Versionskontrolle mit RCS

Johannes Franken
<jfranken@jfranken.de>

Auf dieser Seite zeige ich den Umgang mit einem Versionskontrollsystem am Beispiel des GNU Revision Control System.

Inhalt

Versionskontrolle

Der Einsatz eines Versionskontrollsystems ermöglicht folgende Tätigkeiten:

- **Undo:** Zurückliegende Zustände eines Dateiinhalts wiederherstellen.
- **Teamwork:** Mit mehreren Benutzern den selben Satz von Dateien bearbeiten. Wenn mehrere Autoren versehentlich gleichzeitig die selbe Datei verändern, würde der zuletzt speichernde Autor die Änderungen seiner Kollegen überschreiben, ihre Arbeit wäre verloren. Ein Versionskontrollsystem ermittelt die veränderten Zeilen und setzt sie so zusammen, als ob die Autoren die Datei nacheinander bearbeitet hätten.
- **Changelogs:** Ausgabe aller Veränderungen: pro User, Zeitraum oder Datei/Verzeichnis.
- **Branches:** Durch die Aufteilung auf unterschiedliche Entwicklungszweige können einzelne Bugfixes oder Konfigänderungen zunächst in einer Entwicklungs-Version getestet und im Erfolgsfall "per Knopfdruck" in die offizielle Version übernommen werden.

Es ist offensichtlich, dass die Anwendung dieser Möglichkeiten einen positiven Einfluß auf die Arbeitsgeschwindigkeit von Teams und die Qualität der Ergebnisse bringt.

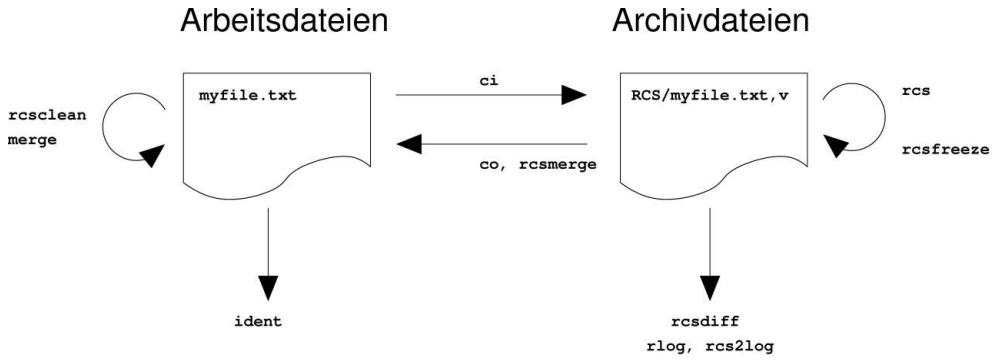
Das GNU Revision Control System (RCS)

Übersicht

RCS wurde 1982 bis 1995 von Walter F. Tichy und Paul Eggert als freie Software entwickelt. Die folgende Dokumentation bezieht sich auf die bis heute (2003) aktuelle Version GNU RCS 5.7 aus dem Jahre 1995.

Theoretische Grundlagen: siehe Tichy-RCS-Paper (21 Seiten PDF).

RCS erstellt für jede Arbeitsdatei eine Archivdatei, in der es die Entwicklung in Form von Deltas speichert. Die Übertragung in und aus den Archivdateien erfolgt mit Hilfe der Programme **`ci`** (für "check-in") und **`co`** (für "check-out"). Zum Schutz vor gegenseitigem Überschreiben kann ein User seine Änderungen nur in diejenigen Versionszweige der Archivdatei einchecken, die er zuvor gelockt hatte. Und schließlich gibt es noch einige Admin- und Auswertungstools:



Installation und Konfiguration

Die Installation von GNU RCS erfolgt idealerweise auf Paketbasis, unter Debian 3.0 beispielsweise mit dem Befehl

```
$ apt-get install rcs
```

RCS verlangt keine Konfiguration, man kann jedoch über die Environmentvariable **`RCSINIT`** einige der Optionen vorbelegen, die man bei *jedem* Aufruf von **`ci`** oder **`co`** verwenden möchte:

Option	Bedeutung
`-q`	quiet
`-V4` oder **`-V3`**	zum Auschecken aus Archiven, die mit RCS Version 3 oder 4 erstellt wurden.
`-x`SUFFIX	Suffix und Verzeichnis der Archivdateien
`-zLT`	Bei der Verwendung von Schlüsselworten statt GMT die lokale Zeit und Zeitzone ausgeben

