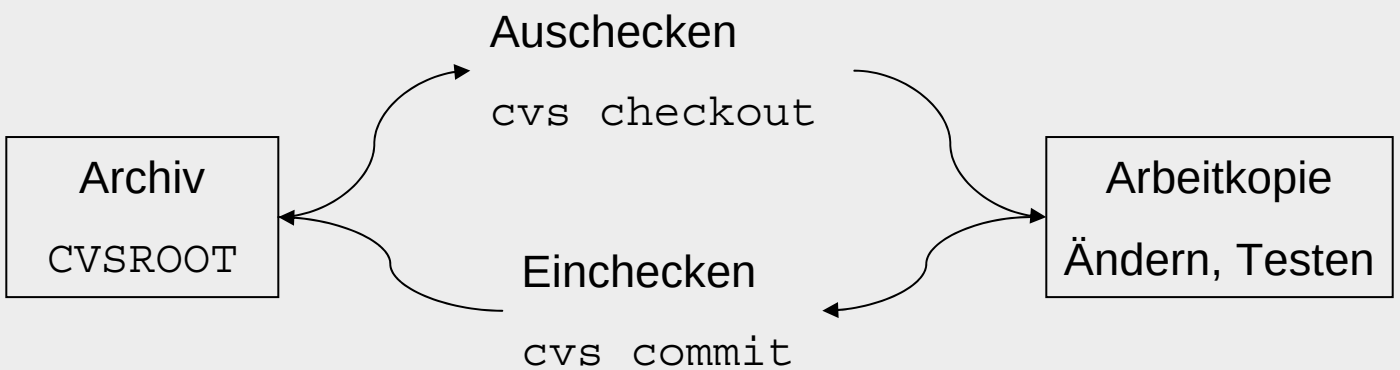
CVS - Concurrent Versions System

Programm zur Versionskontrolle von Quellcode: Alle Versionsstände sind gespeichert. Die Änderungen sind nachvollziehbar. Alte Versionsstände lassen sich zurückholen.

Mehrere Entwickler können an einem Projekt gleichzeitig arbeiten.

Idee: Aus Archiv holen - Modifizieren - Neue Version in Archiv speichern



CVS - Begriffe

• „Archiv-Datenbank“, Repository, durch `$CVSROOT` Umgebungsvariable spezifiziert.

• Revision: Version einer einzelnen Datei. Fortlaufende Nummer, vom System vergeben.

• Release: Symbolischer Versionsname von mehreren zusammengehörenden Dateien (Projekt).

• Arbeitskopie: Lokale Kopie, die der Entwickler ändern und testen kann. Jeder Entwickler kann eine oder mehrere Arbeitskopien haben.

