```

  `file1` war unverändert und kann auf den neuesten Stand gebracht werden.
- Zweiter Fall:

  ```
  M file1
  ```

  `file1` enthielt lokale Änderungen, die der Aktualisierung nicht im Wege standen.
- Dritter Fall:

  ```
  C file1
  ```

  `file1` enthielt lokale Änderungen, die der Aktualisierung im Wege standen.

`cvs update`

Verschmelzen von Änderungen anderer Entwickler mit der Arbeitskopie

- `file1` enthält nun einen Konflikt. Die aktuelle Revision aus dem Repositorium ist inkompatibel zur Arbeitskopie.
- In `file1` befinden sich an Konfliktpunkten, durch Marker getrennt, die beiden Revisionen. Konflikte müssen manuell aufgelöst werden. Dann kann die Kopie eingecheckt werden.
- Die lokale Datei vor dem `update` ist in `.#file1.<rev>` oder `__#file1.<rev>` gespeichert.

cvs update
Manuelles Auflösen von Konflikten

Es sei dies die Revision 1.4 einer Datei driver.c:

```
void main() {
   parse();
   if (nerr == 0)
      gencode();
   else
      fprintf(stderr, ”No code generated.\n”);
   exit(nerr == 0 ? 0 : 1);
}
```

cvs update
Manuelles Auflösen von Konflikten

Revision 1.6 von driver.c:

```
int main(int argc, char **argv) {
    parse();
    if (argc != 1) {
        fprintf(stderr, "tc: Need 0 args.\n");
        exit(1);
    }
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(!!nerr);
}
```

cvs update
Manuelles Auflösen von Konflikten

Dies sei unsere Arbeitskopie von `driver.c`, basierend auf Revision `1.4`:

```
void main() {
    init_scanner();
    parse();
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(nerr == 0 ? SUCCESS : FAILURE);
}
```

cvs update
Manuelles Auflösen von Konflikten

- Wir führen ein `update` aus:

```
$ cvs update driver.c
/path/to/cvsroot/module/driver.c,v
retrieving revision 1.4
retrieving revision 1.6
Merging differences between 1.4 and 1.6 into
driver.c
rcsmerge warning: overlaps during merge
cvs update: conflicts found in driver.c
C driver.c
```

- Nun betrachten wir den Inhalt von `driver.c` nach dem `update`:

```
int main(int argc, char **argv) {
    init_scanner();
    parse();
    if (argc != 1) {
        fprintf(stderr, "tc: Need 0 args.\n");
        exit(1);
    }
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
<<<<<<< driver.c
    exit(nerr == 0 ? SUCCESS : FAILURE);
=======
    exit(!!nerr);
>>>>>>> 1.6
}
```

Vielen Dank für die Aufmerksamkeit.

Konflikte
Diskussion

- Offenbar machen Konflikte Arbeit.
- Wäre es nicht besser, es könnte jede Datei nur von *einem* Entwickler gleichzeitig bearbeitet werden (`reserved checkouts`)?
- Andere Versionenkontrollsysteme erzwingen das. CVS unterstützt es auch.
- Dies behindert Entwickler, die disjunkte Teile einer Datei ändern wollen.
- Entwickler vergessen gerne, `locks` wieder frei zu geben.

Konflikte
Diskussion

- Ernste Konflikte treten nur auf, wenn Entwickler bzgl. des *Designs* uneinig sind.
- Kommunikation und Planung verhindern ernste Konflikte.
- Häufiges Synchronisieren kann zu einfacher aufzulösenden Konflikten führen.
- *Aber*: Es gibt Situationen, in denen `unreserved checkouts` immer fehl am Platz sind: z.B. wenn es kein Werkzeug zum Verschmelzen von Änderungen gibt.

Branching
Parallele Entwicklungsstränge

- Motivation:
  - Release 1.0 unserer Software wurde herausgegeben.
  - Ein Kunde entdeckt kritischen Fehler.
  - Die Entwicklung in Richtung Release 2.0 ist momentan zu instabil.
- Lösung:
  - Starte ab Release 1.0 neuen Entwicklungszweig, der die Änderungen für Release 2.0 nicht enthält.
  - Repariere hier die Fehler.
  - Bevor Release 2.0 herausgegeben wird, verschmelze die Änderungen des Patch-Zweiges mit dem Hauptzweig.

Branching
Parallele Entwicklungsstränge

- Bezeichne die Revisionen vom `date` mit `rel-1-0`:
  `$ cvs tag -D date rel-1-0`
- Starte neuen Entwicklungszweig `rel-1-0-fixes`:
  `$ cvs tag -r rel-1-0 -b rel-1-0-fixes`
- Erstelle neue Arbeitskopie für den Zweig:
  `$ cvs checkout -r rel-1-0-fixes module`

Branching
Parallele Entwicklungsstränge

- Neue Änderungen werden nur dem Zweig `rel-1-0-fixes` hinzugefügt.
- Verschmelze Zweig `rel-1-0-fixes` mit dem `main trunk`:

```
$ cd main-trunk-sandbox
$ cvs update -dP
$ cvs update -j rel-1-0-fixes
```

Branching
Parallele Entwicklungsstränge

- Mehrfaches Verschmelzen eines Zweiges problematisch. Manche Änderungen würden öfter mit Arbeitskopie verschmolzen werden.
- Lösung: Nachdem ein Zweig mit dem `main trunk` verschmolzen wurde, weise ihm ein tag zu, z.B. `rel-1-0-fixes-latest-merge`.
- Verschmelze nur die Änderungen seit der letzten Verschmelzung:
  ```
  $ cvs update -j rel-1-0-fixes-latest-merge -j
  rel-1-0-fixes
  ```