Frank Budszuhn

CVS

Galileo Computing

Auf einen Blick

Inhalt

Vorwort

Der Charakter der Programm- und Softwareentwicklung hat sich in den letzten Jahrzehnten stark verändert. War es vor 20 bis 30 Jahren noch normal, dass Programme von einzelnen Programmierern entwickelt wurden, so ist das heute in den meisten Bereichen undenkbar. Heutige »Softwarepakete« sind Konstrukte von nahezu beliebiger Komplexität. Solche Werke können gar nicht mehr von einzelnen Programmierern geschaffen werden; solche Werke entstehen fast nur noch durch große Teams von Entwicklern. Dabei hat die Komplexität der Software beinahe alle Bereiche der Softwareentwicklung erfasst: nicht nur Office-Pakete und Betriebssysteme werden von großen Teams entwickelt, auch so vermeintlich einfache Programme wie die Software in Handies und Autos übersteigt mittlerweile die Möglichkeiten eines einzelnen Softwareentwicklers. Zusammen mit dem Übergang vom einzelnen Programmierer zum gemeinsam entwickelnden Team hat sich der Prozess der Softwareentwicklung verändert. Die Veränderung betrifft mehrere Aspekte des Entwicklungsprozesses. An erster Stelle steht die Planung der Softwarearchitektur mit einem modularen und einfach zu durchschauenden Aufbau. Zweitens ist die Kommunikation der Entwickler untereinander sehr wichtig. Und drittens gilt es Werkzeuge einzusetzen, die den Softwareentwicklungsprozess im Team unterstützen. Eines der wichtigsten Werkzeuge übernimmt dabei die Verwaltung des Quell- oder Sourcecodes, der allen Programmen zugrunde liegt. War der Sourcecode im Zeitalter des einzelkämpfenden Programmierers oft sein bestgehüteter Schatz, so müssen heutige Softwareentwickler lernen, den Sourcecode mit anderen Entwicklern zu teilen, ja gemeinsam an vielen Teilen des Programmcodes gleichzeitig zu arbeiten. Dabei kommen (neben der mentalen Umstellung) allerlei praktische Aspekte ans Tageslicht: Wie können überhaupt mehrere Entwickler gleichzeitig mit dem gleichen Sourcecode arbeiten? Wie bekommen die Kollegen meine Änderungen mit? Wie bekomme ich die Änderungen der Kollegen mit? Wie wird vermieden, dass ein solches Unterfangen im Chaos endet?

Um Probleme dieser Art zu lösen, wird Softwareentwicklung im Team heute ausschließlich mithilfe von Versionsmanagementsystemen durchgeführt. Ein Versionsmanagementsystem führt die Änderungen der einzelnen Entwickler zusammen, deckt Konflikte auf und verwaltet eindeutig reproduzierbare Versionsstände. Ein solches Versionsmanagementsystem ist CVS, das *Concurrent Version System*. Es ist das bekannteste und meist verbreitete Versionsmanagementsystem aus dem Open Source Bereich und

wird in fast allen Open Source Projekten eingesetzt. Die Mozilla-Browser-Suite wird ebenso unter CVS entwickelt wie das grafische Benutzer-Interface der Unix-Welt XFree und die meisten Werkzeuge aus dem GNU-Bereich. Die Entwicklerseite sourceforge.net verwaltete zum Zeitpunkt der Abfassung dieses Buches mehr als 67.000 Softwareprojekte mit CVS. Lediglich der Linux-Kernel wird seit Februar 2002 nicht mehr unter CVS entwickelt; man hat sich hier für ein kommerzielles Versionsmanagementsystem entschieden. Da CVS der GNU Public Licence unterliegt, kann es von jedermann und jeder Firma kostenfrei eingesetzt werden. CVS ist keinesfalls auf Open Source Projekte beschränkt, es wird in vielen Firmen zur Entwicklung kommerzieller Softwareprodukte eingesetzt. Dieses Buch möchte Sie mit dem Einstieg in die Arbeit mit einem Versionsmanagementsystem im Allgemeinen und mit CVS im Speziellen vertraut machen.

An dieser Stelle möchte ich mich für die gute und unkomplizierte Zusammenarbeit mit dem Verlag, insbesondere bei meiner Lektorin Judith Stevens-Lemoine bedanken. Besonderer Dank geht an meine Ehefrau, Elke Szurowski, die dieses Buchprojekt aktiv unterstützt und alle Grafiken dazu erstellt hat.

Wenn Sie Anregungen, Kritik oder Korrekturen zu diesem Buch haben, so zögern Sie bitte nicht, mir eine E-Mail an **autor@cvsbuch.de** zu schicken und schauen Sie auch mal auf der Webseite zum Buch unter **http://www.cvsbuch.de** vorbei!

Hamburg, 15.02.2004
Frank Budszuhn

Der Autor

Frank Budszuhn (E-Mail: autor@cvsbuch.de), Jahrgang 1966, ist Diplominformatiker und arbeitet seit 1994 als Softwareentwickler. Nach dem Studium der Informatik in Hamburg und in England arbeitete er mehrere Jahre für den Audiospezialisten Steinberg, wo er unter anderem die Grundlagen für das Audiorestaurationsprogramm »Clean« legte. Ab 1999 entwickelte er komplexe Webapplikationen für die Internetagenturen Fluxx.com, Netmatic und SinnerSchrader. Seit 2002 erstellt er in seiner eigenen Firma, der AlsterContor GmbH, Internetanwendungen im Kundenauftrag. Während seiner 10-jährigen Berufstätigkeit hat er alle größeren Softwareprojekte mit CVS verwaltet. Frank Budszuhn ist außerdem Autor des Buchs »Visual C++, Windows-Programmierung mit den MFC«, das bei Addison Wesley erschienen ist.

2 Das Concurrent Versions System

Dieses Kapitel gibt einen kurzen Rückblick auf die Geschichte von CVS, nennt einige andere Versionsmanagementsysteme, führt Client-Programme für CVS auf und beleuchtet, was CVS nicht ist und nicht sein kann.

2.1 Zur Geschichte von CVS

Die Geschichte des Versionsmanagements ist nicht ganz so lang wie die Geschichte der Softwareentwicklung. Einerseits waren die ersten Fortran- und Cobol-Programme meist noch von einem einzelnen Entwickler erstellt worden, andererseits konnte man die alten Versionen in Form der Lochkartenstapel aufheben. Zudem war zu Beginn der Computerentwicklung Speicherplatz so teuer, dass sich das Vorhalten mehrerer Versionen eines Programmcodes aus Kostengründen nicht anbot.

SCCS

Als eines der ersten Systeme zum Versionsmanagement lässt sich heute das von Marc Rochkind entwickelte SCCS (Source Code Control System) ausmachen. Dieses System wurde Anfang der 70er-Jahre bei AT&T entwickelt und brachte schon einige der heute von CVS verwendeten Konzepte mit. Das System konnte bereits mehrere Versionen einer Datei verwalten, allerdings war es nur für einen einzelnen Entwickler gedacht. Es gibt auch heute noch ein GNU-Replacement für SCCS (siehe Anhang D), das System wird aber kaum noch verwendet.

RCS

Etwa 10 Jahre später, 1985, baute Walter Tichy von der Purdue University das RCS (Revision Control System) auf der Basis von SCCS auf. RCS behandelte erstmals das Problem der Zusammenarbeit mehrerer Softwareentwickler. Allerdings verwendet RCS einen Ansatz, der es nur einem Entwickler zur Zeit erlaubt, eine bestimmte Programmcodedatei zu verändern; die Datei wird für alle anderen Entwickler zum Schreiben gesperrt (Locking). RCS wird heute noch manchmal zum Versionsmanagement verwendet, in der Regel gibt man allerdings moderneren Systemen wie CVS den Vorzug.

CVS

Bereits ein Jahr später, 1986, wird die Entwicklung von CVS angestoßen. In einer ersten Version bestand CVS aus einer Reihe von Unix-Shell-Skripten, die 1986 von Dick Grune im Usenet verteilt wurden. Im Jahre 1989 wurde CVS dann von Jeff Polk und Brian Berliner in C neu geschrieben.

CVS basiert auf RCS, verwendet allerdings einen anderen Ansatz bei der Zusammenarbeit mehrerer Entwickler. Bei CVS besitzt jeder Entwickler seine eigene Arbeitskopie einer Datei. Hier nimmt er alle Änderungen vor; die Versionsstände verschiedener Entwickler werden später von CVS zusammengeführt. Weiterhin führt CVS das Konzept der Module ein und betrachtet nicht nur einzelne Dateien.

2.2 CVS im Kontext anderer Versionsmanagementsysteme

Kein beherrschendes System

Wie aus der Geschichte von CVS deutlich wird, ist CVS nicht das einzige Versionsmanagementsystem. Neben CVS gibt es eine ganze Reihe freier und kommerzieller Systeme. Interessanterweise hat im kommerziellen Bereich kein einziges Werkzeug eine herausragende Stellung erreichen können. Es gibt eine ganze Reihe verschiedener Anbieter und Werkzeuge, die unterschiedlich eingesetzt werden.

- Die bereits genannten Systeme SCCS und RCS gehören eher zu den »Klassikern« des Versionsmanagements. SCCS sollte nicht mehr eingesetzt werden und auch RCS ist kein modernes System und hat das Problem des File-Lockings.
- Neben CVS ist im Open Source-Bereich sicherlich das in der Entstehung befindliche Versionsmanagementsystem Subversion am interessantesten. Dieses System steht noch relativ am Anfang der Entwicklung, stellt sich aber als potenzieller Nachfolger von CVS dar (siehe Kapitel 9).
- Wer im Microsoft-Umfeld Software entwickelt, der wird früher oder später auf den Visual Source Safe (VSS) stoßen, das Versionsmanagementsystem von Microsoft. Der größte Vorteil dieses Systems ist sicherlich die optimale Integration in die Entwicklungsumgebungen des Herstellers.
- ClearCase von Rational ist ein »großes«, teures, kommerzielles Versionsmanagementsystem. ClearCase ist ressourcenhungrig und benötigt einen großen Server. Wer es üppig braucht, ist hier richtig. Für die meisten anderen ist der Aufwand jedoch nicht unbedingt angemessen.
- Perforce ist ein kleines kommerzielles Versionsmanagementsystem, das CVS in vielen Punkten ähnelt. Allerdings bügelt es einige Schwächen von CVS aus und ist unter dem Strich schneller.

- BitKeeper ist ebenfalls ein kleiner, schneller und kommerzieller CVS-Ersatz. Interssanterweise wird der Linux-Kernel seit Version 2.5 unter BitKeeper entwickelt. Damit ist der Linux-Kernel eines der ganz wenigen Open Source Projekte, das nicht CVS verwendet.

2.3 Clients für CVS

CVS ist – mit Ausnahme der Zugriffsvariante `local` – ein Client-Server-System. Auf einem speziellen Server im Netzwerk befindet sich der CVS-Server, auf den alle CVS-Benutzer mit ihren Client-Programmen zugreifen. Der Standard-Client für CVS ist eine Kommandozeilen-Applikation, die auch in diesem Buch beschrieben wird. Daneben haben sich eine ganze Reihe von grafischen Werkzeugen entwickelt, die ebenfalls zum Zugriff auf einen CVS-Server verwendet werden können. Einige von diesen sog. GUI-Clients sollen hier kurz vorgestellt werden.

2.3.1 WinCvs

WinCvs ist der wohl am meisten genutzte CVS-Client auf der Windows-Plattform. WinCvs stellt sich dem Benutzer als Fenster mit dreigeteilter Ansicht dar. Links oben zeigt der Modul-Browser Module und Verzeichnisse an. Rechts daneben werden entweder die Dateien eines Verzeichnisses dargestellt oder das grafische Log einer oder mehrerer Dateien. Unten befindet sich das Ausgabefenster, das auch als Shell zur Eingabe von Befehlen verwendet werden kann.

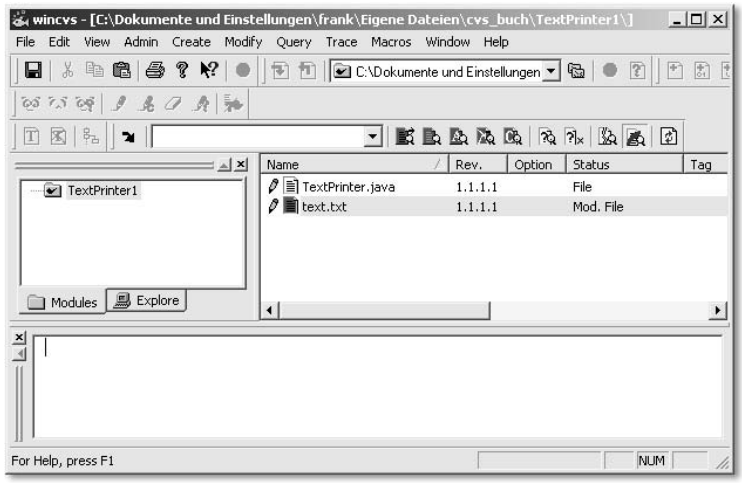

Abbildung 2.1 Die dreigeteilte Ansicht von WinCvs

WinCvs wird im Rahmen dieses Buchs ausführlich vorgestellt werden.

2.3.2 gCvs

gCvs ist das Gegenstück zu WinCvs auf Linux und anderen Unix-Plattformen. gCvs teilt sich – wie auch das unten aufgeführte MacCvsX – einen Teil der Implementierung mit WinCvs. Die Benutzeroberfläche ist jedoch für jedes Programm separat mit einer jeweils plattformspezifischen Klassenbibliothek implementiert worden. Die von den drei Programmen verwendeten Symbole sind in ihrer Bedeutung immer gleich. gCvs hat bisher noch nicht den vollständigen Funktionsumfang von WinCvs erreicht.