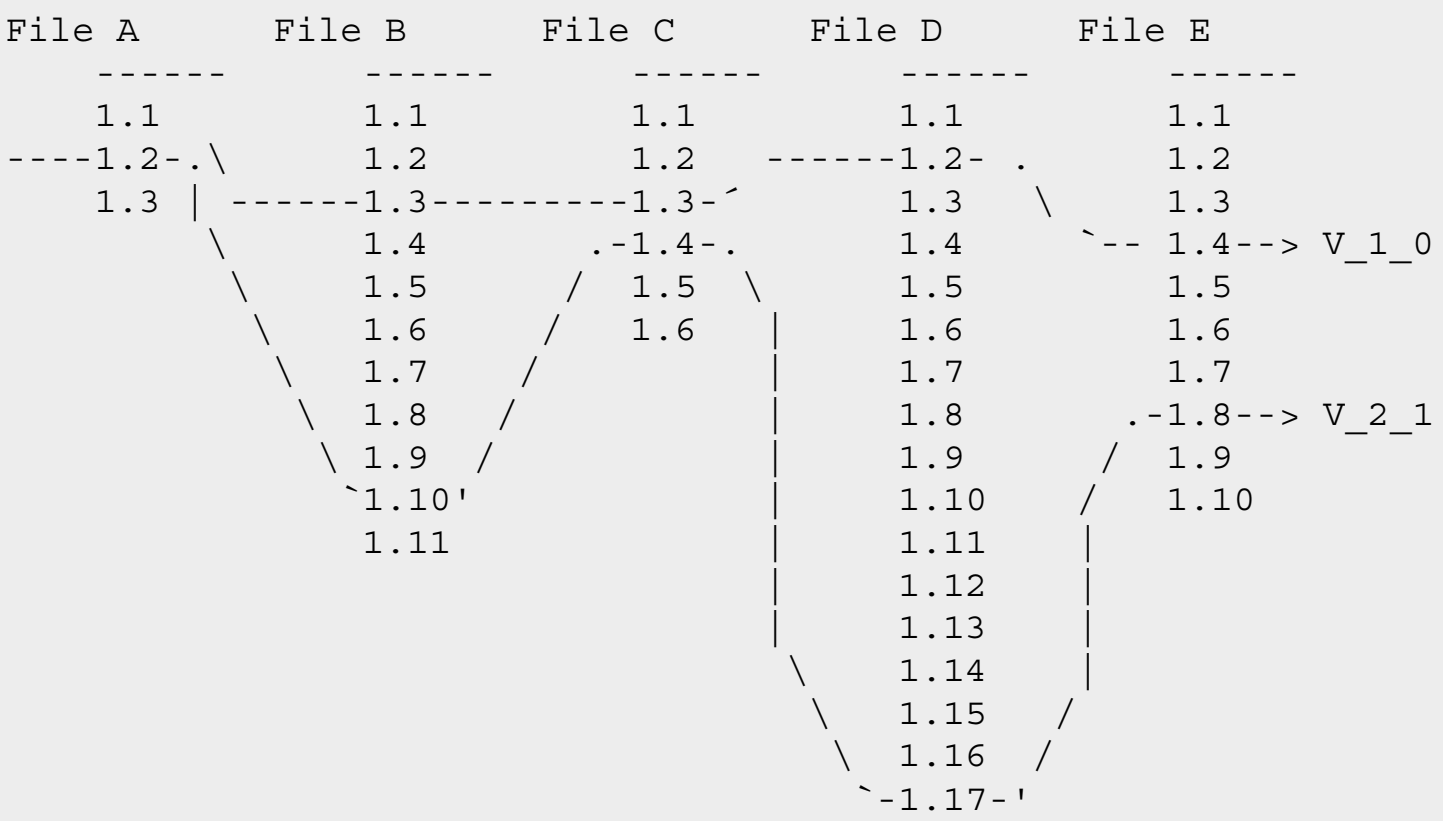
• Checkout: „Kopieren“, eine Arbeitskopie aus dem Archiv herausholen.

• Commit: Änderungen an der Arbeitskopie dem Repository „mitteilen“.

• Aktualisierung, Update: Änderungen (z.B. von anderen Entwicklern) in die eigene Arbeitskopie aufnehmen.

• Konflikt - Wenn zwei Entwickler am gleichen Teil einer Datei Veränderungen vornehmen und einchecken. CVS bemerkt solche Konflikte und teilt sie den Entwicklern mit.

Revisionen und Releases



CVS - Aufruf von der Kommandozeile

Befehle für Login Skript

```
[6]user@workstation> setenv CVSROOT :pserver:user@ws-cvs:/path/to/repository
```

Befehle für CVS Sitzung

```
[9]user@workstation> cvs login
(Logging in to user@ws-cvs)
CVS password:

[10]user@workstation> cvs xxxx
....
[11]user@workstation> cvs yyyy


[25]user@workstation> cvs logout
(Logging out of user@ws-cvs)
```

CVS-Befehl checkout

Syntax: `cvs checkout Projektname`
Extrahiert ein Projekt (Modul) aus dem Archiv und legt eine lokale Arbeitskopie an. Achtung: Das Verzeichnis Projektname samt aller Unterverzeichnisse wird lokal angelegt! Wenn es ein lokales Verzeichnis mit gleichem Namen gibt, werden keine Dateien überschrieben - schon bestehende Dateien und Unterverzeichnisse mit gleichem Namen werden aber bemängelt!

```
> cvs checkout source
cvs checkout: in directory source:
cvs server: Updating source
U source/header.h
U source/mod1.c
U source/mod2.c
U source/mod3.c
U source/main.c
U source/func.c
U source/dbgfunc.c
[..].
>
```

CVS-Befehl commit:

Syntax: `cvs commit -m "Änderungskommentar" [Dateiname]`
Überträgt Änderungen ins Archiv. Falls keine Kommentar angegeben wurde, wird automatisch ein Editor ($EDITOR) geöffnet, in dem eine Mitteilung eingetragen werden sollte!

```
> cvs commit modchanged.c
Checking in modchanged.c;
/projects/cvs/source/modchanged.c,v  <--  modchanged.c
new revision: 1.4; previous revision: 1.3
done
```

CVS-Befehle add, remove und release

`add`:	Fügt eine neue Datei oder ein Verzeichnis hinzu
`remove`:	Entfernt einen Eintrag aus dem Archiv - alte Revisionen bleiben aber erhalten!
`release`:	Löschen der Arbeitskopie (vorbereiten)

Syntax:

```
cvs add [Dateien]; cvs commit [Dateien]

rm [Dateien];cvs remove [Dateien]; cvs commit [Dateien]

cvs release [-d] Projektverzeichnis
```

- Sowohl bei add als auch bei remove ist es sinnvoll die Dateinamen explizit anzugeben!
- Release -d löscht (nach Rückfrage) die Arbeitskopie, daher mit Vorsicht anwenden!

