

# RCS-Einführung, Tipps und Tricks

Version 1.12 — 17.6.2008

© 2001–2008 T. Birnthaler, OSTC GmbH

Die Informationen in diesem Skript wurden mit größter Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Der Autor übernimmt keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene fehlerhafte Angaben und deren Folgen.

Alle Rechte vorbehalten einschließlich Vervielfältigung, Übersetzung, Mikroverfilmung sowie Einspeicherung und Verarbeitung in elektronischen Systemen.

Für Dokumente und Programme unter dem Copyright der OSTC GmbH gilt:

- Dürfen heruntergeladen und im privaten Bereich frei verwendet werden.
- Kommerzielle Nutzung bedarf der vorherigen Zustimmung durch die OSTC GmbH.
- Titelseite und Copyright-Hinweise darin dürfen nicht verändert werden.

Hinweise auf inhaltliche Fehler, Schreibfehler und unklare Formulierungen sowie Ergänzungen, Kommentare, Wünsche und Fragen können Sie gerne an den Autor richten:

OSTC Open Source Training and Consulting GmbH  
Thomas Birnthaler  
eMail: [tb@ostc.de](mailto:tb@ostc.de)  
Web: [www.ostc.de](http://www.ostc.de)

## Inhaltsverzeichnis

<b>1 Einführung</b>	<b>3</b>
<b>2 Übersicht</b>	<b>4</b>
<b>3 Typischer Ablauf</b>	<b>5</b>
<b>4 Beschreibung</b>	<b>7</b>
<b>5 Kommandos</b>	<b>9</b>
<b>6 Spezielle Kommandos</b>	<b>11</b>
<b>7 Zusammenfassung</b>	<b>13</b>
7.1 Kommandos . . . . .	13
7.2 Optionen . . . . .	14
7.3 Schlüsselworte . . . . .	14
7.4 Kommentarkennzeichen . . . . .	14
7.5 Zustand (Status) . . . . .	14
7.6 Umgebungsvariablen . . . . .	15
7.7 Manualseiten . . . . .	15

# 1 Einführung

**Programmentwicklung, Systemkonfiguration** und **Dokumenterstellung** erfolgt in der Regel schrittweise über einen längeren Zeitraum hinweg, wobei trotz ständiger Weiterentwicklung alte Versionen/Varianten verfügbar bleiben sollen. In **großen Projekten** ist dies eine **komplexe Aufgabe**, insbesondere wenn **mehrere Personen** an der Erstellung, Administration und Weiterentwicklung beteiligt sind.

Das RCS (**Revision Control System**) dient ganz allgemein zur Verwaltung der Versionen beliebiger (ASCII-)Dateien (insbesondere Quelltexten von Programmen), die einer **Entwicklungsgeschichte** unterliegen. Die übliche Entwicklungsgeschichte eines Programm-Quelltextes ist z.B.:

- Erste (Null-)Version 1.0
- Wiederholte Erweiterungen und Fehlerkorrekturen
- Erste (Test-)Version 1.18
- Wiederholte Erweiterungen und Fehlerkorrekturen
- Erste ausgelieferte (Produktions-)Version 1.25
- ...
- Letzte ausgelieferte (Produktions-)Version 3.14
- ...

Die **Nullversion** 1.0 ist zwar lauffähig, aber noch unvollständig und aller Wahrscheinlichkeit nach stark fehlerbehaftet. Die erste ausgelieferte (Produktions-)Version 1.25 entspricht noch nicht den Erwartungen der Anwender. Auch die letztendlich ausgelieferte (Produktions-)Version 3.14 wird während ihrer Anwendung wahrscheinlich Fehler zeigen oder aufgrund von Anwenderwünschen erweitert werden müssen.

Per RCS verwaltete **Archivdateien** können an einer **zentralen Stelle** des Dateibaumes, **lokal** in einem Unterverzeichnis namens `RCS` (*groß schreiben!*) oder direkt neben den zu verwaltenden Dateien stehen (*unübersichtlich!*).

Sie haben den gleichen Namen wie die Originaldatei + die **Endung** `,v` (version) und sind **schreibgeschützt**. Sie sind **textbasiert** und können mit einem Editor angesehen (*aber nicht verändert!*) werden (das Format ist in der Manual-Seite `rcsfile` beschrieben).

Für eine **Historie der Systemadministration** bietet es sich z.B. an, im Verzeichnis `/etc` und seinen Unterverzeichnissen jeweils ein Verzeichnis `RCS` einzurichten, um die Versionen der dort abgelegten Konfigurationsdateien zu verwalten.