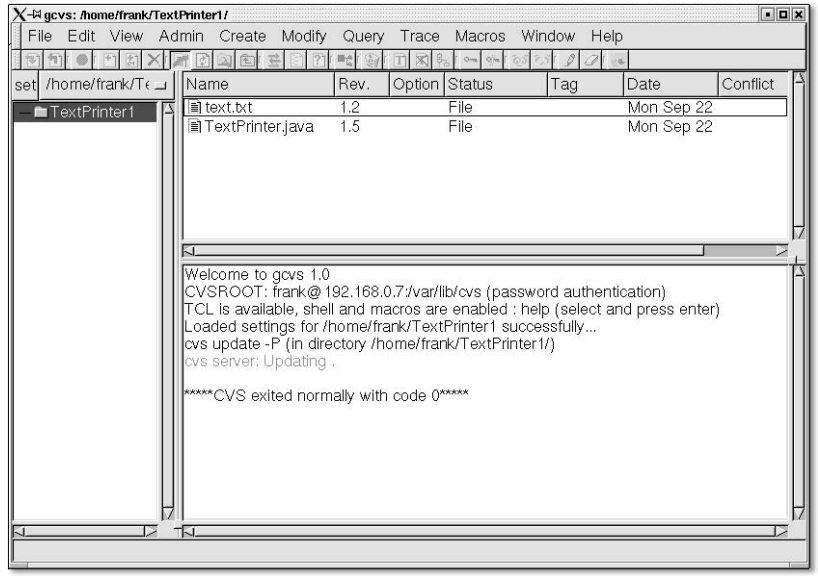

Abbildung 2.2 gCvs in der Version 1.0

2.3.3 MacCvsX

MacCvsX ist ein grafischer CVS-Client für die Apple-Plattform. MacCvsX ist ähnlich wie WinCvs und gCvs aufgebaut. Es verwendet eine Mac-spezifische Klassenbibliothek zur Darstellung seiner Oberfläche, weshalb das »Look & Feel« etwas anders ist.

Erst seit MacOS X ist auch auf dem Apple der Einsatz der Kommandozeilenversion von CVS möglich, da das Mac-Betriebssystem nun auf Unix basiert. Vorher war man auf dem Mac immer auf eigene GUI-Clients wie MacCvs (ohne »X«) angewiesen.

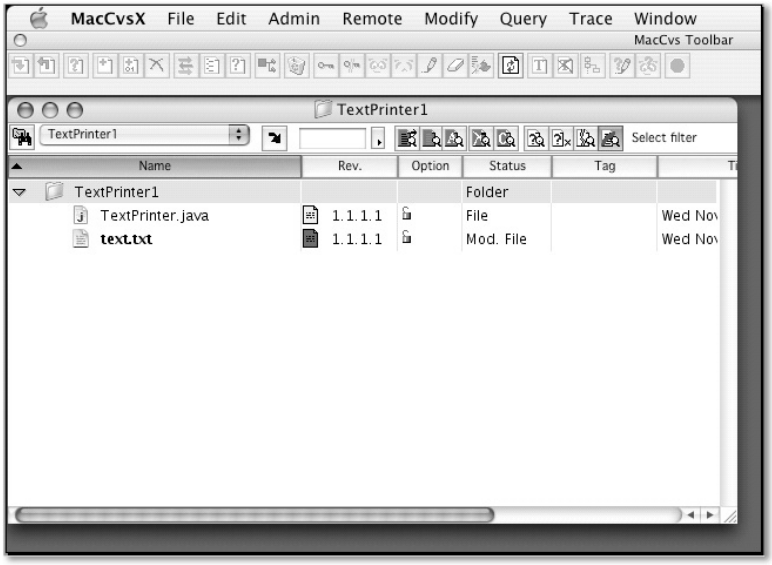

Abbildung 2.3 MacCvsX mit der Aqua-Oberfläche

2.3.4 CVL

Während MacCvsX die Portierung eines Clients aus der »alten« Mac-Welt ist (es gab vorher eine Version für MacOS 9 und frühere Versionen), so kommt CVL (Concurrent Versions Librarian) aus der NextStep-/OpenStep-Welt. Das Programm kann in den MacOSX-ProjectBuilder, die Entwicklungsumgebung von MacOS X, integriert werden.

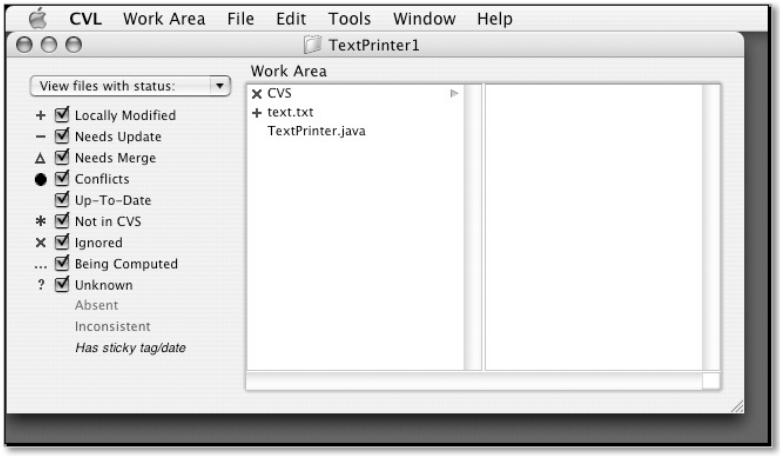

Abbildung 2.4 Concurrent Versions Librarian auf MacOS X

2.3.5 TortoiseCVS

TortoiseCVS ist eine Erweiterung des Windows-Explorers und macht die gängigsten CVS-Funktionen in diesem als Kontextmenü zugänglich. Der Zustand einzelner Dateien wird mit speziellen Icons dargestellt.

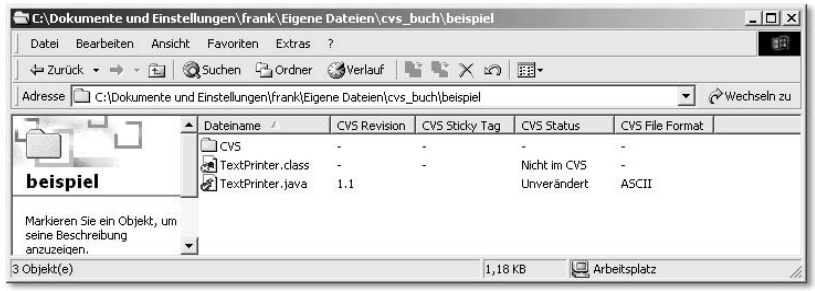

Abbildung 2.5 TortoiseCVS ist eine Erweiterung des Windows-Explorers

TortoiseCVS ist einfach zu bedienen. Man benötigt mit TortoiseCVS kein separates Client-Programm für den Zugriff auf das CVS-System. Für einfache Projektmitglieder kann TortoiseCVS durchaus eine Alternative zu einem CVS-Client-Programm darstellen. Der Projektmanager braucht in jedem Fall zusätzlich ein weiteres Client-Programm, da TortoiseCVS nicht alle Befehle von CVS unterstützt.

2.3.6 Eclipse

Eclipse, ursprünglich eine Java-Entwicklungsumgebung von IBM, ist mittlerweile wohl eines der am vielfältigsten einsetzbaren grafischen Entwicklerwerkzeuge überhaupt. Durch das sehr ausgefeilte Plug-In-Konzept von Eclipse lässt sich diese Entwicklungsumgebung weit über die Programmierung von Java-Anwendungen hinaus einsetzen. So kann man aus der Eclipse-Umgebung direkt auf einen CVS-Server zugreifen.

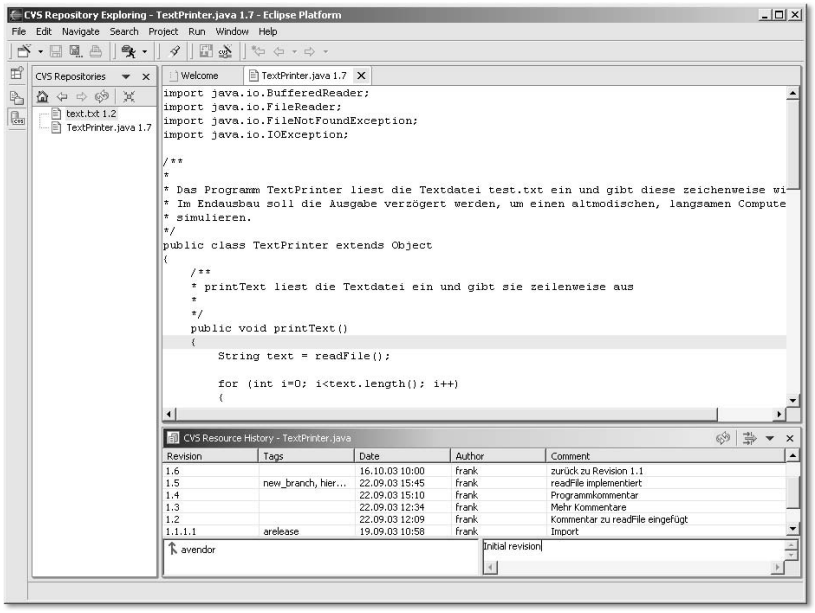

Abbildung 2.6 CVS Repository Exploring Perspective in Eclipse

2.4 Was CVS nicht kann: Abgrenzung zu anderen Entwicklungswerkzeugen

Die Verwaltungseinheit von CVS ist das *Modul*. Ein Modul ist nichts weiter als ein Verzeichnis mit einer Anzahl von Dateien darin und eventuell weiteren Unterverzeichnissen. CVS kümmert sich jedoch nicht um semantische Bezüge der Dateien untereinander, d.h., CVS weiß nichts von Projekten-Dateien oder anderen speziell für den Build-Prozess verwalteten Dateien. Dies ist weiterhin Aufgabe der benutzten Entwicklungsumgebung. Verwendet man beispielsweise Make-Dateien innerhalb eines Softwareprojektes, so müssen diese genauso unter CVS-Kontrolle gestellt werden, wie der Sourcecode selbst. Die Logik zur Verwaltung von Make- oder Projekt-Dateien liegt komplett außerhalb von CVS. Weiterhin besitzt CVS keine Mechanismen zur Qualitätssicherung und Fehlerverwaltung. Hierzu muss man zusätzlich andere Werkzeuge heranziehen.

CVS ist ein reines Werkzeug für Programmierer und Softwareentwickler. Es ist kein Werkzeug für das Management. Nur, wer auch im Projekt Zugriff auf den Sourcecode hat, sollte Zugriff auf CVS haben.

2.5 CVS und Open Source

Mit der bereits genannten Ausnahme des Linux-Kernels und der Datenbank MySQL werden fast alle Open Source-Projekte unter CVS entwickelt. Einerseits entspricht dies dem Gedanken, dass Open Source Entwicklung auch Open Source-Werkzeuge verwenden sollte. Andererseits belegt dies die Reife und die Leistungsfähigkeit von CVS.

Closed Source

CVS ist jedoch nicht auf Open Source-Projekte beschränkt. Zwar kommt das offene Entwicklungsmodell von CVS der Open Source-Entwicklung sehr entgegen, jedoch lässt sich CVS genauso gut für kommerzielle und Closed Source-Entwicklungen verwenden. CVS wird unter der GNU Public License (GPL) herausgegeben, die auch den kommerziellen Einsatz unbeschränkt zulässt.

CVS und Unix

CVS ist anders als viele andere Open Source-Projekte nicht fest mit der Unix-Plattform verknüpft. Zwar werden viele CVS-Server unter Unix betrieben (es gibt auch eine Server-Variante für Windows NT und seine Nachfolger), allerdings ist das Angebot an CVS-Clients bei Windows und MacOS(X) größer, da auf Unix oft noch mit der Kommandozeile gearbeitet wird. Man braucht also nicht zu befürchten, sich durch den Einsatz von CVS auf ein ungewolltes Unix-Abenteuer einlassen zu müssen.

7 Fortgeschrittene CVS-Themen

Dieses Kapitel geht auf die Themenbereiche von CVS ein, die nicht zum alltäglichen Repertoire eines jeden CVS-Benutzers gehören. Es gibt ein paar Tipps zur Verwaltung von HTML-Dokumenten und Webseiten mit CVS. Außerdem werden die zusätzlichen Werkzeuge CVSWeb und ViewCVS vorgestellt.

7.1 Optionen vorgeben

Es gibt eine Reihe von Befehlsoptionen, die man eigentlich bei jedem Aufruf angeben möchte. So kann es sein, dass man den Befehl **update** immer mit der Option –P aufrufen möchte, um leere Verzeichnisse zu löschen. Oder man möchte ebenfalls immer die Option –d angeben, um neu dazugekommene Verzeichnisse anzulegen. Um solche Optionen immer mit dem Befehl automatisch anzugeben, kann man eine Datei mit dem Namen **.cvsrc** anlegen und die Optionen dort hineinschreiben. Eine solche Datei kann beispielsweise folgendermaßen aussehen:

```
update -d -P
diff -c
```

Home-Verzeichnis

Unter Unix wird eine solche Datei immer im Home-Verzeichnis des Benutzers gespeichert. Unter Windows legt man sie in dem Verzeichnis ab, den die beiden Umgebungsvariablen `HOMEDRIVE` und `HOMEPATH` zusammen angeben. Hat man diese nicht explizit gesetzt, so wird das Hauptverzeichnis des Laufwerks angenommen, auf dem Windows installiert ist. Oft ist dies dann also **c:**.