CVS-Befehle add, remove und release

```
> edit cvs_add_remove_test
> cvs add cvs_add_remove_test
cvs add: scheduling file `cvs_add_remove_test' for addition
cvs add: use 'cvs commit' to add this file permanently
> cvs commit -m "Nur ein Test" cvs_add_remove_test
```

cvs add

```
> rm cvs_add_remove_test
> cvs remove cvs_add_remove_test
cvs remove: scheduling `cvs_add_remove_test' for removal
cvs remove: use 'cvs commit' to remove this file permanently
> cvs commit -m "Dies war nur ein Test" cvs_add_remove_test
Removing cvs_add_remove_test;
/projects/cvs/source/cvs_add_remove_test,v  <--
cvs_add_remove_test
new revision: delete; previous revision: 1.1
done
```

cvs remove

```
> cd ..
> cvs release -d serie
You have [0] altered files in this repository.
Are you sure you want to release (and delete) directory `serie': y
cvs release: no such directory: serie
```

cvs release

Manuelle Kontrolle, ob wirklich keine Datei verändert wurde bevor ein y (yes) eingegeben wird!

CVS-Befehle status und log

`status`: Zeigt den aktuellen Status von Dateien in der Arbeitskopie an
`log`: Gibt History-Daten aus

Syntax:

```
cvs status [Dateien]

cvs log [Dateien]
```

```
> cvs status header.h
===================================================================
File: header.h          Status: Up-to-date

   Working revision:    1.4
   Repository revision: 1.4     /projects/cvs/source/header.h,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

CVS-Befehl log

```
> cvs log header.h

RCS file: /projects/cvs/source/header.h,v
Working file: header.h
head: 1.4
branch:
locks: strict
access list:
symbolic names:
        REL_3_8_2: 1.4
        REL_3_8_0: 1.1.1.1
        REL_3_8_1: 1.4
        INITIAL: 1.1.1.1
        EXT: 1.1.1
keyword substitution: kv
total revisions: 5;     selected revisions: 5
description:
----------------------------
revision 1.4
date: 2002/08/30 07:58:31;  author: user;  state: Exp;  lines: +8 -4
Demo Aenderung
----------------------------
...........
revision 1.1
date: 2002/08/30 06:58:48;  author: user;  state: Exp;
branches:  1.1.1;
Initial revision
----------------------------
revision 1.1.1.1
date: 2002/08/30 06:58:48;  author: user;  state: Exp;  lines: +0 -0
Initial import
=============================================================================
```

CVS-Befehl tag

Syntax: `cvs tag [-b] SymbolischerName [Dateien]`
Gibt einer ausgecheckten Version von Dateien einen symbolischen Namen.
Achtung: Dateinamen nur angeben, wenn man expliziert nur diesen Dateien einen Namen zuweisen will. Meist soll aber dem aktuellen Projektzustand (d.h. allen Dateien!) ein symbolischer Name bzw. ein Release Name zugewiesen werden!
Die Option -b erzeugt eine Variante (Branch).

```
> cvs tag REL_3_8_2
```

Mit dem Befehl

```
> cvs checkout -r REL_3_8_2 Projektname
```

kann dann der Projektzustand REL_3_8_2 wiederhergestellt werden. Dieser Befehl setzt aber das Sticky-Tag!

Der symbolische Name darf z.B. keine Leerzeichen enthalten!

CVS is rather strict about what constitutes a valid tag name. The rules are that it *must start with a letter and contain letters, digits, hyphens ("-"), and underscores ("_")*. No spaces, periods, colons, commas, or any other symbols may be used.

CVS-Befehl update

Syntax: `cvs update [-r rev] [-D Datum] [-A] [Dateien]`
Überführt Änderungen aus dem Repository in die Arbeitskopie. Es können aber auch alte Revisionen (Option -r), mit einem symbolischen Namen versehene Projektzustände oder auch Projektzustände von einem Datum überführt werden. Die Option -A setzt Sticky-Tags zurück! Achtung: Wenn man Dateinamen angibt werden nur diese berücksichtigt - ansonsten alle. Gibt man z.B. eine Revision an die es noch nicht gibt, werden diese Dateien aus der Arbeitskopie gelöscht!

M = modified
U = updated
C = conflict
? = Datei unbekannt

Standardverwendung

```
> cvs update
```

Alte Versionen herausholen
Achtung: CVS setzt das Sticky-Tag bei diesen Befehlen!