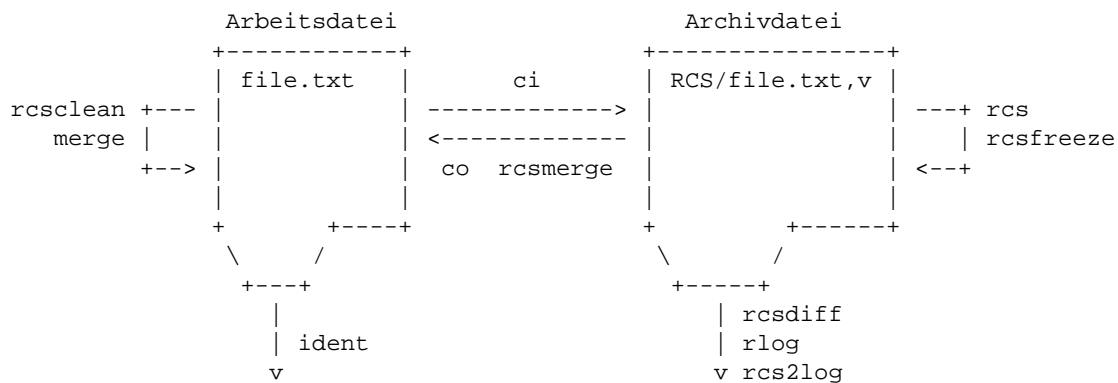
Bei Verwendung von RCS erübrigt es sich auch, **veraltete Quelltext-Teile** (z.B. zu Debugging-Zwecken) auskommentiert im Quelltext stehen zu lassen, da sie sich aus den eingechekkten alten Versionen wieder rekonstruieren lassen.

RCS wurde in den 1980ern von Walter F. Tichy an der Purdue University entwickelt und ist heute Teil des GNU-Projekts. Einen Artikel zu RCS vom Autor selbst findet man in *Software-Practice & Experience* 15, 7 (July 1985), 637-654.

Eine **netzwerkfähige Erweiterung** von RCS ist **CVS (Concurrent Versions System)**, das immer noch das Dateiformat von RCS verwendet. Daher ist es recht einfach möglich, ein mit RCS verwaltetes Projekt nach CVS zu übernehmen. Eine Nachfolger von CVS ist **SVN (Subversion)**. Ein Vorläufer von RCS ist **SCCS (Source Code Control System)**.

## 2 Übersicht

RCS erstellt für jede Arbeitsdatei eine **Archivdatei**, in der ihre Entwicklung in Form von Deltas gespeichert wird. Folgende Grafik bietet eine Übersicht über die von RCS zur Verfügung gestellten **Kommandos**:



Zur Koordination und Verwaltung der Entwicklungsgeschichten einer oder mehrere Dateien bietet RCS folgende Fähigkeiten:

- **Speichern und Wiederherstellen vielfacher Versionen** einer Datei. RCS sichert alle alten Versionen auf platzsparende Weise. Alte Versionen können über **Kennungen** der Form

- ▷ Versionsnummer
- ▷ Symbolischer Name
- ▷ Datum + Uhrzeit
- ▷ Autor
- ▷ Status

wiederhergestellt werden.

- Erhalten der kompletten **Historie von Veränderungen**. RCS protokolliert alle Veränderungen automatisch mit. Außer dem Quelltext jeder Version speichert RCS

- ▷ Autor

- ▷ Datum + Uhrzeit des Eincheckens
- ▷ Eine die Veränderung zusammenfassende Notiz

Dies erleichtert es, herauszufinden, wann ein Quelltext von wem wie bearbeitet wurde, ohne die Quelltexte zu durchsuchen oder Kollegen zu befragen.

- **Lösen von Konflikten** beim Zugriff. Wenn zwei Programmierer die selbe Version einer Datei bearbeiten wollen, macht RCS darauf aufmerksam und verhindert, dass eine Bearbeitung die andere versehentlich zerstört. Im Extremfall werden sogar gleichzeitige Änderungen an verschiedenen Stellen einer Datei ermöglicht.
- Erstellen eines **Baums aller Versionen**. RCS kann für jede Quelltext-Datei **getrennte Entwicklungslinien** unterhalten. Es speichert eine Baumstruktur, die die Abstammungen und Beziehungen der Versionen darstellt.
- **Verschmelzen von Versionen**. Zwei unterschiedliche Entwicklungslinien einer Quelltext-Datei können verschmolzen werden. Falls die Veränderungen der zu verschmelzenden Versionen die gleichen Stellen des Quelltextes betreffen, macht RCS darauf aufmerksam.
- **Überblick** über ausgelieferte Versionen und Konfigurationen. Es ist möglich, bestimmten Versionen der Quelltext-Dateien gemeinsam einen **symbolische Namen** zu geben und sie mit einem **Status** zu kennzeichnen:

Name	Status	Bedeutung
Exp	Experimental	Testversion (Standard)
Stab	Stable	Stabil
Rel	Released	Freigegeben

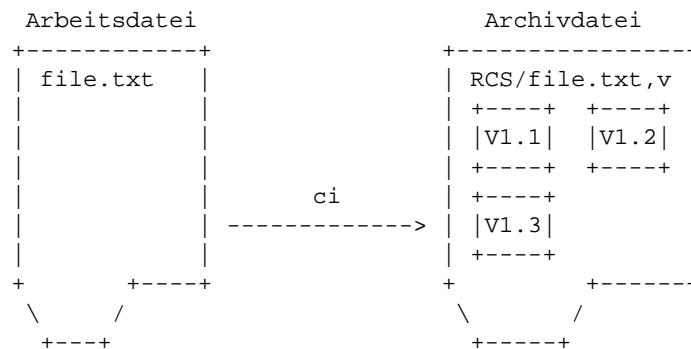
- **Identifizieren jeder Version** durch Name, Versionsnummer, Zeitpunkt der Erstellung, Autor, etc. Diese Identifikation ist wie ein Stempel, der an **beliebig wählbaren Stellen** des Quelltextes einer Version eingefügt werden kann.
- **Minimierung des Speicherplatzbedarfs**. RCS benötigt wenig zusätzlichen Speicherplatz für die einzelnen Versionen, da es nur die letzte Version vollständig und die vorherigen Versionen in Form von **Differenzen** speichert (per Kommando `diff`, d.h. der Unterschied wird **zeilenweise** ermittelt). Falls Versionen gelöscht werden, werden die entsprechenden Deltas komprimiert.

### 3 Typischer Ablauf

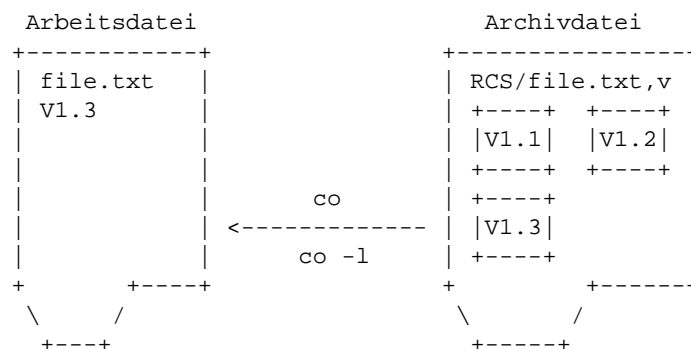
RCS ist *nicht* dazu gedacht, jede kleine Änderung zu protokollieren, sondern es soll durch den Anwender gesteuert die **Entwicklungsschritte** aufzeichnen. Dies ergibt sich schon aus dem Schema der Quelltextverwaltung unter RCS:

1. Eine erste **Nullversion** des Quelltextes (und sei es in Form einer leeren Datei) eines Programms liegt vor (z.B. `f00.c`).

2. Dieser Quelltext wird in einem (allen Entwicklern zugänglichen) **Projekt-Verzeichnis** abgelegt (z.B. `/usr/local/projects/foo/`).
3. In diesem Verzeichnis wird ein **RCS-Unterverzeichnis** namens `RCS` (*groß!*) angelegt und mit der korrekten Besitzergruppe und korrekten Rechten versehen (`chgrp project RCS + chmod 2770`).
4. Die am Projekt beteiligten Entwickler erzeugen **symbolische Links** von ihrem jeweiligen lokalen Projekt-Arbeitsverzeichnis auf das gemeinsame RCS-Verzeichnis dieses Projekts (`ln -s ...`).
5. Um eine Quelltext-Datei `foo.c` an RCS zu übergeben, verwendet man den **Check-in Befehl** `ci foo.c`. RCS erzeugt eine **Versionsdatei** `foo.c,v`, legt `foo.c` darin als Version 1.1 ab, und löscht `foo.c`. Außerdem wird nach einer **Beschreibung** für die Datei gefragt, die ebenfalls in der Versionsdatei abgelegt wird.