7.2 Befehle abkürzen

Synonyme

Die Kommandozeilen-Clients erlauben es, Befehle abzukürzen, so dass man sie schneller eintippen kann. Für viele Befehle gibt es auch noch weitere Synonyme. Eine Liste erhält man, indem man `cvs --help-synonyms` eingibt.

Befehl	Synonyme
add	ad, new
admin	adm, rcs
annotate	ann
checkout	co, get
commit	ci, com
diff	di, dif
export	exp, ex
history	hi, his
import	im, imp
log	lo
login	logon, lgn
rannotate	rann, ra
rdiff	patch, pa
release	re, rel
remove	rm, delete
rlog	rl
rtag	rt, rfreeze
status	st, stat
tag	ta, freeze
update	up, upd
version	ve, ver

7.3 Dateien ignorieren: cvsignore

Es gibt fast in jedem Softwareprojekt Dateien und Verzeichnisse, die beim Erstellungsprozess der Software selbst entstehen, die also nicht Teil des Sourcecodes sind. Weil sie nicht Teil des Sourcecodes sind, wird man sie daher auch nicht mit CVS verwalten wollen. Beispiele für solche Dateien sind **class**-Dateien in Java oder fertig übersetzte **exe**-Dateien bei einer Windows-Programmentwicklung. CVS kennzeichnet solche nicht im Repository befindlichen Dateien mit einem Fragezeichen, beispielsweise beim Befehl **update**:

```
cvs update TextPrinter1
```

Falls sich im gleichen Verzeichnis die fertig kompilierte Datei **TextPrinter.class** befindet, gibt CVS aus:

```
? TextPrinter1/TextPrinter.class
cvs server: Updating TextPrinter1
```

Man kann so ohne Probleme mit CVS arbeiten, falls einen diese Ausgaben nicht stören. Allerdings kann es bei vielen solcher Dateien recht unübersichtlich werden und man weiß später nicht mehr, welche Dateien man tatsächlich vergessen hat, dem Repository hinzuzufügen. Man kann daher CVS anweisen, Dateien mit vorgegebenen Namen zu ignorieren. Eine Möglichkeit hierzu ist es, eine Datei mit dem Namen **.cvsignore** (man achte auf den Punkt am Anfang) im Home-Verzeichnis des Benutzers abzulegen (bei Windows wird das Home-Verzeichnis durch die Umgebungsvariablen `HOMEDRIVE` und `HOMEPATH` bestimmt). In jeder Zeile dieser Datei wird ein Name angegeben, der ignoriert werden soll. Da Wildcards erlaubt sind, kann man Dateien mit vorgegebenen Endungen ausblenden. Eine solche Datei könnte beispielsweise folgendermaßen aussehen:


```
*.class
*.exe
*.dll
notizen.txt
```

Lokale .cvsignore-Dateien

Man kann **.cvsignore**-Dateien auch im Projekt selbst anlegen. Sie gelten dann jeweils nur für das Verzeichnis, in dem sie gespeichert sind. Solche lokalen **.cvsignore**-Dateien sollten dann auch dem Repository hinzugefügt werden, damit alle Entwickler die gleichen Dateien ignorieren.

Der Windows-Explorer kann Dateien mit einem Punkt zu Beginn nicht anlegen. Daher legt man die Datei zunächst ohne einen Punkt an und speichert sie ab. Dann öffnet man eine Eingabeaufforderung und benennt die Datei mit dem Befehl

```
ren cvsignore .cvsignore
```

um!

Serverseitige .cvsignore-Datei

Auch bei den serverseitigen Konfigurationsdateien (siehe Abschnitt 8.2, »Die administrativen Dateien in CVSROOT«) kann man eine Datei **cvsignore** (in diesem Fall ohne Punkt) hinzufügen. Normalerweise ist sie

nach der Installation des CVS-Servers nicht vorhanden. Die Syntax dieser Datei ist die gleiche, wie bei den auf dem Client gespeicherten Versionen.

Die Umgebungsvariable CVSIGNORE

Schließlich gibt es auch die Möglichkeit, statt einer Datei die Umgebungsvariable `CVSIGNORE` zu setzen. In dieser werden die zu ignorierenden Dateinamen durch Leerzeichen getrennt.

WinCvs und gCvs kennzeichnen ignorierte Dateien und Verzeichnisse mit einem blau durchgestrichenen Symbol oder blenden sie ganz aus (Umschaltung über **View · File Filter · Show Ignored**). Allerdings berücksichtigen sie dabei leider nicht die serverseitige **cvsignore**-Datei.

Dateien beim Import ignorieren

Wichtig kann **cvsignore** beim Import sein. Sind beim Import bereits Dateien vorhanden, die nicht ins Repository importiert werden sollen, so ist es ratsam, diese Dateien vorher in **cvsignore** aufzunehmen. Man spart sich dann sehr viele Angaben mittels der –I-Option von **import**. Außerdem ist das Vorgehen mittels **cvsignore** weniger fehlerträchtig.

Vier verschiedene Quellen

Die tatsächlich zu ignorierenden Dateien werden aus allen vier Informationsquellen zusammen bestimmt. Dabei gilt die serverseitige Reihenfolge: **cvsignore**-Datei, **.cvsignore** im Home-Verzeichnis, die Umgebungsvariable `CVSIGNORE` und zum Schluss die **.cvsignore**-Datei im Verzeichnis der Arbeitskopie. Außerdem ignoriert CVS seine eigenen Verwaltungsverzeichnisse, ohne dass man dies explizit irgendwo angeben muss. Möchte man an einer Stelle alle vorherigen Namen für ungültig erklären, so verwendet man dazu das Ausrufezeichen. Das kann beispielsweise sinnvoll sein, wenn man die Voreinstellungen des CVS-Servers nicht übernehmen möchte:

```
!
*.class
```

Zurücksetzen der Einstellungen

Diese Datei setzt alle vorher gemachten Einstellungen zurück und ignoriert dann alle **class**-Dateien. Doch Vorsicht: Das Zurücksetzen mittels des Ausrufezeichens bewirkt auch, dass CVS seine eigenen lokalen Verwaltungsverzeichnisse (siehe den folgenden Abschnitt 7.4, »Die CVS-Verzeichnisse in der lokalen Arbeitskopie«) von nun an nicht mehr ignoriert! Besser ist also:

```
!
CVS
*.class
```

7.4 Die CVS-Verzeichnisse in der lokalen Arbeitskopie

Nicht alle Informationen, die CVS zu seiner Arbeit benötigt, können sofort auf dem Server gespeichert werden. Solche zunächst lokal gespeicherten Informationen betreffen beispielsweise neu hinzugefügte Dateien (Befehl **add**) oder gelöschte Dateien (Befehl **remove**). CVS legt die zu verwaltenden Informationen in einem Verzeichnis **CVS** ab, das es als Unterverzeichnis in jedes Verzeichnis der Arbeitskopie einfügt. Normalerweise werden diese **CVS**-Verzeichnisse beim **Checkout** angelegt, allerdings bewirkt auch das Hinzufügen eines Verzeichnisses mit dem Befehl **add** die Anlage eines **CVS**-Ordners in dem neuen Verzeichnis.

Entries, Repository und Root

Innerhalb des Verzeichnisses **CVS** befinden sich mindestens drei Dateien: **Entries**, **Repository** und **Root**. Die Datei **Root** enthält die Information zum Zugriff auf das Repository (z.B. `:pserver:frank@192.168.0.7:/var/lib/cvs`), die Datei **Repository** enthält den relativen Verzeichnispfad von der Wurzel des Repositories und die Datei **Entries** enthält eine Liste aller Dateien und Verzeichnisse. Zu jeder Datei wird hier die lokale Revisionsnummer und der Zeitstempel des letzten Abgleichs mit dem Repository vermerkt.

Andere Dateien

Neben diesen drei immer vorhandenen Dateien, können temporär oder nach Eintreten bestimmter Umstände, noch andere Dateien in den lokalen CVS-Verzeichnissen gespeichert werden. Diese Dateien sind aber für den Entwickler nicht relevant, sie dienen nur der internen Verwaltung von CVS. Man sollte die lokalen CVS-Verzeichnisse mit den darin enthaltenen Dateien nicht entfernen und nicht manuell bearbeiten, da dann das einwandfreie Funktionieren des CVS-Clients nicht mehr gewährleistet ist. Möchte man eine Version seines Sourcecodes ohne CVS-Verzeichnisse erstellen, so bietet sich der Befehl **export** an, der genau dies bewirkt.

7.5 Module exportieren

7.5.1 Der Befehl export

Der Befehl **export** dient dazu ein Modul aus dem Repository auszuchecken, ohne dass CVS dies als lokale Arbeitskopie verwaltet. Der Befehl **export** ähnelt einem Checkout, besitzt aber zwei gravierende Unterschiede: CVS exportiert keine Verwaltungsinformationen, d.h. es werden keine CVS-Verzeichnisse im exportierten Modul angelegt. Damit

lässt sich das exportierte Modul auch nicht als lokale Arbeitskopie verwenden. Außerdem ist beim Export die Angabe eines Tags oder eines Datums zwingend; man kann nicht einfach den aktuellen Stand eines Moduls exportieren. Ein Beispiel:

```
cvs export -r Version1 TextPrinter1
```

erzeugt als Ausgabe:

```
cvs export: Updating TextPrinter1
U TextPrinter1/TextPrinter.java
U TextPrinter1/text.txt
```

In vielen Projekten wird der Befehl **export** nicht gebraucht. Man kann Software auch aus einer lokalen Arbeitskopie erstellen lassen. Vorsicht ist allerdings bei Internet-Projekten angesagt, hier können die CVS-Verzeichnisse kompromittierend sein (siehe Abschnitt 7.10, »Webseiten mit CVS verwalten«)!

7.5.2 Der Befehl export in WinCvs

In WinCvs und gCvs wird Export als ein Sonderfall von Checkout betrachtet. Man ruft einen Export daher über das Menü **Create · Checkout module...** auf. Im Optionen-Dialogfeld wechselt man auf den Reiter **Checkout options** und setzt dort ein Häkchen bei »Do not create CVS directories (export)«.

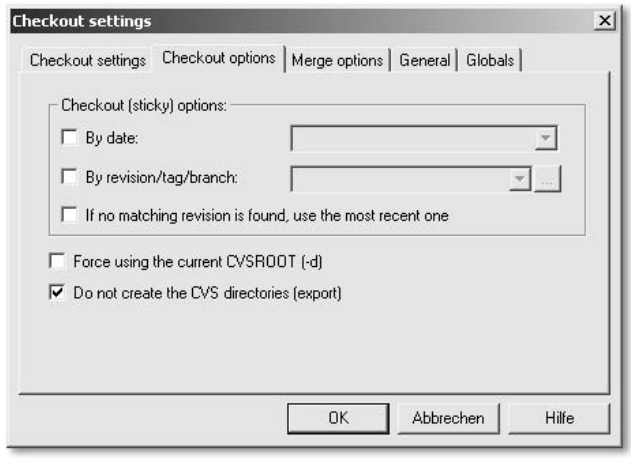

Abbildung 7.1 Export in WinCvs

Im Gegensatz zu **checkout** verlangt der Befehl **export** zwingend nach einem Tag oder einer Datumsangabe. Wenn man den letzten Stand aus dem Repository exportieren möchte, hat man nicht immer ein Tag gesetzt. Man kann in diesem Fall ein Datum angeben. Möchte man sich die umständliche Eingabe von Datum und Uhrzeit ersparen, so gibt man einfach »now« für das Datum an. Allerdings kann es auch dann noch Probleme mit der Uhrzeit geben, wenn die Uhrzeit auf dem lokalen Rechner und dem CVS-Server nicht überein stimmen. Besser man legt den Export in die Zukunft, dann sollte nichts mehr schief gehen:

```
cvs export -D tomorrow TextPrinter1
```


7.6 Dateien zeilenweise analysieren

7.6.1 Die Befehle annotate und rannotate

Manchmal möchte man in einem Projekt erfahren, wann sich bestimmte Änderungen ergeben haben. Man kann dazu den bereits beschriebenen Befehl **diff** verwenden, der die Unterschiede zwischen zwei Revisionen zeigt. Die Verwendung von **diff** kann allerdings recht mühsam werden, wenn es sehr viele Revisionen gibt und man nicht so genau weiß, in welcher Revision sich eine Änderung ergeben haben könnte. Für diesen Fall bietet CVS ein anderes Werkzeug an: die Befehle **annotate** und **rannotate**. Beide Befehle listen Dateien zeilenweise auf und geben dann für jede Zeile an, in welcher Revision die Zeile zuerst auftauchte. Dazu wird das zugehörige Datum ausgegeben. Während **annotate** auf Dateien der lokalen Arbeitskopie arbeitet, werkelt **rannotate** auf ganzen Modulen, die es direkt aus dem Repository holt. Daher ist für **rannotate** keine lokale Arbeitskopie notwendig. Ein Beispiel:

```
cvs annotate TextPrinter1/TextPrinter.java
```

gibt aus:

```
1.5   (frank    22-Sep-03): import java.io.BufferedReader;
1.5   (frank    22-Sep-03): import java.io.FileReader;
1.5   (frank    22-Sep-03):
         import java.io.FileNotFoundException;
1.5   (frank    22-Sep-03): import java.io.IOException;
1.5   (frank    22-Sep-03):
```

```
1.4   (frank    22-Sep-03): /**
1.4   (frank    22-Sep-03): *
1.4   (frank    22-Sep-03): * Das Programm TextPrinter liest
1.4   (frank    22-Sep-03): * die Textdatei test.txt ein und
1.4   (frank    22-Sep-03): * gibt diese zeichenweise wieder
1.4   (frank    22-Sep-03): * aus.
1.4   (frank    22-Sep-03): * Im Endausbau soll die Ausgabe
1.4   (frank    22-Sep-03): * verzögert werden, um einen
1.4   (frank    22-Sep-03): * altmodischen, langsamen
1.4   (frank    22-Sep-03): * Computer zu simulieren
1.1   (frank    19-Sep-03):
         public class TextPrinter extends Object
1.1   (frank    19-Sep-03): {
1.3   (frank    22-Sep-03):    /**
1.3   (frank    22-Sep-03):    * printText liest die
1.3   (frank    22-Sep-03):    * Textdatei ein und gibt sie
1.3   (frank    22-Sep-03):    * zeilenweise aus
1.3   (frank    22-Sep-03):    *
1.3   (frank    22-Sep-03):    */
1.1   (frank    19-Sep-03):   public void printText()
1.1   (frank    19-Sep-03):   {
1.1   (frank    19-Sep-03):     String text = readFile();
1.1   (frank    19-Sep-03):
1.1   (frank    19-Sep-03):
         for (int i=0; i<text.length(); i++)
1.1   (frank    19-Sep-03):    {
1.1   (frank    19-Sep-03):      printChar (text.charAt(i));
1.1   (frank    19-Sep-03):     }
1.1   (frank    19-Sep-03): }
```

Wie man sieht, wird jede nicht leere Zeile mit Revisionsnummer, Autor und Datum ausgegeben. Dabei werden immer die Daten der letzten Änderung angezeigt.