Beispiel:

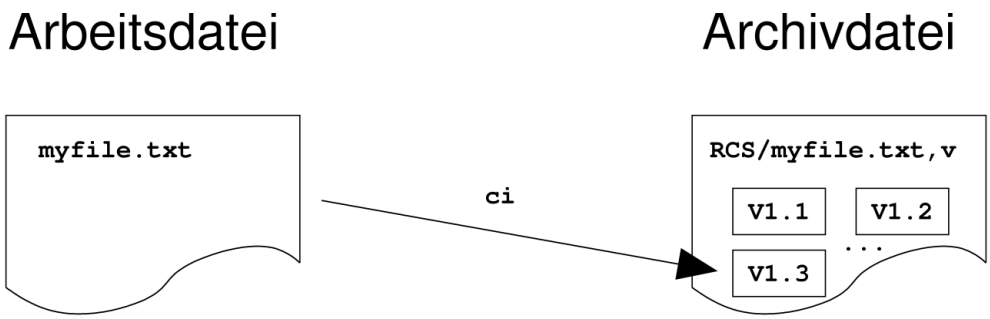
```
$ echo 'export RCSINIT="-zLT -q"' >> ~/.bashrc
```

Wer seine Arbeitsdateien von den Archivdateien trennen möchte, kann in jedem Arbeitsdateien-Verzeichnis ein **`RCS`**-Unterverzeichnis anlegen. Die RCS-Tools werden dann die Archivdateien automatisch darin anlegen und suchen, sofern man ihnen nicht mit dem **`-x`**-Parameter einen anderen Verzeichnisnamen vorgibt.

Mit Archiven arbeiten

ci

ci (für "check-in") fügt eine neue Version einer Arbeitsdatei zu seiner Archivdatei hinzu. Beim ersten Aufruf für eine Arbeitsdatei erstellt es die Archivdatei automatisch.



Beispiel:

```
$ ls -l
-rw-r--r--    1 jfranken users          15 10. Jan 14:25 myfile.txt
$ cat myfile.txt
RCS-Test
Nr. 1
$ ci -l myfile.txt
myfile.txt,v  <--  myfile.txt
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> test check-in
>> .
initial revision: 1.1
done
$ ls -l
-rw-r--r--    1 jfranken users          15 10. Jan 14:25 myfile.txt
-r--r--r--    1 jfranken users         223 10. Jan 14:25 myfile.txt,v
$ echo + Nr. 2 >> myfile.txt
$ ci -l myfile.txt
myfile.txt,v  <--  myfile.txt
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> addition Nr. 2
>> .
done
$ ls -l
-rw-r--r--    1 jfranken users          23 10. Jan 14:33 myfile.txt
-r--r--r--    1 jfranken users         351 10. Jan 14:34 myfile.txt,v
$
```

Um eine Version später besser finden zu können, kann man mit dem Parameter **-s** einen beliebigen Statustext übergeben (z.B. **stable**). Der voreingestellte Statustext lautet sonst **Exp**.

Locking:
Einchecken kann nur der User, der das "Lock" (Schreibberechtigung) auf diesen Versionszweig in der Archivdatei besitzt. Der Parameter **-l** bewirkt, dass er das Lock behält, also zum Einchecken weiterer Änderungen berechigt bleibt. Alle übrigen User sehen bei einem Eincheck-Versuch folgende Meldung:

```
$ ci -l myfile.txt
myfile.txt,v  <--  myfile.txt
ci: myfile.txt,v: no lock set by bfranken
```

Bevor sie ihre Änderungen einchecken können, müssen sie entweder

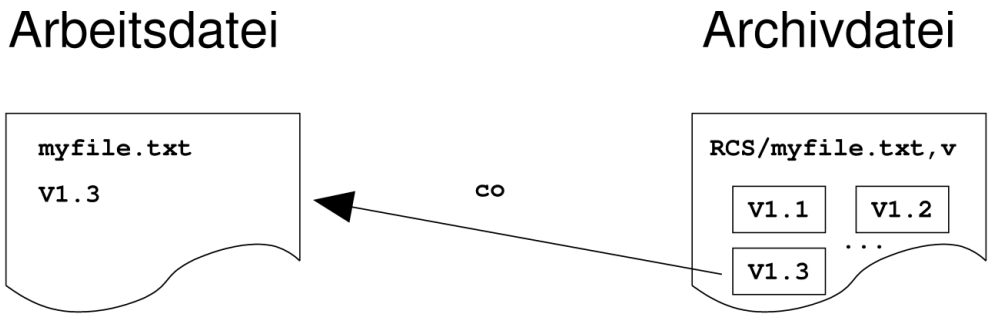
- warten, bis das Lock entfernt wurde (z.B. indem der Besitzer **ci** ohne **-l** oder mit **-u** aufruft) und dann mit **co -l** die Änderungen des Vorgängers übernehmen.
- das Lock administrativ brechen (siehe rcs), oder
- sich für einen anderen oder neuen Versionszweig entscheiden (siehe Branches).