6. Zum Bearbeiten einer Quelltext-Datei muss diese mit dem **Check-out Befehl** `co -l filename.c` aus ihrer Versionsdatei **ausgecheckt** werden. D.h. ihre **aktuelle Version** wird zur Verfügung gestellt und die Versionsdatei für die anderen Entwickler **gesperrt** (`-l = lock`).



7. Eine Quelltext-Datei kann auch so ausgecheckt werden, dass sie nicht verändert werden darf (`-l` weglassen). Dies funktioniert sogar, während die Datei gesperrt ist.
8. Nach dem Bearbeiten der Quelltext-Datei wird sie wieder in ihre Versionsdatei **eingchecked**. Dies kann nur derjenige Entwickler, der sie auch ausgecheckt hat. Die **Versionsnummer** wird erhöht, **Datum + Uhrzeit** und der **Bearbeiter** aufgezeichnet und ein

**Kommentar** für den Grund der Änderung abgefragt, der dieser Version als Information hinzugefügt wird. Anschließend ist die Versionsdatei wieder für andere Entwickler freigegeben.

- Über ihre **Versionsnummer X.Y** kann jede der in einer Versionsdatei vorhandenen Versionen einer Quelltext-Datei wieder hergestellt werden (`-rX.Y` = revision).

## 4 Beschreibung

- Die Form der von RCS verwendeten Versionsnummer ist (je nachdem, ob eine **lineare** oder **hierarchische** Folge von Versionen verwaltet wird):

```
Release.Level # oder
Release.Level.Branch.Sequence # oder
Release.Level.Branch.Sequence.Branch.Sequence # oder
...
```

- Die **erste Version** einer Quelltext-Datei bekommt die Versionsnummer **1.1**. Jedes Aus- und wieder Einchecken führt zu einer Erhöhung des **Level** um 1. Wahlweise kann auch (bei größeren Änderungen oder dem Abschluss eines Entwicklungsschrittes) der **Release** erhöht und der Level wieder auf 1 zurückgesetzt werden.
- Die beiden Stellen **Branch** und **Sequence** dienen als **Reserve** und können verwendet werden, wenn an einer eigentlich abgeschlossenen älteren Version **kleine Änderungen** vorzunehmen sind, ohne diese bis zur neuesten Version durchzuziehen. Oder wenn zwei oder mehr Versionen der gleichen Quelltext-Datei gleichzeitig zu pflegen sind (z.B. ausgelieferte Version und aktuelle Entwicklungsversion).
- RCS kennt folgende **Kommandos**:

Kommando	Bedeutung
ci	Speichert neue Version in einer Archivdatei ( <b>check in</b> )
co	Entnimmt eine Version aus einer Archivdatei ( <b>check out</b> )
rsc	Erstellt neue Archivdateien oder ändert ihre Attribute
rcsclean	Entfernt ausgecheckte Dateien, die unverändert sind
rsc2log	Änderungshistorie aus Archivdatei extrahieren
rcsdiff	Vergleicht RCS-Versionen ( <b>difference</b> )
rcsfreeze	Vergibt einen symbolischen Versionsnamen für Archivdateien, die eine gültige Konfiguration bilden
rscmerge	Kombiniert die Veränderungen zweier Versionen
rscintro	Einführung in RCS
rscfile	Beschreibung des Archivdateiformats
ident	Sucht in Dateien nach RCS-Einträgen der Form <code>\$Id...\$</code> und <code>\$Log...\$</code> und gibt sie aus
rlog	Gibt Informationen über Archivdateien aus

- Folgende **RCS-Schlüsselworte** (begrenzt von zwei Dollarzeichen) können in Quelltext-Dateien als **Platzhalter** eingetragen werden. Sie werden bei jedem Check-out durch

den **aktuellen Wert** ersetzt. Dadurch werden wichtige Informationen zur Version (Quellname, Versionsnummer, Bearbeiter, Datum + Uhrzeit, ...) *automatisch* im Quelltext eingetragen:

Keyword	Bedeutung
\$Source\$	Name der Archivdatei (mit Pfad)
\$RCSfile\$	Name der Archivdatei (ohne Pfad)
\$Revision\$	Versionsnummer
\$Date\$	Datum + Uhrzeit (GMT) des Eincheckens
\$Author\$	Login-Name des eincheckenden Anwenders
\$State\$	Zustand einer Quelltext-Datei (Exp, Stab, Rel)
\$Locker\$	Login-Name des Anwenders, der die Version sperrt
\$Header\$	Standard-Kopf = alle o.g. Keywords außer \$RCSfile\$ (Source Revision Date Author State Locker)
\$Id\$	Standard-Kopf = alle o.g. Keywords außer \$Source\$ (RCSfile Revision Date Author State Locker) (z.B. rcs.tex,v 1.3 2005/05/31 16:42:27 tsbirn Exp tsbirn)
\$Log\$	Liste der Check-in Kommentare
\$Name\$	Symbolischer Name mit dem ausgecheckt wurde

Diese Schlüsselworte werden häufig **als Kommentar** im Quelltext eingetragen, damit sie nicht als normaler Dateibestandteil mißinterpretiert werden. Übliche einen Kommentar einleitende oder umrahmenden Zeichen sind:

Zeichen	Bedeutung
#	Shell-, Perl-, Awk-Skripte, UNIX/Linux-Tools und -Konfigurationsdateien
//	C++-Programme
/*...*/	C-Programme
%	LaTeX-Dokumente
"	Vim-Konfiguration
--	SQL-Anweisungen

RCS erkennt automatisch an der **Dateiendung**, welche Art der Kommentierung zu verwenden ist.

Möglich ist aber auch das „**Einbetten**“ von RCS-Schlüsselworten in Zeichenketten/Strings, um sie direkt im Quellcode eines Programms/Skripts/Konfigurationsfiles/Dokuments verfügbar zu machen (z.B. Versionsinformationen).

```
static char rcsid[] = "$Id: rcs.tex,v 1.17 2009/04/04 10:56:32 tsbirn Exp tsbirn $";
printf("Version $Id: rcs.tex,v 1.17 2009/04/04 10:56:32 tsbirn Exp tsbirn $\n");
```

- Der **Zustand** einer Quelltext-Datei \$State\$ wird durch die Option `-s (state)` von `rcs` oder `ci` definiert. Es stehen drei verschiedene Zustände zur Verfügung, nämlich:

Name	Status	Bedeutung
Exp	Experimental	Testversion (Standard)
Stab	Stable	Stabil
Rel	Released	Freigegeben

- Die beiden Schlüsselwörter `$Id$` und `$Log$` sollten in jeder Quelltext-Datei am Dateianfang bzw. am Dateiende vorhanden sein, damit die **Versionsinformation** und die **Änderungshistorie** direkt im Quelltext lesbar vorhanden sind (in der **Archivdatei** sind diese Informationen natürlich ebenfalls vorhanden).

## 5 Kommandos

Zum Abschluss dieser Beschreibung von RCS werden die wichtigsten Kommandos in einem **Musterbeispiel** zusammenhängend verwendet. Im Prinzip können bei jedem Kommando **mehrere Dateien** angegeben werden, die Aufrufe verwenden aber der Einfachheit halber immer nur eine Datei.

Die gerade ausgecheckte (und möglicherweise bearbeitete) Version wird **aktuelle Version** genannt, die zuletzt in RCS eingechekkte Version wird **letzte Version** genannt.

- Bei allen RCS-Kommandos kann der Schalter `-rN.M` angegeben werden, wenn nicht die letzte Version, sondern die **Version N.M** angesprochen werden soll (`-r` = revision).
- Zu verwalten sei die Quelltext-Datei `foo.c` eines **Programms** namens `foo` mit folgendem Inhalt als 1. Version:

```
/* $Id: rcs.tex,v 1.17 2009/04/04 10:56:32 tsbirn Exp tsbirn $ */
#include <stdio.h>

int
main(int argc, char* argv[])
{
    printf("Hallo Welt!\n");
    exit(0);
}
/* $Log: rcs.tex,v $
 * Revision 1.17  2009/04/04 10:56:32  tsbirn
 * *** empty log message ***
 * */
```

- Im ersten Schritt das **zentrale Projektverzeichnis** `foo` und ein **RCS-Verzeichnis** darin (Gesamtpfad `/usr/local/projects/foo/RCS`) anlegen und einer Projektgruppe `project` mit geeigneten Zugriffsrechten zuordnen:

```
cd /usr/local
mkdir projects
cd projects
mkdir -p foo/RCS          # -p = path
chgrp -R project foo
chmod -R 2770 foo
```

- Dann im **persönlichen Projekt-Verzeichnis** des Benutzers `USER` einen **symbolischer Link** auf dieses zentrale RCS-Verzeichnis erstellen (`USER` ist irgendein Benutzername):

```
cd ~USER
mkdir foo
cd foo
ln -s /usr/local/projects/foo/RCS .
```

- Durch den **Check-in Befehl** `ci` wird aus `foo.c` die Archivdatei `foo.c,v` mit der Versionsnummer 1.1 im Unterverzeichnis `RCS` erzeugt, `foo.c` gelöscht und eine **Beschreibung** des Programms abgefragt.