Der Befehl **annotate** ist eigentlich nur bei Textdateien sinnvoll, obwohl man bei neueren CVS-Version mit der Option –F auch die Ausführung auf Binärdateien erzwingen kann. Mit den Optionen –r und –D kann man die Änderungen bis zu einer Revision, eines Tags oder eines Datums anzeigen.

7.6.2 Der Befehl annotate in WinCvs

In WinCvs ruft man den Befehl **annotate** über den Menüpunkt **Query · Annotate...** auf. Es erscheint dann das in Abbildung 7.2 gezeigte Dialogfeld. Nach der Bestätigung mit OK zeigt WinCvs die Ausgabe mit dem in den Preferences gesetztem Editor an. Ist dort kein Editor gesetzt, so verweigert WinCvs die Ausgabe!

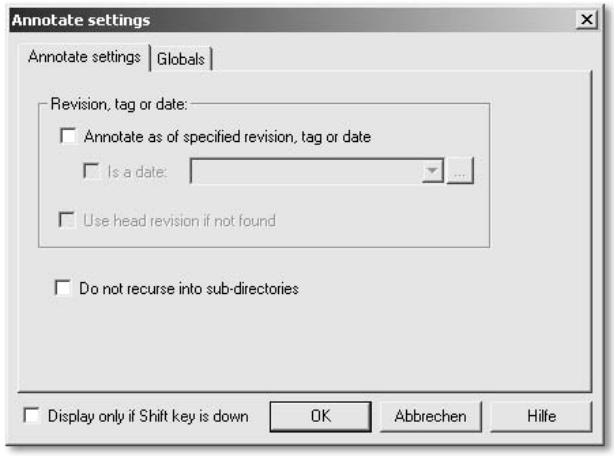

Abbildung 7.2 Annotate-Dialogfeld in WinCvs

7.7 Überwachtes Arbeiten

7.7.1 Der Ansatz anderer Versionsmanagementsysteme

Im Gegensatz zu vielen anderen Versionsmanagementsystemen folgt CVS dem Ansatz, dass jeder Entwickler seine private Kopie des Sourcecodes besitzt, mit der er zunächst einmal machen kann, was er möchte. Der Entwickler hat seine eigene »Spielwiese«, auf Englisch auch als *Sandbox* bezeichnet. Diese Umgebung ist zunächst in sich abgeschlossen, andere Entwickler bekommen die Arbeit eines Entwicklers erst nach einem Commit zu sehen und wissen vorher nicht, wer an welchen Dateien arbeitet. Eine ganze Reihe von Versionsmanagementsystemen verfolgen hier einen anderen Ansatz. Möchte ein Entwickler eine Datei bearbeiten, so sperrt er diese für alle anderen Entwickler. Die anderen Entwickler können dann die Datei so lange nicht bearbeiten, bis der Entwickler die Sperre wieder aufhebt. Dieses Verfahren hat den Nachteil, dass eine Datei nur von einem Entwickler zur Zeit bearbeitet werden kann. Außerdem kommt es zu Problemen, wenn ein Entwickler vergisst, eine Sperre wieder aufzuheben. Bei allen Nachteilen hat dieses

System allerdings den Vorteil, dass man durch Mechanismen des Versionsmanagementsystems herausfinden kann, wer gerade an welcher Datei arbeitet. Man muss sich einfach nur die gesperrten Dateien auflisten lassen und kann sehen, wer welche Datei gesperrt hat.

7.7.2 Sperren in CVS

Auch CVS erlaubt das Sperren von Dateien. Dies entspricht allerdings nicht der normalen Verwendungsweise von CVS. Es wird daher empfohlen, die Möglichkeit nicht zu verwenden. Wer sie trotzdem verwenden möchte, siehe dazu Abschnitt 7.17.1, »Das Sperren von Dateien«.

Watches

Die normale Arbeitsweise von CVS erlaubt es nicht festzustellen, wer an welchen Dateien arbeitet. Ist diese Information in einem Projekt allerdings erwünscht, so kann man den Entwicklungszyklus mit CVS leicht erweitern. CVS bietet zu diesem Zweck so genannte *Watches* an. Ein Watch beobachtet bestimmte Dateien. Ein Beispiel:

```
cvs watch add TextPrinter.java
```

Der Befehl gibt keine Bestätigung aus.

Der Befehl **watch** besitzt als einziger CVS-Befehl weitere »Unterbefehle«. Diese werden direkt nach dem Befehlswort **watch** angegeben. Es gibt vier dieser Unterbefehle: **add**, **remove**, **on** und **off**. Sie werden im Folgenden beschrieben.

Man kann sich die Beobachter der Datei mit

```
cvs watchers TextPrinter.java
```

ausgeben lassen:

```
TextPrinter.java        frank   edit    unedit  commit
```

Erweiterung des Entwicklungszyklus

Der CVS-Benutzer frank ist also ein »Watcher«, ein Beobachter der Datei **TextPrinter.java**. Er beobachtet die Aktionen edit, unedit und commit. Das bedeutet, dass frank vom CVS-Server eine E-Mail gesendet bekommt (der CVS-Server muss dazu entsprechend eingerichtet sein, siehe hierzu Abschnitt 8.4, »Den Server für überwachtes Arbeiten einrichten«), wenn ein anderer Benutzer einen der Befehle **edit**, **unedit** oder **commit** auf der Datei **TextPrinter.java** ausführt. Hier kommt die bereits erwähnte Erweiterung des Entwicklungszyklus ins Spiel. Damit Watches wirklich sinnvoll sind, muss ein Entwickler den Befehl **edit**

aufrufen, wenn er eine Datei zu bearbeiten gedenkt. Wenn er mit der Bearbeitung fertig ist, ruft er entweder **unedit** auf, wenn er seine Änderungen nicht ins Repository übernehmen möchte, oder **commit**, wenn die Änderungen ins Repository geschrieben werden sollen.

Ruft beispielsweise die Benutzerin elke den Befehl

```
cvs edit TextPrinter.java
```

auf, um anzukündigen, dass sie beabsichtigt, diese Datei zu bearbeiten, so bekommen alle Beobachter dieser Datei folgende Mail vom CVS-Server zugesandt:


```
Date: Wed, 10 Dec 2003 13:47:00 +0001
From: elke
To: frank
Subject: CVS notification

TextPrinter1 TextPrinter.java
---
Triggered edit watch on /var/lib/cvs/TextPrinter1
By elke
```

CVS versendet Mails

Die Mail wird sogar mit dem Absender von elke versendet. Der Text wird jedoch von CVS automatisch erstellt und folgt immer dem gleichen Schema. Es werden das Modul und die betroffene Datei gelistet. Danach stehen das ausgelöste Ereignis, die Position im Repository und schließlich die Person, die das Ereignis ausgelöst hat.

7.7.3 Nachteile von Watches

Watches haben zwei Nachteile: Sie erhöhen den Arbeitsaufwand des Entwicklers geringfügig, denn er muss sich jede Datei, die er ändern möchte, vorher durch den Aufruf des Befehls **edit** »freischalten« lassen. Der wohl schwer wiegendere zweite Nachteil ist, dass das überwachte Arbeiten nicht erzwungen werden kann. Wenn ein Entwickler den Befehl **edit** nicht aufruft, weil er es vergisst oder weil er es absichtlich nicht tun möchte, dann kann CVS nichts dagegen tun. Es werden in diesem Fall nur Benachrichtigungen verschickt, wenn der Entwickler den Befehl **commit** aufruft, um die Änderungen einer beobachteten Datei einzuchecken. Damit geht allerdings der Vorteil des überwachten Arbeitens verloren, da ja gerade eine Benachrichtigung verschickt werden soll, bevor eine beobachtete Datei verändert wird. Andere Entwickler könnten dann rechtzeitig auf eine solche Nachricht reagieren

und den die Datei bearbeitenden Entwickler vor möglichen Konflikten mit ihrer eigenen Entwicklungsarbeit warnen.

Schreibschutz als Erinnerung

Damit die Verwendung des Befehls **edit** nicht nur aus Vergesslichkeit der Entwickler unterbleibt, hat CVS einen kleinen Mechanismus zur Erinnerung eingebaut. Mit dem Befehl

```
cvs watch on TextPrinter1
```

wird dieser Mechanismus für das Verzeichnis **TextPrinter1** mit allen seinen Dateien angeschaltet. Von nun an checkt CVS alle Dateien dieses Verzeichnisses mit gesetztem Schreibschutz aus. Erst der Befehl **edit** hebt den Schreibschutz auf. Die Befehle **unedit** und **commit** setzen den Schreibschutz hingegen wieder. Durch den gesetzten Schreibschutz vergisst der Entwickler nicht, dass er den Befehl **edit** aufrufen muss. Natürlich könnte der Entwickler den Schreibschutz auch mit den Mitteln des Betriebssystems zurücksetzen. Daran kann man ablesen, dass das überwachte Arbeiten von den Entwicklern eines Teams gewollt sein muss. Alle müssen konstruktiv mitarbeiten. Wer das System unterlaufen möchte, der kann es auch. Die Mitarbeit kann in diesem Punkt nicht erzwungen werden.

Den Schreibschutz wieder abschalten

Möchte man den Schreibschutzmechanismus wieder abschalten, so gibt man ein:

```
cvs watch off TextPrinter1
```

Damit ist der Schreibschutzmechanismus für das Verzeichnis **TextPrinter1** mit seinen Dateien wieder abgeschaltet. Allerdings nimmt CVS in einem solchen Fall den bestehenden Schreibschutz von Dateien nicht weg; dies muss man manuell mit den Bordmitteln des Betriebssystems machen. Man kann auch ohne den Schreibschutzmechanismus mit Watches arbeiten, es ist dann nur etwas mehr Disziplin gefordert.

Sinnvollerweise verwendet man den Befehl **watch on** auf ganzen Modulen oder Projekten. Zwar lässt sich der Befehl auch dateiweise anwenden, jedoch ist es für die betroffenden Entwickler dann kaum noch nachvollziehbar, weshalb der Schreibschutz bei einigen Dateien gesetzt ist, bei anderen dagegen nicht.

7.7.4 Ereignisse beim überwachten Arbeiten

Wie eingangs bereits beschrieben, lassen sich beim überwachten Arbeiten drei Ereignisse unterscheiden, für die jeweils eine eigene Benachrichtigung ausgelöst werden kann: edit, unedit und commit. Die drei Ereignisse entsprechen dem Aufruf der drei gleichnamigen Befehle. Mit dem Aufruf des Befehls **edit** kündigt der Entwickler eine Veränderung einer Datei an. Durch den Aufruf des Befehls **unedit** tut der Entwickler kund, dass er die Veränderung der Datei abgebrochen hat und die Änderungen verwirft. Der Aufruf von **commit** bedeutet schließlich, dass der Entwickler eine bereits angekündigte Änderung vollzogen hat.

Überwachung einzelner Ereignisse

CVS bietet die Möglichkeit, sich nur bei bestimmten Ereignissen benachrichtigen zu lassen. Die Voreinstellung ist allerdings, dass man Mails zu allen Ereignissen erhält. So führt der Befehl

```
cvs watch add TextPrinter.java
```

dazu, dass man Benachrichtigungen zu den Ereignissen edit, unedit und commit auf der Datei **TextPrinter.java** erhält. Möchte man hingegen nur von bestimmten Ereignissen unterrichtet werden, so kann man diese explizit beim Aufruf des Befehls **watch add** mit der Option –a angeben. So fügt man mit

```
cvs watch add -a edit TextPrinter.java
```

eine Benachrichtigung nur für das Ereignis edit hinzu. Beim Aufruf der Befehle **unedit** und **commit** erfolgt in diesem Fall keine Benachrichtigung.

Angabe mehrerer Ereignisse

Man kann die Option –a auch mehrfach angeben, um sich bei mehreren Ereignissen benachrichtigen zu lassen:

```
cvs watch add -a edit -a unedit TextPrinter.java
```

Nach Eingabe dieses Befehls werden Benachrichtigungen beim Aufruf der Befehle **edit** und **unedit** versandt, nicht jedoch bei **commit**. Mehrfaches Aufrufen des Befehls **watch add** führt dazu, dass noch nicht beobachtete Ereignisse hinzugefügt werden. Wird ein bereits gesetztes Ereignis nochmals gesetzt, so hat das keinen weiteren Effekt, der Befehl wird einfach ignoriert.