Der Parameter **-u** verhindert übrigens, dass **ci** die Arbeitsdatei nach dem Einchecken löscht.

Mehr zum Thema:
Weitere Optionen: siehe ci(1) manpage

co

co (für "check-out") extrahiert eine Arbeitsdatei aus der Archivdatei.



Wenn man den Parameter **-p** angibt, gibt **co** die Version auf STDOUT aus.

Auswahl der auszucheckenden Version:
Normalerweise checkt **co** automatisch die aktuellste Version aus.

Mit dem Parameter **-r** kann man die Versionsbezeichnung der auszucheckenden Version angeben:

```
$ ls -l
-r--r--r--    1 jfranken users         337 10. Jan 15:48 myfile.txt,v
$ co -r1.1 myfile.txt,v
myfile.txt,v  -->  myfile.txt
revision 1.1
done
$ ls -l
-r--r--r--    1 jfranken users          15 10. Jan 15:49 myfile.txt
-r--r--r--    1 jfranken users         337 10. Jan 15:48 myfile.txt,v
$
```

Mit dem **-d**-Parameter kann man die letzte vor dem übergebenen Datum eingecheckte Version anfordern, mit **-w** sucht **co** nur nach Updates duch den übergebenen Autor und mit **-s** nur nach solchen mit dem übergebenen Statustext.

Locking:

co erstellt die Arbeitsdateien schreibgeschützt. Wer Änderungen an der ausgecheckten Arbeitsdatei vornehmen und später selbst einchecken möchte, muß beim Auschecken mit dem Parameter **-l** ein Lock beantragen:

```
$ co -l myfile.txt
myfile.txt,v  -->  myfile.txt
revision 1.2 (locked)
done
```

Das funktioniert jedoch nur, solange noch kein anderer User diesen Versionszweig gelockt hat, sonst erscheint eine Fehlermeldung:

```
$ co -l myfile.txt
myfile.txt,v  -->  myfile.txt
co: myfile.txt,v: Revision 1.2 is already locked by jfranken.
```

In diesem Fall sollte man den Besitzer des Locks bitten, seine Änderungen mit **`ci`** (ohne **`-l`**) einzuchecken, oder (notfalls) administrativ das Lock mit **`rcs -u`** brechen.

Mehr zum Thema:
Weitere Optionen: siehe co(1) manpage

rcs

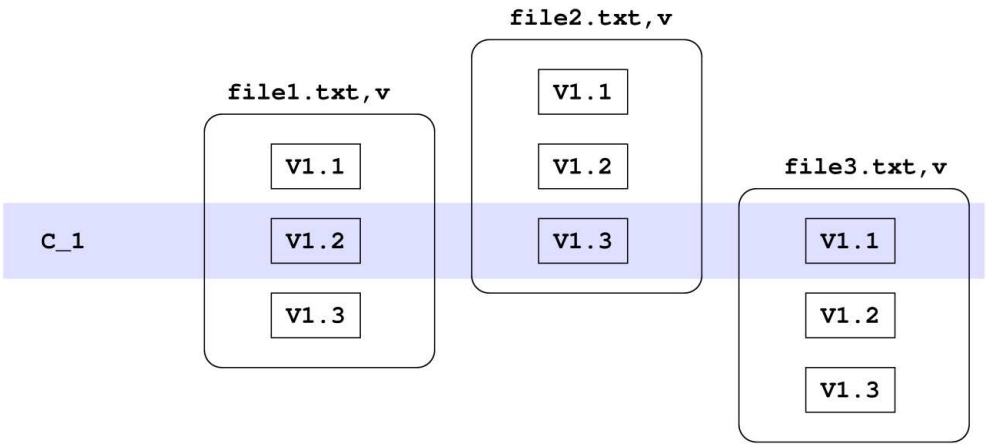
`rcs` verändert Archivdateien. Man kann damit z.B.