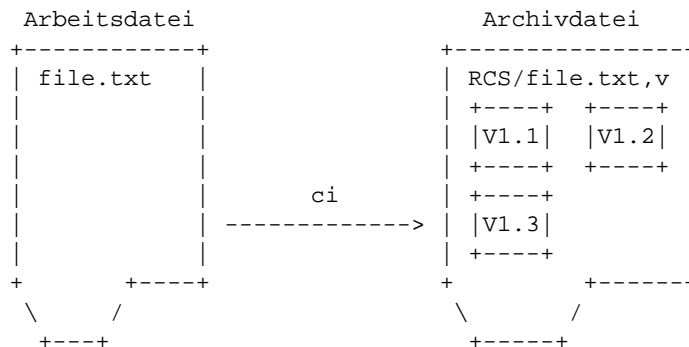
```
vi foo.c      # Mit obigem Inhalt oder
touch foo.c   # leer anlegen
ci foo.c      # Einchecken (Beschreibung mit "." auf extra Zeile beenden)
```

- Um aus der letzten Version von `foo.c,v` wieder die Quelltext-Datei `foo.c` herzustellen, verwendet man den **Check-out Befehl** `co` (standardmäßig read-only):

```
co foo.c
```

- Um eine bestimmte Version mit Versionsnummer 1.34 oder symbolischem Namen `NAME` von `foo.c,v` als Arbeitsdatei `foo.c` **auszuchecken**, verwendet man den **Check-out Befehl** `co` mit Option `-r` (= release):

```
co -r1.34 foo.c # Release 1.34 (Read-only)
co -rNAME foo.c # Release NAME (Read-only)
co -nNAME RCS/* # Alle Vers. mit Namen NAME auschecken
```



- Will man die Datei `foo.c` **ändern**, muss beim Auschecken die **Sperre** mit der Option `-l` (**lock**) eingeschaltet werden:

```
co -l foo.c # Read-write (lock)
```

Dies funktioniert nur, wenn **kein anderer Benutzer** einen Lock auf diese Datei gesetzt hat. Ist das der Fall, bekommt man dessen Namen angezeigt und kann ihn dann z.B. bitten, den Lock freizugeben.

Alternativ kann der Lock explizit „**gebrochen**“ werden (der Halter des Locks erhält eine entsprechende Mail zugesandt):

```
rcs -u foo.c # unlock
```

- **Alle** zu einem Projekt gehörenden Archivdateien **auschecken**:

```
co RCS/* # Read-only
co -l RCS/* # Read-write (lock)
```

- **Unterschiede** zwischen Versionen einer Datei **anzeigen**:

```
rcsdiff foo.c # Aktuelle und letzte eingetragene Version
rcsdiff -r1.15 foo.c # Aktuelle Version und Version 1.15
rcsdiff -r1.12 -r1.20 foo.c # Version 1.12 und 1.20
```

Die Zeilen der alten Version sind mit dem **Präfix** „<“ gekennzeichnet, die der neuen Version mit dem **Präfix** „>“.

- Aktuelle Version (ausgecheckt) **aller** zu einem Projekt gehörenden Archivdateien mit der letzten eingetragenen Version **vergleichen**:

```
rcsdiff RCS/*
```

- Nach dem Abschluss der Änderungen wird `foo.c` wieder in RCS **eingetrag**:

```
ci foo.c # Originaldatei wird entfernt
ci -u foo.c # Originaldatei bleibt read-only stehen (unlock)
ci -l foo.c # Originaldatei bleibt read-writable stehen (lock)
```

Die Versionsnummer hat sich dadurch auf 1.2 erhöht. Der **Zeitpunkt**, der **Bearbeiter** und die Änderungen sind in der Versionsdatei festgehalten, ein **Kommentar zur Änderung** wird interaktiv abgefragt (*mit **einzelnen Punkt** „.“ auf einer Zeile für sich abschließen*) und die Datei `foo.c` wird anschließend evtl. gelöscht.