Überwachung aufheben

Möchte man eine Datei nicht mehr beobachten, so entfernt man die Datei mit dem Befehl **watch remove** aus der Liste der zu beobachtenden Dateien:

```
cvs watch remove TextPrinter.java
```

Genau wie der Befehl **watch add** hat auch **watch remove** keine Ausgabe.

Dieser Aufruf entfernt die Datei **TextPrinter.java** aus der Überwachung. Wie der Befehl **watch add**, so unterstützt auch der Befehl **watch remove** die Option –a, um die Benachrichtigungen für einzelne Ereignisse gezielt zu löschen:

```
cvs watch remove -a commit TextPrinter.java
```

Dies entfernt nur die Benachrichtigung für den Aufruf des Befehls **commit**. Alle anderen gesetzten Benachrichtigungen bleiben bestehen.

7.7.5 Temporäre Beobachter

CVS ist so programmiert, dass es Benachrichtigungen, die ein Entwickler auslöst, an diesen Entwickler selbst nicht versendet. Schließlich weiss der Entwickler selbst, dass er die betreffende Datei verändert hat und muss daher nicht benachrichtigt werden. Daneben besitzt der Benachrichtigungsmechanismus von CVS eine weitere Eigenheit, die etwas weniger offensichtlich ist: Führt ein Entwickler den Befehl **edit** auf einer Datei aus, so wird er temporär ebenfalls auf die Liste der Beobachter dieser Datei gesetzt, das heißt, CVS verhält sich so, als wenn der Entwickler zusätzlich auch **watch add** auf dieser Datei ausgeführt hätte. Im Unterschied zu diesem Befehl ist die durch **edit** ausgelöste Überwachung allerdings nur vorübergehend; der Aufruf der Befehle **unedit** oder **commit** hebt sie nämlich wieder auf. Hintergrund dieses Verhaltens von CVS ist die Annahme, dass ein Entwickler, der eine Datei bearbeitet, auch von Veränderungen an dieser Datei durch andere informiert werden möchte. Möchte der Entwickler allerdings dauerhaft Benachrichtigungen zu einer Datei erhalten, so ist es besser, den Befehl **watch add** manuell aufzurufen, da dann ein Aufruf des Befehls **commit** die Benachrichtigungen nicht gleich wieder abstellt. Die temporäre Beobachtung von Dateien lässt sich ebenfalls an der Ausgabe des Befehls **watchers** ablesen:

```
TextPrinter.java
   elke   tedit  tunedit  tcommit
   frank  edit   unedit   commit   tedit  tunedit  tcommit
```

Kennzeichnung von temporären Beobachtungen

Alle temporären Beobachtungen werden durch den Buchstaben »t« eingeleitet. Die Ausgabe bedeutet, dass elke und frank beide temporär alle Ereignisse auf der Datei **TextPrinter.java** beobachten, weil sie **edit** auf dieser Datei aufgerufen haben. frank hat jedoch zusätzlich eine permanente Beobachtung auf der Datei angefordert, die separat ausgegeben wird.

7.7.6 Die Befehle watchers and editors

CVS besitzt zwei Befehle, die der Statusabfrage von beobachteten Dateien dienen: **watchers** und **editors**. Der Befehl **watchers** zeigt an, wer eine Datei beobachtet. Ein Beispiel:

```
cvs watchers TextPrinter.java
```

gibt aus:

```
TextPrinter.java
       elke    edit
       frank   edit    unedit  commit
```

Die Ausgabe bedeutet, dass frank und elke die Datei **TextPrinter.java** beobachten. Während frank alle Ereignisse überwacht, beobachtet elke nur den Aufruf des Befehls **edit**.

Der Befehl **editors** zeigt an, wer den Befehl **edit** für eine Datei aufgerufen hat und diese Datei bearbeitet. Ein Beispiel:

```
cvs editors TextPrinter.java
```

gibt aus:

```
TextPrinter.java
elke   Wed Dec 10 09:51:59 2003 GMT    hal
   H:\TextPrinter1
frank  Wed Dec 10 09:57:15 2003 GMT    zappa
   /home/frank/TextPrinter1
```

Anzeige der Bearbeiter

In diesem Fall gibt es tatsächlich zwei Bearbeiter der Datei **TextPrinter.java**. Es handelt sich dabei wieder um die Benutzer elke und frank. Der Befehl **editors** gibt aber nicht nur die Benutzernamen aus, sondern verrät außerdem noch, wann der Benutzer den Befehl **edit** aufgerufen hat, wie der Rechner des Benutzers heißt und in welchem Verzeichnis der Benutzer arbeitet!

7.7.7 Der Befehl unedit

Möchte man das Bearbeiten einer Datei abbrechen, also den vorherigen Aufruf des Befehls **edit** rückgängig machen, so ruft man den Befehl **unedit** auf. Dabei ist allerdings Vorsicht geboten, wenn die Datei in der lokalen Arbeitskopie modifiziert worden ist. CVS geht in diesem Fall nämlich davon aus, dass der ursprüngliche Zustand wiederhergestellt werden soll. Es fragt daher nach, ob es die lokalen Änderungen rückgängig machen soll:

```
TextPrinter.java has been modified; revert changes?
```

unedit macht lokale Änderungen rückgängig!

Beantwortet man diese Frage mit nein (»n« eintippen), so wird die Datei von CVS zwar nicht verändert, allerdings wird **unedit** auch nicht ausgeführt! Möchte man also die lokalen Änderungen behalten und trotzdem den Abbruch der Bearbeitung signalisieren, so muss man sich eine Sicherungskopie der Datei außerhalb der Arbeitskopie anlegen und dann auf die Frage von CVS mit ja (»y« eintippen) antworten. Danach kann man seine Sicherungskopie zurückkopieren.

Es sind eine Reihe von CVS-Versionen im Umlauf, bei denen die meisten Befehle im Zusammenhang mit Watches nicht funktionieren. Der Fehler wurde in der Version 1.11.3 behoben. Sollten Sie Fehlermeldungen der Art

```
cvs [server aborted]: unknown command: watch_add
```

erhalten, so ist es wahrscheinlich, dass Sie eine solche fehlerhafte Serverversion installiert haben. In diesem Fall hilft nur ein Update auf eine neuere Version! Um die installierte Version herauszufinden, gibt man ein:

```
cvs version
```

eine mögliche Ausgabe:

```
Client: Concurrent Versions System (CVS) 1.11.5 (client)
Server: Concurrent Versions System (CVS) 1.11.1p1
   (client/server)
```

In diesem Fall ist ein Update angeraten, sofern man mit Watches arbeiten möchte!

7.7.8 Überwachtes Arbeiten in WinCvs

WinCvs gruppiert alle zum überwachten Arbeiten benötigten Befehle zusammen in das Menü **Trace**. Es gibt auch eine eigene Symbolleiste für das überwachte Arbeiten (siehe Anhang E). Allerdings werden die Befehle **watch on** und **watch off** von WinCvs nicht direkt unterstützt, man muss diese direkt in das Ausgabefenster eingeben.

7.8 Umgebungsvariablen

CVS wertet eine ganze Reihe von Umgebungsvariablen aus, um daraus Konfigurationsinformationen zu entnehmen. Insbesondere bei der Verwendung der Kommandozeilen-Clients kann es sinnvoll sein, einige der von CVS ausgewerteten Umgebungsvariablen zu setzen, um sich die Arbeit mit CVS zu vereinfachen und die Konfiguration den eigenen Bedürfnissen anzupassen. In den GUI-Clients lassen sich viele Optionen auch direkt einstellen, ohne dass dafür Umgebungsvariablen gesetzt werden müssten. Daher sind die Umgebungsvariablen bei GUI-Clients meist weniger relevant.

7.8.1 Die Umgebungsvariable CVSROOT

Die Umgebungsvariable `CVSROOT` ist bereits in Abschnitt 4.1.1 »CVS-Kommandozeile unter Windows« beschrieben worden. Mit dieser Variablen lässt sich die CVSROOT angeben, so dass sie nicht mit der Option –d angegeben werden muss. Das kann bei häufiger Verwendung der Befehle **checkout** und **login** einige Tipparbeit sparen.

7.8.2 Den Editor vorgeben

Ebenfalls mit einer Umgebungsvariablen lässt sich der Editor einstellen, den CVS zur Eingabe von Log Messages aufrufen soll. CVS wertet hier gleich drei Umgebungsvariablen aus: `EDITOR`, `VISUAL` und `CVSEDITOR`. Sind mehrere dieser Variablen gesetzt, dann hat die Variable `CVSEDITOR` eine höhere Priorität als `VISUAL`, und `VISUAL` hat eine höhere Priorität als `EDITOR`. Die Variable `CVSEDITOR` sollte man setzen, wenn man speziell einen Editor für CVS bestimmen möchte, denn die Variablen `EDITOR` und `VISUAL` werden teilweise auch von anderen Programmen ausgewertet. Wenn keine der drei Variablen gesetzt sind, dann verwendet CVS auf Unix den Editor vi und auf Windows den Notepad.

7.8.3 Das Home-Verzeichnis

Wichtig auf Unix-Systemen ist die Umgebungsvariable `HOME`. Diese wird von CVS ausgewertet, um im angegebenen Verzeichnis lokale Konfigurationsdateien wie **.cvspass** und **.cvsignore** zu suchen und zu speichern. Da diese Umgebungsvariable auf Unix-Systemen normalerweise gesetzt ist, muss man sich nicht um sie kümmern. Unter Windows bestimmt CVS den Speicherort der lokalen Konfigurationsdateien aus den beiden Umgebungsvariablen `HOMEDRIVE` und `HOMEPATH`. Erst beide zusammen genommen ergeben der Speicherort. Setzt man

```
HOMEDRIVE=c:
```

und

```
HOMEPATH=\Dokumente und Einstellungen\frank\Eigene Dateien
```

dann speichert CVS die lokalen Konfigurationsdateien in **c:\Dokumente und Einstellungen\frank\Eigene Dateien**. Wenn mehrere Entwickler an einem Rechner mit CVS arbeiten, so sollte man die beiden Variablen `HOMEDRIVE` und `HOMEPATH` benutzerspezifisch setzen. Dies macht man im oberen Teil des Windows-Dialogfeldes »Umgebungsvariablen«, das in Abbildung 7.3 gezeigt wird. Man ruft das Dialogfeld aus der Systemsteuerung unter **System · Erweitert · Umgebungsvariablen** auf.

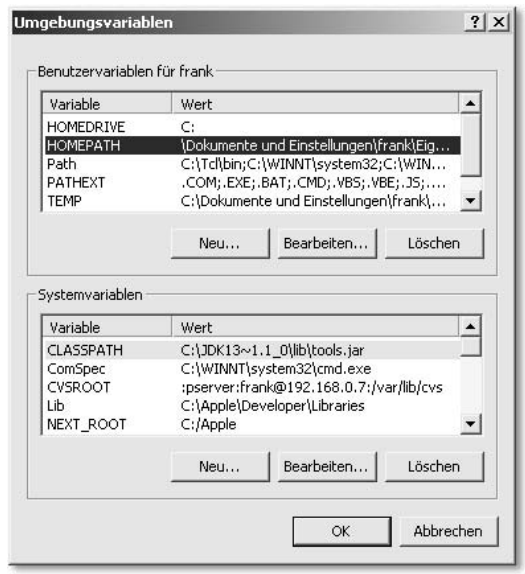

Abbildung 7.3 HOMEDRIVE und HOMEPATH

Setzt man die Umgebungsvariablen `HOMEDRIVE` und `HOMEPATH` nicht, so verwendet CVS das Hauptverzeichnis des Windows-Laufwerks als Speicherort für die Konfigurationsdateien, oft also **c:**.

7.8.4 Die Umgebungsvariable CVSIGNORE

Schließlich sei noch mal kurz die Möglichkeit genannt, auch zu ignorierende Dateien per Umgebungsvariable zu bestimmen. Die dazugehörige Umgebungsvariable heißt `CVSIGNORE`. Dies wurde bereits in Abschnitt 7.3 »Dateien ignorieren: cvsignore« beschrieben.

7.9 XML, HTML und CVS

CVS ist als ein Versionsmanagementsystem entwickelt worden, das in erster Linie Textdateien verwalten soll. Zwar kann CVS auch binäre Dateien im Repository speichern, dann gibt es allerdings bei diesen keine Möglichkeit, die mächtigen Merging-Fähigkeiten von CVS zu nutzen. Ändern zwei Entwickler gleichzeitig eine binäre Datei, so gibt es immer einen Konflikt.

7.9.1 Besonderheiten von XML und HTML

HTML- und XML-Dateien sind Textdateien und können als solche auch durch CVS verwaltet werden. Allerdings gibt es bei diesen eine Besonderheit: Sowohl HTML- als auch XML-Dateien bestehen aus einer Reihe von Tags mit dazwischenliegendem Text. Die meisten Tags bestehen aus einem öffnenden und aus einem schließenden Teil. Zwischen öffnendem und schließendem Teil können weitere Tags oder Text eingebettet sein. Einige Tags einer HTML- oder XML-Datei werden immer sehr weit »außen« stehen, d.h. sie werden sehr viele weitere Tags einschließen. Bei einer HTML-Datei trifft das beispielsweise für das HTML- und das BODY-Tag zu. Nimmt man nun strukturelle Änderungen an einer HTML- oder XML-Datei vor, beispielsweise indem man direkt nach dem BODY-Tag ein TABLE-Tag einfügt, so wird CVS immer den gesamten Bereich zwischen öffnendem Teil und schließendem Teil des Tags als verändert erkennen. Dies kann große Teile der Datei betreffen.