```
> cvs update -r REL_3_8_2
> cvs update -D "19 Apr 1999 23:40:30 GMT"
```

Sticky-Tag rückgängig machen

```
> cvs update -A
```

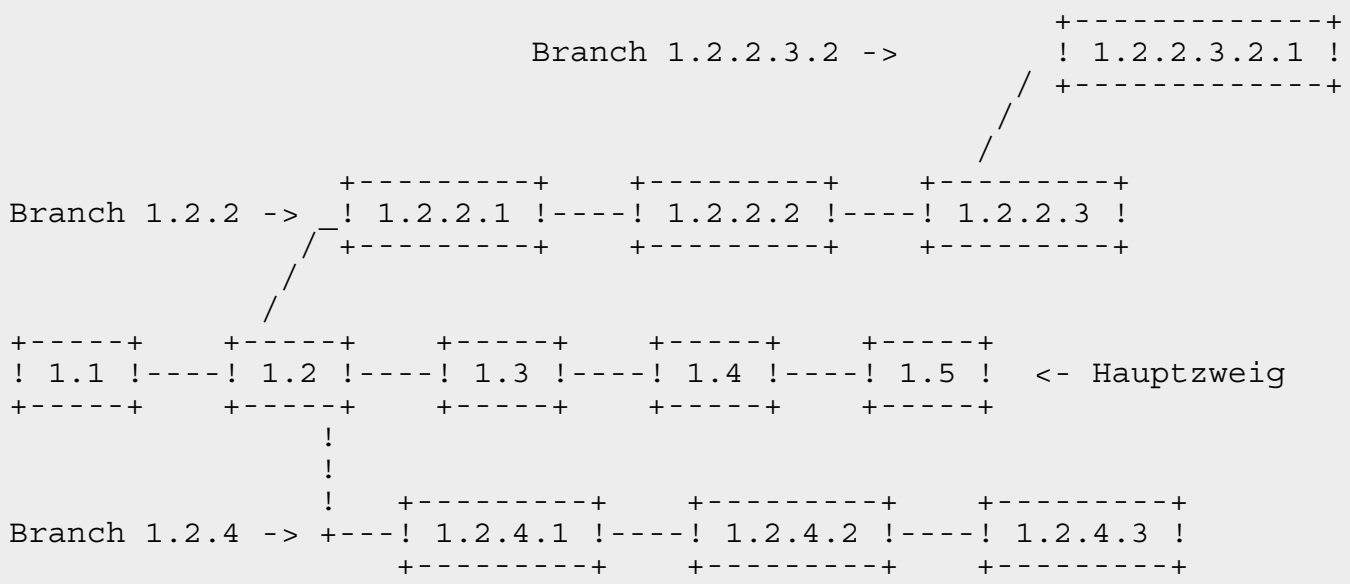
Sticky-Tags

Bei einem cvs update-Befehl auf eine alte Version setzt cvs das Sticky-Tag. Alle Änderungen beziehen sich ab sofort auf die alte Version! Ein cvs update holt nicht mehr die neueste Version heraus!

```
> cvs update -r REL_3_8_1 modchanged.c
U modchanged.c
> cvs status modchanged.c
 ===================================================================
File: modchanged.c Status: Up-to-date

   Working revision:    1.2
   Repository revision: 1.2     /projects/cvs/source/modchanged.c,v
   Sticky Tag:          REL_3_8_1 (revision: 1.2)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Branches (Zweige, Varianten)



Branchnummer: Ungerade Anzahl von Zahlen, letzte Zahl gerade.
Revisionsnummer: Gerade Anzahl von Zahlen.

Branches bearbeiten

Mit dem cvs tag-Befehl kann man Branches erzeugen. Mit dem folgenden Befehl wird ein Branch der Arbeitskopie erzeugt.

```
> cvs tag -b REL_3_8_1_PATCH1
```

Mit der Arbeitskopie auf den Branch setzen:

```
> cvs update -r REL_3_8_1_PATCH1
```

Branch und Hauptzweig zusammenführen (Join)

```
> cvs update -r HEAD
.....
> cvs update -j REL_3_8_1_PATCH1 header.h
RCS file: /projects/cvs/source/header.h,v
retrieving revision 1.4
retrieving revision 1.4.2.1
Merging differences between 1.4 and 1.4.2.1 into header.h
rcsmerge: warning: conflicts during merge
```

Warnungen nur bei überlappenden Textbereichen, sonst nicht!