- einem User das Lock erteilen (**`-l`**) oder nehmen (**`-u`**),
- nachträglich das Changelog (**`-m`**) oder den Status (**`-s`**) einer Version ändern,
- eine Version löschen (**`-o`**),
- die Beschreibung ändern (**`-t`**) oder
- die Berechtigung zum Check-in einzelnen Usern erteilen (**`-a`**) oder nehmen (**`-e`**).

Mehr zum Thema:
Weitere Optionen: siehe rcs(1) manpage

rcsfreeze

`rcsfreeze` ist ein Shellscript, das der aktuellsten Version in jeder Archivdatei im aktuellen Verzeichnis jeweils einen gemeinsamen Versionsnamen zuordnet.



In diesem Beispiel war **`rcsfreeze`** aufgerufen worden, als **`file1.txt`** gerade in Version 1.2 und **`file3.txt`** in Version 1.1 vorlag. **`file2.txt`** ist seitdem nicht mehr aktualisiert worden. **`rcsfreeze`** hat dieser Zusammenstellung den Namen **`C_1`** gegeben.

Wozu das Ganze? - Den Versionsnamen kann man beim Auschecken alternativ zu einem Datum angeben:

```
$ for a in RCS/*,v; do co -rC_1 $a; done
```

Mehr zum Thema:
Weitere Optionen: siehe rcsfreeze(1) manpage

rcsclean

rcsclean löscht die ungelockten (d.h. mit **co** ohne **-l** ausgecheckten) Arbeitsdateien aus dem aktuellen Verzeichnis:

```
$ ls -l
-r--r--r--    1 jfranken users          23 20. Jan 22:05 myfile.txt
-r--r--r--    1 jfranken users         346 20. Jan 22:04 myfile.txt,v
$ rcsclean
rm -f myfile.txt
$ ls -l
-r--r--r--    1 jfranken users         346 20. Jan 22:04 myfile.txt,v
$
```

Wenn man den Parameter **-u** angibt, löscht **rcsclean** zusätzlich die unveränderten, gelockten (d.h. mit **co -l** ausgecheckten) Arbeitsdateien und gibt die Locks wieder frei.

Mehr zum Thema:
Weitere Optionen: siehe rcsclean(1) manpage

merge

merge mischt die Änderungen, die zwei Autoren an Kopien einer Textdatei vorgenommen haben.

Beispiel: Angenommen, eine Datei hat ursprünglich folgenden Inhalt:

```
$ cat orig
Q1:      Why do ducks have big flat feet?

Q2:      Why do elefants have big flat feet?
```

Wenn jetzt zwei User an Kopien dieser Datei Änderungen vornehmen: einer im oberen Teil

```
$ cat changed1
Q1:      Why do ducks have big flat feet?
A1:      To stamp out forrest fires.

Q2:      Why do elefants have big flat feet?
```

und der andere im unteren:

```
$ cat changed2
Q1:      Why do ducks have big flat feet?

Q2:      Why do elefants have big flat feet?
A2:      To stamp out flaming ducks.
```

dann kann man diese mit **merge** zusammenführen:

```
$ merge -p changed1 orig changed2
Q1:      Why do ducks have big flat feet?
A1:      To stamp out forrest fires.

Q2:      Why do elefants have big flat feet?
A2:      To stamp out flaming ducks.
```

Wenn die User ungeschickterweise die selben Zeilen geändert haben, kennzeichnet **merge** diese als "Konflikt":

```
$ echo 'A2:      No idea :-(' >> changed1
$ merge -p changed1 orig changed2
Q1:      Why do ducks have big flat feet?
A1:      To stamp out forrest fires.

Q2:      Why do elefants have big flat feet?
<<<<<<< changed1
A2:      No idea :-(
=======
A2:      To stamp out flaming ducks.
>>>>>>> changed2
merge: warning: conflicts during merge
```

Mehr zum Thema:
Weitere Optionen: siehe merge(1) manpage

Mit Schlüsselworten arbeiten

Schlüsselworte

Wenn in der Arbeitsdatei die folgenden Schlüsselworte vorkommen, wird RCS beim nächsten Check-out dort die folgenden Werte einfügen:

Schlüsselwort	Wert
$Author$	Der Unix Username des Users, der diese Datei eingecheckt hatte
$Date$	Datum und Uhrzeit des Einchecken
$Header$	Dateiname (inkl. Pfad), Datum, Autor, Status
Id	Dateiname (ohne Pfad), Datum, Autor, Status
Log	Das Changelog
$Name$	Der symbolische Name, der beim Auschecken dieser Version verwendet wurde
$Source$	Der Dateiname der Archivdatei (mit Pfad)
$RCSFile$	Der Dateiname der Archivdatei (ohne Pfad)
$Revision$	Die Versionsnummer
$State$	Der Statustext (Default: **Exp**)

ident

ident gibt alle Schlüsselwörter und deren Werte aus, die in den übergebenen Dateien enthalten sind.