- Soll das Programm `foo` erzeugt werden, ist die Quelltext-Datei aus RCS zu extrahieren und anschließend zu übersetzen:

```
co foo.c
cc foo.c -o foo # oder
make foo
```

## 6 Spezielle Kommandos

- Die **erste (leere) Version** einer mit RCS zu verarbeitenden Datei kann auch so angelegt werden (`-i` = initialize), eine **Beschreibung** dazu wird interaktiv abgefragt und muss mit einem **einzelnen Punkt** (auf einer Zeile für sich) abgeschlossen werden:

```
rcs -i foo.c # Einchecken (Beschreibung mit „.“ auf extra Zeile beenden)
```

- Eine Version **nachträglich sperren** (`-l = lock`) bzw. **entsperren** (`-u = unlock`) mit:

```
rcs -l foo.c           # Aktuelle Version sperren
rcs -l1.23 foo.c      # Version 1.23 sperren
rcs -u foo.c          # Aktuelle Version entsperren
rcs -u1.23 foo.c      # Version 1.23 entsperren
```

Ein bereits bestehender Lock wird von `-u` (= unlock) explizit „gebrochen“ (und dem Halter des Locks eine entsprechende Mail zugesandt).

- Falls `ci` die Meldung

```
ci error: no lock set by USER
```

ausgibt, wurde versucht eine Datei einzuchecken, obwohl sie beim Auschecken nicht gesperrt worden war. *In diesem Falle nicht mit einer Sperre auschecken, da dies die neuen Änderungen überschreiben würde!* Stattdessen zunächst

```
rcs -l foo.c
```

anwenden, um die letzte Version zu **sperren** (`-l = lock`). Allerdings funktioniert dies nur, wenn nicht schon jemand anderes zuvorgekommen ist (vorher mit `rcsdiff` vergleichen, ob ein anderer Änderungen gemacht hat).

- Ist einem jemand zuvorgekommen und hat ebenfalls Änderungen an der Datei durchgeführt, so muss man *sehr sorgfältig* die beiden Änderungen **manuell zusammenfügen** („**mergen**“). Oder das Kommando `rcsmerge` einsetzen (`-p = preserve` verändert nicht die aktuelle Version, sondern gibt das Ergebnis auf der Kommandozeile aus):

```
rcsmerge foo.c           # Letzte + aktuelle Version m.
rcsmerge -p -r1.2 > foo.new.c # Letzte + 1.2-Version mergen
rcsmerge -p -r1.2 -r2.4 foo.c > foo.new.c # 1.2- + 2.4-Version mergen
```

- Einen **neuen Release 2.1** (der Level wird immer auf 1 zurückgesetzt) erzwingen (sofern die jüngste Version eine kleinere Versionsnummer hat):

```
ci -r2 foo.c           # oder
ci -r2.1 foo.c
```

- Einen **Zustand Rel** (`-s = state`) für eine Version vergeben (alternativ `Stab` oder `Exp`):

```
rcs -sRel foo.c         # Aktuelle Version mit Zustand "Rel" versehen
rcs -sRel:1.24 foo.c    # Version 1.24 mit Zustand "Rel" versehen
rcs -sRel RCS/*         # Alle aktuellen Vers. mit Zustand "Rel" versehen
```

- Einen **symbolischen Namen** `NAME` (`-n = name`) für eine bestimmte Version vergeben (Doppelpunkt nicht vergessen!):

```

rcs -nNAME foo.c          # Aktuelle Version mit Namen NAME versehen
rcs -nNAME:1.24 foo.c     # Version 1.24 mit Namen NAME versehen
rcs -nNAME: RCS/*        # Alle aktuellen Vers. mit Namen NAME versehen

```

- Einen **Branch** anlegen (z.B. zur Version 1.2, die neue Version erhält die Nummer 1.2.1.1):

```
ci -r1.2.1 foo.c
```

- Eine per RCS verwaltete Datei **vollständig löschen** (-f = force):

```
rm -f foo.c RCS/foo.c,v
```

- Versionen einer per RCS verwalteten Datei **löschen** (-o = outdate):

```

rcs -o foo.c              # Letzte eingecheckte Version
rcs -o1.21 foo.c          # Version 1.21
rcs -o1.41:1.45 foo.c     # Version 1.41 bis 1.45

```

- Alle zu einem Projekt gehörenden Dateien „**aufräumen**“, d.h. nur die Archivdateien und die gelockten Dateien stehen lassen (ungelockte Dateien entfernen):

```
rcsclean *
```