7.9.2 Merging-Algorithmus in CVS

Der Algorithmus in CVS, der das Merging von verschiedenen Versionen einer Textdatei durchführt, ist für die Syntax gängiger Programmiersprachen implementiert worden. In fast allen Programmierspra-

chen sind Änderungen meist lokal sehr begrenzt, wenn sie logisch zusammengehörend sind. Aus diesem Grund ist das Merging von HTML- und XML-Dateien oft problematisch, denn strukturelle Änderungen an HTML- und XML-Dateien entsprechen nicht dem »normalen Muster« von Änderungen an Sourcecode-Dateien gängiger Programmiersprachen. Bei HTML- und XML-Dateien sind die veränderten Bereiche potenziell größer und überschneiden sich leichter. Daher kann es bei HTML- und XML-Dateien deutlich schneller zu Konflikten kommen als bei üblichen Sourcecode-Dateien. Als Konsequenz aus diesem Verhalten sollte man bei HTML- und XML-Dateien Änderungen noch besser als bei der Softwareentwicklung kommunizieren und häufige Updates und Commits durchführen. Hier ist auch ein Fall gegeben, in dem sich das überwachte Arbeiten anbietet (siehe Abschnitt 7.7 »Überwachtes Arbeiten«).

7.10 Webseiten mit CVS verwalten

Zusätzlich zu dem zu XML und HTML bereits angesprochenen Punkten, sollte man einen weiteren Punkt im Auge behalten, wenn man Webseiten mit CVS verwaltet: Webseiten sollten immer per **export**-Befehl aus CVS exportiert werden, bevor man sie auf den Webserver aufspielt. Man sollte niemals direkt eine lokale Arbeitskopie dazu verwenden! Der Grund für dieses Vorgehen liegt in den Verwaltungsinformationen, die sich in der lokalen Arbeitskopie befinden. Spielt man die lokale Arbeitskopie direkt auf dem Webserver ein, so stellt man auch die CVS-Verwaltungsinformationen online! Ein CVS-Kenner kann dann leicht Informationen erhalten, die eigentlich gar nicht herausgegeben werden sollten!

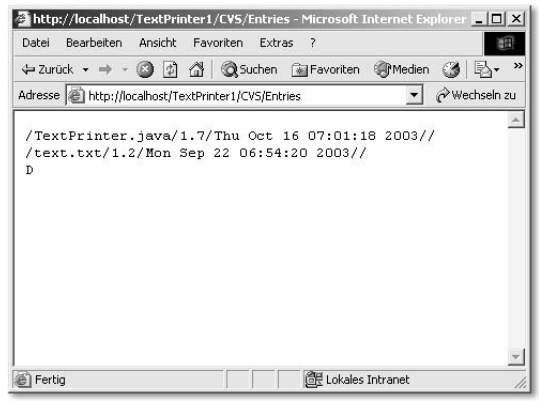

Abbildung 7.4 Entries-Datei im Internet-Explorer

In der **Entries**-Datei werden alle Dateien des Projektes aufgelistet, also ggf. auch solche, die Online (noch) gar nicht verlinkt sind. In der **Root**-Datei wird die CVS-Root gespeichert. Hier könnte unvorsichtigerweise sogar das CVS-Passwort stehen!

7.11 Vendor Branches

7.11.1 Einbindung von fremder Software

In Abschnitt 6.1, »Revisionen und Releases«, sind Vendor Branches bereits im Zusammenhang mit dem Import von Sourcecode in das Repository erwähnt worden. An dieser Stelle soll nun erklärt werden, was es mit den Vendor Branches auf sich hat. Vendor Branches dienen dem Zweck, den Sourcecode Dritter in das Versionsmanagement einzubinden. Der Lieferant dieses Sourcecodes wird als Vendor (Verkäufer) bezeichnet. Weshalb sollte man Sourcecode Dritter in das eigene Versionsmanagement aufnehmen? Ein Fall könnte sein, dass man ein eigenes Projekt in Teilen auf einem Open Source-Projekt aufgebaut hat und auch dieses laufend weiterentwickelt wird. Um die eigene Arbeit und die Änderungen an dem Open Source-Projekt zusammenzuführen, bietet sich CVS natürlich an. Ein anderer Fall könnte sein, dass man Programmbibliotheken eines anderen Herstellers einsetzen möchte und diese in Form von Sourcecode ausgeliefert werden. Somit kann man die ausgelieferten Bibliotheken selbst verändern. Irgendwann wird der Hersteller eine neue Version seiner Bibliotheken ausliefern. Auch in diesem Fall bietet es sich an, Vendor Branches einzusetzen, und CVS zur Zusammenführung und Verwaltung des Sourcecodes zu verwenden.

CVS legt immer einen Vendor Branch an

CVS arbeitet grundsätzlich so, als wenn der erste Import eines Projekts fremden Sourcecode enthalten würde und legt einen Vendor Branch, also eine Verzweigung für diesen fremden Sourcecode an. In dieser Verzweigung wird der unmodifizierte fremde Sourcecode gespeichert. Man kann das daran erkennen, dass nach einem Import alle importierten Dateien die Revisionsnummer 1.1.1.1 tragen. CVS reserviert nämlich den Zweig mit der Revisionsnummer 1.1.1 (die Revisionsnummer eines Zweigs ist immer dreistellig) für den Vendor Branch. Erst wenn man eine Datei verändert und dann eincheckt, wechselt die Revisionsnummer der Datei von 1.1.1.1 auf 1.2. CVS sorgt automatisch dafür, dass man nicht auf der Verzweigung, sondern auf dem Hauptzweig weiterarbeitet. Mit dem Wissen um Vendor Branches im Hinterkopf, kann man nun auch erkennen, was es mit den beiden Tags auf sich hat,

die man beim Import angeben muss! Das erste Tag (Vendor Tag) bezeichnet die Verzweigung, in der die fremde Software abgelegt wird. Das zweite Tag (Release Tag) kennzeichnet das Release des fremden Sourcecodes.

7.11.2 Die Arbeit mit Vendor Branches

Wie geht man vor, wenn man Vendor Branches verwenden möchte? Im ersten Schritt ändert sich kaum etwas zu der bisherigen Vorgehensweise. Man nimmt den Sourcecode des Drittanbieters und importiert diesen ganz normal in das Repository. Allerdings sollte man Vendor Tag und Release Tag mit sinnvollen Werten belegen. Als Beispiel soll ein Softwarepaket »TextPro« einer Firma »TextTools« dienen. Als Basis für ein eigenes Projekt dient die Version 1 dieser Software. Der Import könnte daher so aussehen (in einer Zeile):

```
cvs import -m "Neues Projekt"
   TextPrinter2 TextTools TextPro_1_0
```

Erneuter Import

Die Verzweigung wird mit »TextTools« benannt, die zugrunde liegende Version der Bibliothek ist »TextPro 1.0« und wird daher als Release Tag angegeben. Interessant wird es, wenn die Firma »TextTools« irgendwann eine neue Version ihrer Bibliothek veröffentlicht und diese in das eigene Projekt übernommen werden soll. Man führt in diesem Fall einen weiteren Import mit dem neuen Sourcecode durch. Er unterscheidet sich nur in der Angabe des Release Tags von dem ersten Import:

```
cvs import -m "Neue Version TextPro"
   TextPrinter2 TextTools TextPro_2_0
```

Konflikte beim Import

Der einzige Unterschied – mit Ausnahme der Log Message – ist die Angabe des Release Tags »TextPro_2_0«. Nun kann es natürlich sein, dass während der Entwicklung des Fremdpaketes und der eigenen Softwareentwicklung an gleichen Dateien gearbeitet worden ist. Es kann also zu Konflikten kommen. Und so gibt CVS beispielsweise auf den obigen Befehl aus:

```
C TextPrinter2/text.txt

1 conflicts created by this import.
Use the following command to help the merge:

cvs checkout -j<prev_rel_tag> -jTextPro_2_0 TextPrinter2
```

CVS macht einen Vorschlag!

Um den Konflikt aufzulösen, müssen nun zunächst einmal die normalen Merging-Mechanismen von CVS zur Anwendung kommen. Beim Import versucht CVS nämlich nicht Dateien zusammenzuführen, sondern wertet jede Veränderung der Datei, die sowohl im Repository als auch in der zu importierenden Datei erfolgt ist, als Konflikt. Daher sollte man nun erst einmal CVS anweisen, die Dateien zusammenzuführen. CVS schlägt sogar den dafür passenden Befehl vor! Man gibt also ein:

```
cvs checkout -jTextPro_1_0 -jTextPro_2_0 TextPrinter2
```

und CVS gibt aus:

```
cvs server: Updating TextPrinter2
U TextPrinter2 /text.txt
RCS file: /var/lib/cvs/TextPrinter2/text.txt,v
retrieving revision 1.1.1.1
retrieving revision 1.1.1.2
Merging differences between 1.1.1.1 and 1.1.1.2 into
   text.txt
```

CVS hat also die Änderungen an der Datei **text.txt** selbst zusammenführen können! Es kann in einem solchen Fall natürlich auch zu echten Konflikten kommen, diese müssen dann wie gewohnt aufgelöst werden.

Mehr als ein Vendor Branch

Man kann auch mit mehr als einem Vendor Branch arbeiten, man muss diese beim Import dann allerdings mit der Option –b angeben. Mehrere Vendor Branches benötigt man dann, wenn man seine eigene Arbeit auf mehr als einem fremden Softwarepaket aufsetzt.

Vendor Branches stellen einen hilfreichen Mechanismus zur Arbeit mit Sourcecode aus fremden Quellen dar. Etwas ungewöhnlich ist, dass CVS jedes importierte Projekt in einer solchen Verzweigung ablegt. Vendor Branches führen daher – neben ihrer eigentlichen Funktion – oft zur Verwirrung von CVS-Neulingen, die sich über die »Merkwürdigkeiten« beim Projektimport wundern.

7.12 CVSWeb und ViewCVS

7.12.1 Das Original: CVSWeb

Wer anonymen, lesenden Zugriff auf das Repository seines CVS-Servers gestatten möchte, der muss nicht unbedingt den in Abschnitt 8.5, »Ano-

nymer Zugriff auf das Repository«, beschriebenen anonymen Zugang einrichten. Es ist auch möglich, lesenden Zugriff über ein Web-Interface anzubieten. Dazu muss man lediglich das Zusatzprogramm CVSWeb installieren, das mit einem CVS-Server auf Unix zusammenarbeitet. CVSWeb stellt den Inhalt eines Repositories in einer einfach zu bedienenden HTML-Oberfläche dar. Man kann sich Dateien ansehen, ins lokale Dateisystem kopieren, alte Revisionen abrufen und sich die Differenzen zwischen einzelnen Revisionen anzeigen lassen. Der vielleicht größte Vorteil von CVSWeb ist, dass es zu all diesen Operationen keines Auscheckens bedarf. Man klickt sich einfach durch die Verzeichnisse des Repositories. Das macht CVSWeb auch für Entwickler interessant, die sich ein Projekt ansehen möchten, ohne es auschecken zu müssen. Installiert man CVSWeb auf einem Webserver, der vom Internet aus erreichbar ist, so ermöglicht das weltweiten lesenden Zugriff auf das Repository. Dies ist gerade für Open Source-Projekte interessant, die zwar jedem den Einblick in den aktuellen Sourcecode ermöglichen wollen, die Änderungen daran aber nur vorher beim Projekt registrierten Entwicklern gestatten möchten. CVSWeb bietet hier den Vorteil, dass kein Client-Programm installiert werden muss, um auf den Sourcecode zuzugreifen.

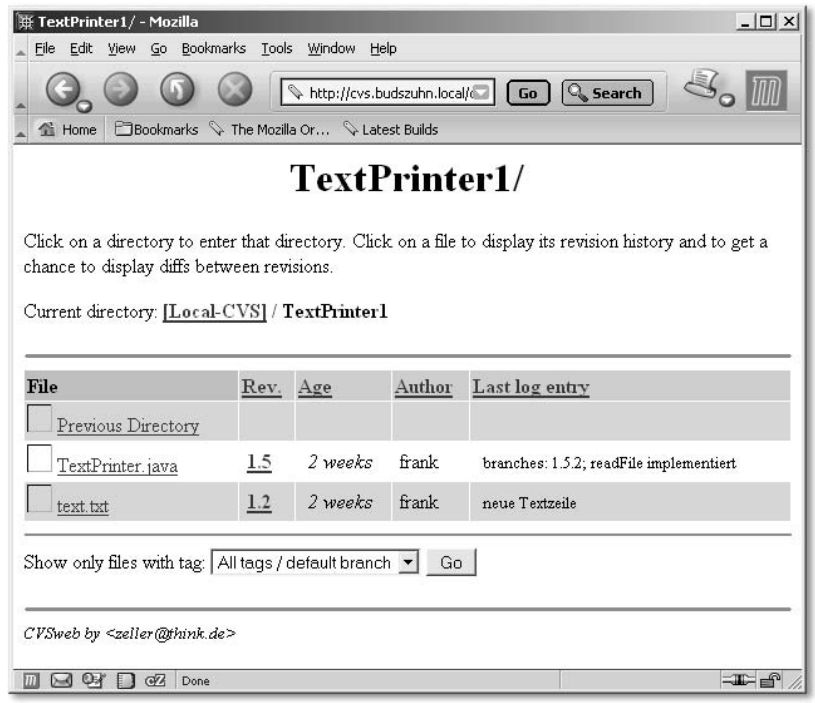

Abbildung 7.5 Das Modul TextPrinter1 in CVSWeb

Installation unter Debian

Die Installation von CVSWeb ist einfach. Auf einem Debian-System gibt man dazu lediglich

```
apt-get install cvsweb
```

ein. Auf anderen Unix-Systemen muss CVSWeb gegebenenfalls durch die dortige Paketverwaltung installiert werden.