CVS-Befehl import

Syntax: `cvs import Projekt VendorTag ReleaseTag`

Importiert Quellcode eines „externen Softwareherstellers“ ins Repository. Alle Dateien des aktuellen Verzeichnisses werden in einen Branch (1.1.1) ins Repository geschrieben. Neue Releases des Herstellers können unter Angabe eines anderen Release Tags in den gleichen Branch importiert werden. Diese Releases können mit Hilfe des Release Tags wie „normale“ Releases bearbeitet werden. Auschecken, Zusammenführen, ...

Mit CVS können auch verschiede Hersteller verwaltet werden. Einzelheiten stehen in der CVS Dokumentation.

CVS - Beispiel zwei Entwickler

```
> cvs status header.h
===================================
File: header.h          Status: Up-to-date

   Working revision:    1.6
....

> edit header.h
> cvs commit header.h
Checking in header.h;
/projects/cvs/source/header.h,v  <--  header.h
new revision: 1.7; previous revision: 1.6
done
```

```
> cvs status header.h
===================================
File: header.h          Status: Up-to-date

   Working revision:    1.6
   Repository revision: 1.6
/projects/cvs/source/header.h,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)


> edit header.h
> cvs update header.h
RCS file: /projects/cvs/source/header.h,v
retrieving revision 1.6
retrieving revision 1.7
Merging differences between 1.6 and 1.7 into
header.h
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in header.h
C header.h
```

CVS - Beispiel zwei Entwickler

In der Datei wird der überlappende Teil als Konflikt markiert.

```
<<<<<<< header.h

/* Überlappender Bereich Entwickler2                                    */

=======

/* Überlappender Bereich Entwickler1                                    */

>>>>>>> 1.7
```

Der nicht überlappende Teil wird stillschweigend zusammengemischt!
Nach Beseitigung der Konflikte kann commit durchgeführt werden.

Was CVS nicht kann

- CVS erstellt keine Datei- und Verzeichnisstruktur.
- CVS gibt nicht vor, wann commit oder tag durchgeführt werden.
- CVS ersetzt keine Entwicklerabsprachen. Die Konfliktanzeige basiert auf reiner Textverarbeitung. Logische Konflikte werden nicht erkannt.
- CVS hat keinen eingebauten Workflow (Freigabe, Testverfahren).
- CVS kennt keine Projektverwaltung (Meilensteine, Terminverfolgung).
- CVS verwaltet Änderungen an Software. Die Gründe für diese Änderungen (neue Anforderungen, Fehlerbehebung,...) werden nicht verwaltet.

CVS-Keywörter

`$Author$, $Date$, $Header$, $Id$, $Log$, $Name$, $Revision$`

Werden beim Einchecken ins Repository ersetzt/expandiert:

```
$Author: user $
$Date: 2002/09/20 10:08:29 $
$Revision: 1.4 $

$Log: modchanged.c,v $
Revision 1.4  2002/09/20 10:08:29  user
Doxygen Kommentare ergänzt

Revision 1.3  2002/09/12 11:47:36  user
Wichtige Änderung eingefügt

Revision 1.2  2002/09/04 12:41:24  user
Wenn der Parameter 'Nachmessen' geaendert wird, wird die Funktion
NachmessMode(int mode) aufgerufen.
mode = 1   : Nachmessmode
mode = 0   : kein Nachmessmode
```

CVS-Umgebungsvariablen

$CVSROOT: Verweist auf das Archiv. Durch ändern von CVSROOT kann der Nutzer auf ein anderes Archiv zugreifen

$CVSEDITOR, $VISUAL und $EDITOR: Editor zur Eingabe Mitteilungen

$CVSIGNORE: Namen von Dateien bzw. Wildcards die von CVS ignoriert werden sollen

CVS Befehle

`cvs checkout` Projektname

Holt die Quellen eines Software Projekts aus dem Repository.

Legt lokal ein Unterverzeichnis mit Projektname an.

`cvs update`

Überführt Änderungen aus dem Repository in die Arbeitskopie und zeigt eigene Änderungen an.

`cvs commit` Datei

Überführt eigene Änderungen in Repository.

`cvs import` Projektname Vendor-Tag Release-Tag

Importiert Quellcode erstmalig ins Repository.

`cvs tag` Name

Gibt einem Softwarestand einen symbolischen Namen.

CVSweb

- Web Frontend für lesenden Zugriff auf ein Repository
- Alle Dateien eines Projekts zum Browsen
- Status- und Log-Informationen
- Diffs zwischen beliebigen Versionen, als HTML aufbereitet