Beispiel: Ich verwende den wml-Compiler, um aus einer ***.wml**- und mehreren ***.inc**-Dateien meine Webseiten (***.html**) zu erstellen. Mit dem **ident**-Befehl kann ich feststellen, aus welchen Versionen der Sourcecodes jede Webseite berechnet wurde:

```
$ ident -q ssh2.wml ssh2.de.html
ssh2.wml:
     $Id: rcs.wml,v 1.16 2006/01/06 19:54:46 jfranken Exp $

ssh2.de.html:
     $Id: rcs.wml,v 1.16 2006/01/06 19:54:46 jfranken Exp $
     $Id: template.inc,v 1.29 2003/01/02 13:58:47 jfranken Exp
```

Ergebnis: Das Kompilat **ssh2.de.html** ergibt sich aus der vorliegenden Version von **ssh2.wml** sowie **template.inc v1.29**.

Mehr zum Thema:
Weitere Optionen: siehe ident(1) manpage

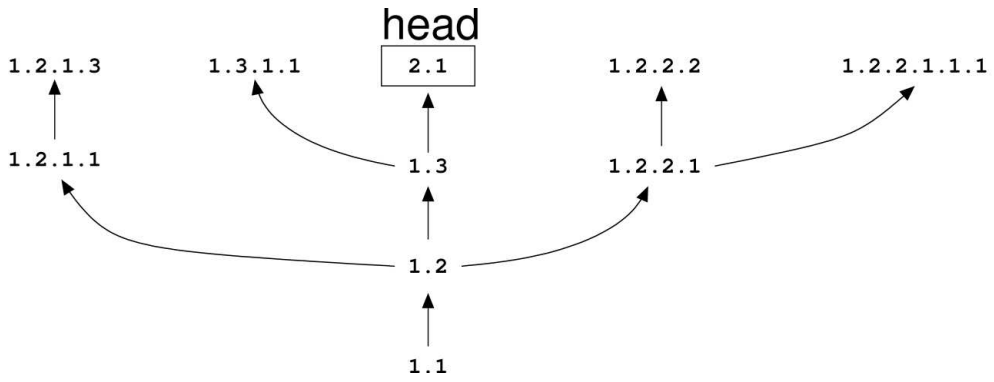
Mit Branches arbeiten

Übersicht

Branches bieten die Möglichkeit, eine Folge spezieller Änderungen zunächst alleine zu testen und erst im Erfolgsfall mit den übrigen Änderungen zu mischen, die möglicherweise in der Zwischenzeit entstanden sind. Auf diese Weise kann man Unverträglichkeiten verschiedener Bugfixes vor der Ausgabe einer offiziellen Version feststellen und beseitigen.

In komplexen Software-Projekten bietet es sich an, permanent mehrere Entwicklungszweige (z.B. "unstable", "testing", "stable") zu betreiben.

Die Versionsnummer bestimmt, zu welchem Zweig eine Version gehört: Der erste, auf der Versionsnummer **`x.y`** aufsetzende Zweig enthält die Versionsnumern **`x.y.1.1`** bis **`x.y.1.n`**, der zweite **`x.y.2.1`** bis **`x.y.2.n`** usw.



Einen neuen Versionszweig anlegen

Um einen neuen Versionszweig anzulegen, übergibt man beim Einchecken eine Versionsnummer mit einer zusätzlichen Stelle:

```
$ ci -l -r1.2.1 myfile.txt
myfile.txt,v  <--  myfile.txt
new revision: 1.2.1.1; previous revision: 1.2
enter log message, terminated with single '.' or end of file:
>> test branch
>> .
done
```

Eine Version aus einem Versionszweig auschecken

Um eine Version aus einem Versionszweig auszuchecken, gibt man beim Check-out einfach die gewünschte Versionsnummer an:

```
$ co -l -r1.2.1.1 myfile.txt,v
myfile.txt,v  -->  myfile.txt
revision 1.2.1.1 (locked)
done
```

Wenn man diese Version nach der Bearbeitung wieder in den Zweig einchecken möchte, genügt ein einfacher Check-in-Aufruf:

```
$ echo goes to same branch >> myfile.txt
$ ci myfile.txt
myfile.txt,v  <--  myfile.txt
new revision: 1.2.1.2; previous revision: 1.2.1.1
enter log message, terminated with single '.' or end of file:
>> checking ci for branches
>> .
done
```

rcsmerge

`rcsmerge` wendet die Änderungen zwischen den übergebenen Versionen auf eine Arbeitsdatei an. So kann man z.B. in verschiedene Entwicklungszweige eingecheckte Bugfixes zu einer neuen Arbeitsdatei zusammenführen.