- RCS-Informationen aus Archivdateien **extrahieren und ausgeben** (erkennbar an dem Präfixen \$Id...\$ und \$Log...\$, dürfen auch Binärdateien sein):

```
ident FILE...
```

- Informationen zu allen per RCS verwalteten Dateien **ausgeben**:

```
rlog RCS/*
```

## 7 Zusammenfassung

### 7.1 Kommandos

Kommando	Bedeutung
ci	Speichert neue Version in einer Archivdatei ( <b>check in</b> )
co	Entnimmt eine Version aus einer Archivdatei ( <b>check out</b> )
rcs	Erstellt neue Archivdateien oder ändert ihre Attribute
rcsclean	Entfernt ausgecheckte Dateien, die unverändert sind
racs2log	Änderungshistorie aus Archivdatei extrahieren
rcsdiff	Vergleicht RCS-Versionen ( <b>difference</b> )
rcsfreeze	Vergibt einen symbolischen Versionsnamen für Archivdateien, die eine gültige Konfiguration bilden
racsmerge	Kombiniert die Veränderungen zweier Versionen
ident	Sucht in Dateien nach RCS-Einträgen der Form \$Id...\$ und \$Log...\$ und gibt sie aus
rlog	Gibt Informationen über Archivdateien aus

## 7.2 Optionen

Option	Name	Bedeutung
-u	unlock	Datei entsperren
-l	lock	Datei entsperren
-o <i>N.M</i>	outdate	Version entfernen
-p	preserve	Versionen erhalten
-i	initialize	Datei initialisieren
-r <i>N.M</i>	revision	Version auswählen
-s <i>STATE</i>	state	Status auswählen
-n <i>NAME</i>	name	Versionsname auswählen

## 7.3 Schlüsselworte

Keyword	Bedeutung
\$Source\$	Name der Archivdatei mit Pfad
\$RCSfile\$	Name der Archivdatei ohne Pfad
\$Revision\$	Versionsnummer
\$Date\$	Datum + Uhrzeit (GMT) des Eincheckens
\$Author\$	Login-Name des eincheckenden Anwenders
\$State\$	Zustand einer Quelltext-Datei
\$Locker\$	Login-Name des Anwenders, der die Version sperrt
\$Header\$	Standard-Kopf = alle o.g. Keywords außer \$RCSfile\$ (Source Revision Date Author State Locker)
\$Id\$	Standard-Kopf = alle o.g. Keywords außer \$Source\$ (RCSfile Revision Date Author State Locker) (z.B. rcs.tex,v 1.3 2005/05/31 16:42:27 tsbirn Exp tsbirn)
\$Log\$	Liste der Check-in Kommentare
\$Name\$	Symbolischer Name mit dem ausgecheckt wurde

## 7.4 Kommentarkennzeichen

Zeichen	Bedeutung
#	Shell-, Perl-, Awk-Skripte, UNIX/Linux-Tools und -Konfigurationsdateien
//	C++-Programme
/*...*/	C-Programme
%	LaTeX-Dokumente
"	Vim-Konfiguration
--	SQL-Anweisungen

## 7.5 Zustand (Status)

Name	Status	Bedeutung
Exp	Experimental	Testversion (Standard)
Stab	Stable	Stabil
Rel	Released	Freigegeben

## 7.6 Umgebungsvariablen

Variable	Bedeutung
RCSINIT	Standard-Optionen für alle RCS-Kommandos vorgeben Standard-Verzeichnis für temporäre Dateien (alternativ TMP und TEMP bzw. /tmp).
TMPDIR	

## 7.7 Manualseiten

Manualseite	Inhalt
rcsintro	Einführung in RCS
rcsfile	Beschreibung des Archivdateiformats
ci	Speichert neue Version in einer Archivdatei ( <b>check in</b> ) Entnimmt eine Version aus einer Archivdatei ( <b>check out</b> )
co	
rcs	Erstellt neue Archivdateien oder ändert ihre Attribute Entfernt ausgecheckte Dateien, die unverändert sind
rcsclean	
rcs2log	Änderungshistorie aus Archivdatei extrahieren Vergleicht RCS-Versionen ( <b>difference</b> ) Vergibt einen symbolischen Versionsnamen für Archivdateien, die eine gültige Konfiguration bilden
rcsdiff	
rcsfreeze	
rcsmerge	Kombiniert die Veränderungen zweier Versionen
ident	Sucht in Dateien nach RCS-Einträgen der Form \$Id...\$ und \$Log...\$ und gibt sie aus
rlog	