Konfiguration

Nach der Installation ist zumindest eine Änderung an der Konfiguration von CVSWeb vorzunehmen. CVSWeb besitzt eine Konfigurationsdatei. Sie liegt normalerweise im Verzeichnis **/etc** und heißt **cvsweb.conf**. Dort sucht man folgende Stelle:


```
# 'symbolic_name' 'path_to_the_actual_repository'
%CVSROOT = (
# Uncomment next line and modify the path
# if you have only one CVS repository.
           'Local-CVS' => '/var/lib/cvs',
#          'Development' => '/usr/local/src/cvsrep',
#          'Configuration' => '/tmp/cvsroot/conf',
#          'HTML-files' => '/tmp/upload'
           );
```

Bei »Local-CVS« sollte man das Kommentarzeichen entfernen und den eigenen Pfad zum Repository eintragen. Besitzt der eigene CVS-Server mehr als ein Repository, so kann man die weiteren Repositories wie in der Beispieldatei gezeigt angeben.

CGI-Programm

Meist funktioniert CVSWeb bereits nach dieser einen Änderung in der Konfigurationsdatei. Da CVSWeb als CGI-Programm arbeitet, ruft man es einfach mit der URL des Webservers, dem Namen des CGI-Verzeichnisses und dem Namen von CVSWeb selbst auf:

http://www.meinserver.de/cgi-bin/cvsweb

Manchmal wird auch der Name **cvsweb.cgi** für das Programm verwendet.

In der Konfigurationsdatei zu CVSWeb lässt sich die HTML-Darstellung des Programms mittels vieler Parameter konfigurieren. Auf diese Art ist es möglich, dem Programm das eigene »Look & Feel« zu geben.

7.12.2 Die Alternative: ViewCVS

Als Alternative zu CVSWeb gibt es das Programm ViewCVS, dessen Zielsetzung, Funktionsweise und Aussehen dem Programm CVSWeb stark ähneln. ViewCVS ist letztlich eine Neuimplementierung von CVSWeb in einer anderen Programmiersprache (Python statt Perl).

Installation unter Debian

Auf einem Debian-System ist die Installation von ViewCVS sogar noch einfacher als die von CVSWeb:

```
apt-get install viewcvs
```

Das Debian-Installationsprogramm fragt alle notwendigen Informationen zur Konfiguration des Programms ab. Dazu gehören das Verzeichnis des Repositories, die E-Mail-Adresse des CVS-Administrators und die Frage, ob man sich von ViewCVS tar-Archive erzeugen lassen möchte. Diese Option ist sehr praktisch, da man sich so über den Browser ein fertig gepacktes Projektarchiv downloaden kann, ein Auschecken ist nicht notwendig!

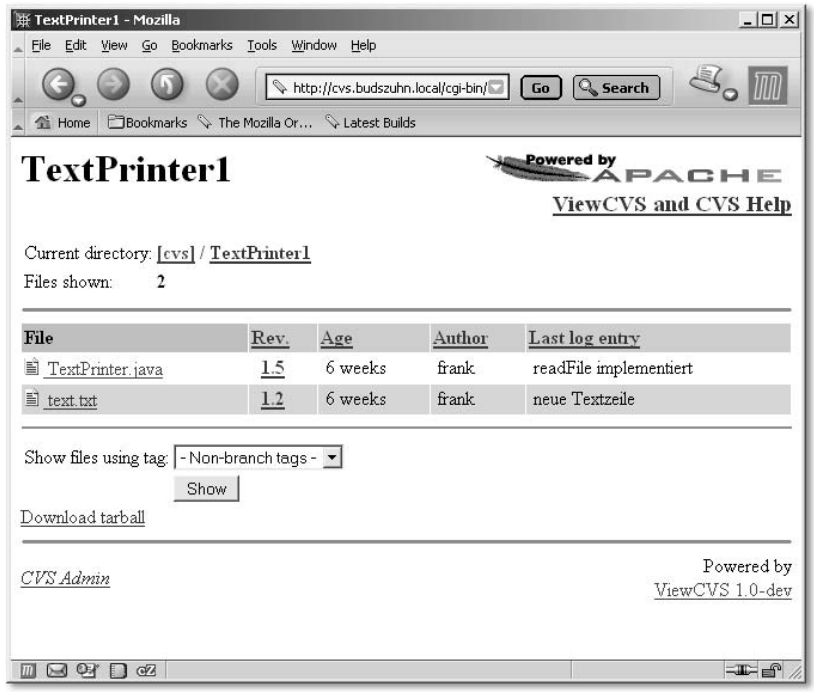

Abbildung 7.6 Das Modul TextPrinter1 in ViewCVS

ViewCVS vs. CVSWeb

Was die Anzahl der Funktionen anbelangt, so hat ViewCVS einen kleinen Vorsprung vor CVSWeb. Allerdings besitzt zum Zeitpunkt der Abfassung dieses Buchs CVSWeb (noch) die weitere Verbreitung. Den Grundfunktionen zur Anzeige eines Repositories über das Web werden beide gerecht und ähneln sich dabei weitgehend in ihrer Bedienung, so dass ein späterer Wechsel zwischen den Programmen nicht schwer fallen dürfte.

7.13 Typische CVS-Probleme und deren Lösung

CVS ist im Allgemeinen ein zuverlässiges und robustes System, das selten zu Totalausfällen neigt. CVS-Server sind oft jahrelang im Betrieb, ohne dass sie größerer Wartung bedürfen. Durch das – für ein Softwaresystem – schon recht hohe Alter ist CVS eine schon seit Jahren ausgereifte Software. Die meisten Schwächen von CVS sind konzeptioneller Natur, einige von ihnen werden in Kapitel 9, »Die Zukunft von CVS«, aufgeführt. An dieser Stelle soll es jedoch nicht um konzeptionelle Probleme gehen, sondern um die kleinen »Macken«, die einem beim Umgang mit CVS im Alltag begegnen können.

7.13.1 CVS und Zugriffsrechte

CVS ist generell schwach bei der Handhabung von Zugriffsrechten. CVS selbst kennt keine eigene Verwaltung von Zugriffsrechten, dies wird an das Betriebssystem delegiert. Oft wird man feststellen, dass CVS in der lokalen Arbeitskopie gesetzte Zugriffsrechte eigenmächtig umsetzt. Entweder sind dann die Zugriffsrechte in der lokalen Arbeitskopie genauso gesetzt wie im Repository oder sie werden wie bei neu angelegten Dateien gesetzt. Dies wird insbesondere dann deutlich, wenn sowohl Client als auch Server auf einer Unix-Plattform laufen, und damit die Schemata zur Vergabe von Zugriffsrechten kompatibel sind.

Meist gelten die Einstellungen aus dem Repository

Viel Einfluss auf dieses Verhalten von CVS hat man nicht. Die pragmatische Vorgehensweise an dieser Stelle ist, dass man die Zugriffsrechte direkt im Repository so setzt, wie man sie auch in der lokalen Arbeitskopie haben möchte. Aber auch das ist leider keine Garantie dafür, dass CVS die Zugriffsrechte wie gewünscht setzt. Immer zu funktionieren scheint es beim Ausführungsrecht von Dateien auf Unix. Beispielsweise könnte man die Entwicklung von Unix-Shellskripten mit CVS verwalten. Die Shellskripte sollen natürlich immer ausführbar sein. Dazu muss das entsprechende Zugriffsrecht gesetzt sein. Man wird allerdings feststellen, dass CVS ein lokal gesetztes Ausführungsrecht immer dann

überschreibt, wenn es eine Datei aus dem Repository in die lokale Arbeitskopie kopiert. Dieser Fall tritt beispielsweise ein, wenn die Datei im Repository modifiziert wurde, in der lokalen Arbeitskopie aber nicht. Man bekommt dann die Datei mit den Rechten, wie sie im Repository gesetzt sind. Zur Lösung des Problems wechselt man also direkt in das Repository und setzt dort das Ausführungsrecht der Datei. Diese Vorgehensweise ist nicht unbedingt elegant, löst aber in diesem Fall das Problem.

7.13.2 Inkonsistente lokale Arbeitskopie

Manchmal kann es vorkommen, dass durch einen Abbruch eines CVS-Befehls während der Ausführung oder durch einen Absturz des Client-Rechners, die Verwaltungsinformationen in der lokalen Arbeitskopie in einen inkonsistenten Zustand geraten. Wenn sich CVS ungewöhnlich verhält oder es zu Fehlermeldungen kommt, sollte man zunächst überprüfen, ob der Fehler auf die eigene lokale Arbeitskopie beschränkt ist. Der beste Weg dies herauszufinden ist, die gleiche Operation, die den Fehler auslöst, auf einer anderen Arbeitskopie auszuführen. Hat man keinen Zugang zu einer weiteren Arbeitskopie, so kann man sich zu diesem Zweck natürlich eine weitere lokale Arbeitskopie in ein anderes Verzeichnis auschecken. Kommt es in der anderen Arbeitskopie nicht zum Fehler, so ist mit hoher Wahrscheinlichkeit die eigene lokale Arbeitskopie der Auslöser. An dieser Stelle muss man sich nun entscheiden, wie man weiter vorgehen möchte. Man kann natürlich versuchen, die betroffene lokale Arbeitskopie zu reparieren. Ein erster Anlaufpunkt dazu sind die Verwaltungsinformationen in den CVS-Verzeichnissen in der lokalen Arbeitskopie. Man kann die dort gespeicherten Informationen analysieren und eventuelle Unstimmigkeiten beseitigen. Eine Anleitung dazu kann an dieser Stelle allerdings nicht gegeben werden. Meist ist es effizienter, sich eine neue Arbeitskopie auszuchecken und mit dieser weiter zu arbeiten. Sollten sich in der beschädigten Arbeitskopie noch wertvolle Änderungen befinden, die nicht mehr in Repository geschrieben werden konnten, so muss man diese eventuell manuell in die neue Arbeitskopie überführen. Ein einfaches Kopieren dieser Dateien in die neue Arbeitskopie kann allerdings gefährlich sein, da man auf diese Weise einen alten Versionsstand in das Repository einschleppen kann!

7.13.3 CVS und Uhren

CVS ist ein System, das stark von der richtigen Funktion der in den verwendeten Systemen eingebauten Uhren abhängig ist. Dies gilt sowohl für den Server als auch die Client-Rechner! Die Uhren sollten im Idealfall nicht voneinander abweichen. CVS bleibt durchaus benutzbar, wenn die Uhrzeiten zwischen den Rechnern einige Minuten voneinander abweichen. Die Anzahl der »Merkwürdigkeiten«, die einem CVS-Benutzer begegnen, können allerdings zunehmen, wenn einerseits die Uhrzeit und das Datum selbst nicht stimmen, und andererseits die Uhren auf Client und Server auseinander laufen. So liefern beispielsweise die Befehle **checkout** und **update** mit der Option –D aufgerufen nicht die gewünschten Ergebnisse, wenn das Datum des Servers nicht stimmt!

Zeitstempel

Die Modifikation von Dateien in der lokale Arbeitskopie bestimmt CVS anhand von Zeitstempeln, die mit der lokalen Uhr verwaltet werden. Wenn CVS schreibend auf eine Datei zugegriffen hat, so vermerkt es einen Zeitstempel in der Datei **Entries** in den lokalen Verwaltungsinformationen (siehe Abschnitt 7.4, »Die CVS-Verzeichnisse in der lokalen Arbeitskopie«). Durch Vergleich dieses Zeitstempels mit dem Zeitstempel der Datei selbst kann CVS feststellen, ob die Datei in der Zwischenzeit modifiziert worden ist. So prüft CVS diese beiden Zeitstempel nach einem Konflikt, um festzustellen, ob der Entwickler überhaupt *versucht* hat, den Konflikt zu beseitigen. Stimmen beide Zeitstempel überein, so verweigert CVS ein Einchecken mit dem Hinweis, dass der Konflikt nicht beseitigt worden ist. Man könnte CVS hier austricksen, indem man den Zeitstempel der Uhr manuell setzt, unter Unix beispielsweise mit dem Befehl **touch**. Sinnvoll ist das allerdings nicht.

Network Time Protocol

Wie schon beschrieben, sind ein paar Minuten Abweichung zwischen den Uhren nicht schlimm. Wer das Problem gründlich lösen möchte, der sollte die Uhren des CVS-Servers und aller Client-Rechner mit dem *Network Time Protocol* (NTP) synchronisieren. Dabei gleichen die Rechner ihre internen Uhren mit Zeit-Servern im Internet ab, die ihre Zeitinformationen von sehr genauen Atomuhren erhalten. Nimmt man eine solche Zeit-Synchronisation vor, dann braucht man sich über falsch gehende Uhren keine Gedanken mehr machen. Allerdings sollte man – wie auch sonst – natürlich auf die Einstellung der richtigen Zeitzone achten!