Mehr zum Thema:
Weitere Optionen: siehe rcsmerge(1) manpage

Archivdateien auswerten

rlog

`rlog` gibt Einblicke in den Inhalt von Archivdateien:

```
$ rlog *,v
RCS file: myfile.txt,v
Working file: myfile.txt
head: 1.2
branch:
locks: strict
access list:
symbolic names:
        C_1: 1.2
keyword substitution: kv
total revisions: 4;     selected revisions: 4
description:
test checkin
----------------------------
revision 1.2
date: 2003/01/10 13:34:20;  author: jfranken;  state: Exp;  lines: +1 -0
branches:  1.2.1;
addition Nr. 2
----------------------------
revision 1.1
date: 2003/01/10 13:25:44;  author: jfranken;  state: Exp;
Initial revision
----------------------------
revision 1.2.1.2
date: 2003/01/27 06:22:19;  author: jfranken;  state: Exp;  lines: +1 -0
checking ci for branches
----------------------------
revision 1.2.1.1
date: 2003/01/24 14:56:54;  author: jfranken;  state: Exp;  lines: +1 -0
test branch
=============================================================================
```

Mehr zum Thema:
Weitere Optionen: siehe rlog(1) manpage

rcs2log (CVS)

`rcs2log` ist Bestandteil des CVS-Pakets und erstellt Changelogs aus den Archivdateien. Wenn man ihm keine Archivdateien übergibt, sucht es die Archivdateien in allen Unterverzeichnissen.

```
$ rcs2log -v
2003-01-27  Johannes Franken  <jfranken@tp>

        * myfile.txt 1.2.1.2: checking ci for branches

2003-01-24  Johannes Franken  <jfranken@tp>

        * myfile.txt 1.2.1.1: test branch

2003-01-10  Johannes Franken  <jfranken@tp>

        * myfile.txt 1.2: addition Nr. 2

        * myfile.txt 1.1: New file.
```

Mehr zum Thema:
Weitere Optionen: siehe rcs2log(1) manpage

rcsdiff

`rcsdiff` gibt die Unterschiede zwischen zwei verschiedenen Versionen einer Datei aus.

Beispiele:

- Vergleich der Arbeitsdatei mit der aktuellsten Version aus der Archivdatei:

```
$ rcsdiff -q myfile.txt,v
$
```

  Ergebnis: Keine Unterschiede - die Arbeitsdatei wurde seit dem Ein- oder Auschecken nicht verändert.
- Vergleich der Arbeitsdatei mit einer älteren Version aus der Archivdatei:

```
$ echo 'and Nr. 3' >> myfile.txt
$ rcsdiff -q -r1.1 myfile.txt,v
2a3,4
> + Nr. 2
> and Nr. 3
$
```

  Ergebnis: Im Gegensatz zu der Version 1.1. enthält die Arbeitsdatei nach der zweiten Zeile die neuen Zeilen 3 bis 4 mit dem o.g. Inhalt.
- Vergleich von zwei in der Archivdatei enthaltenen Versionen:

```
$ rcsdiff -q -r1.1 -r1.2 myfile.txt,v
2a3
> + Nr. 2
$
```

  Ergebnis: Der Unterschied zwischen den Versionen 1.1 und 1.2 besteht darin, dass in Version 1.2 nach der zweiten eine dritte Zeile angehängt wurde mit dem Inhalt **`+ Nr. 3`**.

Mehr zum Thema:
Weitere Optionen: siehe rcsdiff(1) manpage

Ausblick: CVS

Das Concurrent Versions System (CVS) erweitert RCS um folgende Fähigkeiten:

- Unterverzeichnisse
- Server-Schnitstelle (Pipes oder TCP-Ports)
- Automatische Ausführung von Scripten beim Einchecken (automated build system, ABS)

Mehr zum Thema:
Weitere Optionen: siehe cvs(1) manpage