7.13.4 Locks im Repository

Eine weitere Quelle von Problemen können die *Lock-Dateien* sein. Dies sind Dateien, die CVS auf dem Server anlegt, um den gleichzeitigen Zugriff auf ein Verzeichnis zu verbieten. Manchmal kann man Meldungen folgender Art erhalten:

```
cvs update: [11:43:23] waiting for elke's lock in
   /var/lib/cvs/TextPrinter1
```

Dies ist zunächst einmal eine ganz normale Meldung und noch kein Fehler. Es bedeutet, dass der genannte CVS-Benutzer bereits eine Operation auf dem Verzeichnis ausführt und CVS nun darauf wartet, dass diese Operation beendet wird. CVS sichert die Operation durch eine Datei, die es – per Voreinstellung – in das gleiche Verzeichnis schreibt; im obigen Beispiel ist dies das Verzeichnis **/var/lib/cvs/TextPrinter1**. Ist die Lock-Datei bereits vorhanden, so lässt CVS keinen weiteren Zugriff auf das Verzeichnis zu und die oben gezeigte Meldung wird ausgegeben. Nach etwa 30 Sekunden prüft CVS erneut, ob die Lock-Datei noch vorhanden ist. Falls sie nicht mehr vorhanden ist, läuft alles normal weiter. Ist die Lock-Datei noch vorhanden, so wird erneut eine Meldung ausgegeben und CVS wartet weitere 30 Sekunden. Geht es nach mehreren Minuten immer noch nicht weiter, dann sollte man versuchen, den betreffenden Entwickler zu kontaktieren und ihn fragen, ob er noch irgendeinen CVS-Befehl ausführt. Ist das nicht der Fall, dann ist die Lock-Datei von CVS nicht gelöscht worden und muss manuell entfernt werden. Normalerweise sollte dieser Fall natürlich nicht auftreten, aber auch CVS ist nicht perfekt. Die Lock-Dateien tragen Namen wie **#cvs.rfl**, **#cvs.wfl** oder **#cvs.lock**. Man findet sie entweder in dem betroffenen Verzeichnis oder – wenn der CVS-Server entsprechend eingerichtet worden ist – in einem eigenen Lock-Verzeichnis. Ein solches Lock-Verzeichnis spiegelt den Verzeichnisbaum des Repositories. In diesem Fall sind die Lock-Dateien im korrespondierenden Unterverzeichnis zu löschen; im Repository befinden sich dann keine Lock-Dateien. Eine Anleitung zum Einrichten eines Lock-Verzeichnis ist in Abschnitt 8.6, »Ein eigenes Lock-Verzeichnis einrichten«, zu finden.

7.14 Schlüsselwortersetzung

CVS bietet die Möglichkeit eine Reihe von Schlüsselwörtern in den Quelltext einzusetzen, die vom System durch Werte ersetzt werden. Nützlich ist das, wenn man Statusinformationen zu einer Datei direkt

in dieser gespeichert haben möchte. Ein Beispiel für eine Java-Datei könnte folgendermaßen aussehen:

```
/**
*
* @author: $Author$
* @version: $Revision$
* $Date$
* $Log$
*
*/
public class MyTextPrinter extends TextPrinter
{
   //...
}
```


Schlüsselwörter

So sieht die Datei vor dem ersten Einchecken aus. CVS-Schlüsselwörter sind `$Author$`, `$Revision$`, `$Date$` und `$Log$` (Anmerkung für Java-Unkundige: `@author` und `@version` sind JavaDoc-Tags und spielen hier keine Rolle). Beim ersten Commit ersetzt CVS die Schlüsselwörter durch ihre aktuellen Werte. Im Beispiel könnte das so aussehen:

```
/**
*
* @author: $Author: elke $
* @version: $Revision: 1.1 $
* $Date: 2003/12/18 10:21:03 $
* $Log: header.java,v $
* Revision 1.1  2003/12/18 10:21:03  elke
* Datei neu hinzugefuegt
*
*
*/
public class MyTextPrinter extends TextPrinter
{
   //...
}
```

Man kann sehen, dass auch nach der Schlüsselwortersetzung die Schlüsselwörter selbst im Quelltext erhalten bleiben, so dass die Ersetzung im Folgenden immer wiederholt werden kann. So sei die Datei nochmals nach einem weiteren Commit betrachtet:

```
/**
*
* @author: $Author: frank $
* @version: $Revision: 1.2 $
* $Date: 2003/12/18 10:36:30 $
* $Log: header.java,v $
* Revision 1.2  2003/12/18 10:36:30  frank
* Elkes Version verbessert
*
* Revision 1.1  2003/12/18 10:21:03  elke
* Datei neu hinzugefuegt
*
*
*/
public class MyTextPrinter extends TextPrinter
{
   //......
}
```

An diesem Beispiel ist die Besonderheit des Schlüsselworts `$Log$` zu erkennen. `$Log$` ist das einzige CVS-Schlüsselwort, das neue Zeilen zu einer Datei hinzufügt.

Wer viel mit Verzweigungen arbeitet, der sollte sich den Einsatz des Schlüsselworts `$Log$` genau überlegen. CVS wird bei Verwendung von `$Log$` sowohl im Hauptzweig als auch in der Verzweigung mit jeder Revision neue Zeilen an genau der gleichen Stelle der Datei einfügen. Da diese Zeilen sich in Hauptzweig und Verzweigung unterscheiden werden, ist der Konflikt beim Zusammenführen der beiden Zweige vorprogrammiert!

Schlüsselwortersetzung abschalten

Nützlich ist die Schlüsselwortersetzung, wenn man immer einen Überblick über den aktuellen Stand der Sourcecode-Dateien haben möchte, ohne dafür Befehle wie **cvs status** aufrufen zu müssen. Daneben bleiben die durch Schlüsselwortersetzung in die Dateien geschriebenen Informationen auch nach einem Export bestehen. Wenn die Schlüsselwortersetzung an einer bestimmten Stelle stört, dann lässt sie sich auch – dateibezogen – abschalten. Ein Grund könnte sein, dass eines oder mehrere der CVS-Schlüsselwörter in der verwendeten Programmiersprache eine eigene Bedeutung haben und daher nicht ersetzt werden

dürfen. Bei neuen Dateien schaltet man die Schlüsselwortersetzung ab, indem man bei Aufruf des Befehls **add** die Option –ko angibt:

```
cvs add -ko header.java
```

Bei bestehenden Dateien kann man dies mit dem Befehl **admin** erreichen:

```
cvs admin -ko header.java
```

Die nachfolgende Tabelle führt alle möglichen Schlüsselwortersetzungsmethoden auf.

Liste der Schlüsselwortersetzungsmethoden

Option	Beschreibung
-kkv	Diese Option bewirkt das Standardverhalten von CVS. Sie ist nur sinnvoll, um eine andere Einstellung zurückzusetzen.
-kkvl	Bei gesperrten Dateien (siehe Abschnitt 7.17.1, »Das Sperren von Dateien«) wird immer der CVS-Benutzer eingetragen, der die Datei gesperrt hat.
-kk	Ersetzt das Schlüsselwort durch sich selbst und »verschluckt« dabei den Wert. Aus $Revision 1.17$ wird damit $Revision$. Diese Option ist sinnvoll, wenn man mit CVS Vorlagen (Templates) verwaltet, in denen das Schlüsselwort nicht durch einen Wert ersetzt werden soll.
-ko	Schaltet die Schlüsselwortersetzung komplett ab. Die Datei wird aber weiterhin als Textdatei behandelt.
-kb	Schaltet die Schlüsselwortersetzung und die automatische Zusammenführung von Dateien (Merging) ab. Die Datei wird als Binärdatei behandelt.
-kv	Die Option sorgt dafür, dass das Schlüsselwort durch seinen Wert ersetzt wird, also beispielsweise $Revision$ durch 1.2. Damit sind dann natürlich keine nachfolgenden Ersetzungen mehr möglich, da das Schlüsselwort nicht mehr vorhanden ist!

7.15 Wrapper

Wrapper sind ein Mechanismus von CVS, der Aktualisierungs- und Schlüsselwortersetzungsmethode für vorgegebene Dateimuster setzt. Die Aktualisierungsmethode legt dabei fest, ob auf den betroffenen Dateien ein Merge unternommen werden soll oder nicht. Die Schlüsselwortersetzungsmethode bestimmt, ob es sich um eine Binärdatei handelt und wie eventuell vorhandene CVS-Schlüsselwörter ersetzt werden sollen (siehe dazu den vorangehenden Abschnitt). Es gibt drei Stellen, an denen entsprechende Dateimuster abgelegt werden können: in der lokalen Datei **.cvswrappers,** in der Datei **cvswrappers**, in den

administrativen Dateien in CVSROOT (siehe Abschnitt 8.2 »Die administrativen Dateien in CVSROOT«) und in der Umgebungsvariablen `$CVSWRAPPERS`.

Der Schlüsselwortersetzungsmechanismus wird wie gewohnt mit der Option –k angegeben. Die möglichen Optionen sind die gleichen wie sie in der Tabelle im vorherigen Abschnitt aufgeführt worden sind. Der Aktualisierungsmechanismus wird mit der Option –m gesetzt. Hier gibt es nur zwei mögliche Werte: COPY und MERGE. Bei MERGE wird versucht die Datei aus den beiden Quellen automatisch zusammenzuführen, bei COPY wird dieser Mechanismus abgeschaltet und es wird die Datei aus dem Repository heraus in die lokale Arbeitskopie kopiert. Eine solche Datei könnte beispielsweise folgendermaßen aussehen:

```
*.gif -kb
*.jpeg -kb
*.text -m COPY -kkv
```

Diese Datei bestimmt, dass alle Dateien, die auf gif oder jpeg enden, als Binärdateien behandelt werden sollen und dass Dateien, die auf text enden, mit der normalen Schlüsselwortersetzung arbeiten sollen, jedoch kein Merge auf diesen Dateien versucht werden soll.

Wrapper beim Import

Wrapper sind ausgesprochen nützlich und wichtig, wenn man Projekte importieren möchte. Man muss dann nicht für jeden möglichen Dateityp auf der Kommandozeile angeben, wie er behandelt werden soll. Dies gilt allerdings nicht in WinCvs, da hier ein eigener Mechanismus beim Import von Projekten implementiert worden ist. Dieser ist in Abschnitt 6.4, »Ein neues Projekt beginnen: import«, beschrieben worden. Natürlich werden Wrapper nicht nur beim Befehl **import** verwendet, sondern auch beim Befehl **add**.

CVS-Versionen bis 1.9 überschreiben Dateien einfach, wenn die Option COPY gesetzt ist. Man sollte COPY daher nur bei neueren Versionen von CVS verwenden!

7.16 Das CVS-Protokoll und der Befehl history

7.16.1 CVS führt Protokoll

Manchmal gibt es den Wunsch, detailliert nachzuvollziehen, was zu einem bestimmten Zeitpunkt im Repository passiert ist. CVS kann zu

diesem Zweck ein Protokoll führen, das über einen eigenen Befehl – den Befehl **history** – abgefragt werden kann. Gespeichert wird dieses Protokoll in einer Datei, die ebenfalls **history** heißt und die im Verzeichnis **CVSROOT** des Repositories liegen muss. Gesteuert wird die Protokollfunktion durch die Existenz der Datei **history**: Ist sie vorhanden, dann wird protokolliert, ist sie nicht vorhanden, dann gibt es auch kein Protokoll. Um das Protokollieren zu starten, sollte man die Datei also anlegen, wenn sie noch nicht vorhanden ist. Achten muss man auf die Zugriffsrechte der Datei. Jeder sollte lesend und schreibend auf diese Datei zugreifen können, damit die Protokollierung reibungslos funktioniert. In der Datei **checkoutlist** (siehe Abschnitt 8.2.1 »Die Datei checkoutlist«) sollte **history** übrigens nicht eingetragen werden, da es sich bei der Datei nicht um eine Konfigurationsdatei handelt, und die Datei weder von CVS-Benutzern noch vom Administrator verändert werden sollte. Der Zugriff auf die Datei sollte nur über den Befehl **history** erfolgen und keinesfalls direkt, da das Dateiformat von **history** nicht dokumentiert ist und sich in zukünftigen Versionen von CVS ändern könnte.

Ein Beispiel:

```
cvs history -a -e -n TextPrinter1
```

gibt aus:

```
O 2003-09-29 10:34 +0000 frank TextPrinter1 =TextPrinter1=
   <remote>/*
O 2003-10-07 07:01 +0000 frank TextPrinter1 =TextPrinter1=
   <remote>/*
T 2003-10-07 10:06 +0000 frank TextPrinter1 [Version1:A]
T 2003-10-14 07:37 +0000 frank TextPrinter1 [Version1:A]
O 2003-10-14 08:39 +0000 frank [Version1] TextPrinter1
   =TextPrinter1= <remote>/*
T 2003-10-14 09:59 +0000 frank TextPrinter1 [branch:A]
O 2003-10-14 10:01 +0000 frank [branch] TextPrinter1
   =TextPrinter1= <remote>/*
T 2003-10-14 10:04 +0000 frank TextPrinter1 [branch:D]
O 2003-10-14 10:06 +0000 frank [Verzweigung1] TextPrinter1
   =TextPrinter1= <remote>/*
E 2003-10-16 09:06 +0000 frank [Version1] TextPrinter1
   =TextPrinter1= <remote>/*
```

```
O 2003-10-16 09:34 +0000 frank TextPrinter1 =TextPrinter1=
   <remote>/*
```

Die Option –a bedeutet, dass man die Ausgabe für alle Benutzer und nicht nur für die, von einem selbst ausgeführten Befehle sehen möchte. Die Option –e fordert die Ausgabe aller Eintragstypen an. Mit der Option –n wird die Ausgabe auf Dateien aus dem Modul TextPrinter1 beschränkt.

7.16.2 Das Ausgabeformat von history

Das Format von **history** ist leider nicht selbsterklärend. Es wurde mit dem Hintergedanken einer maschinellen Auswertung entworfen und wirkt daher eher kryptisch. Es handelt sich um ein zeilenorientiertes Format. Jede Zeile beschreibt einen ausgeführten Befehl und beginnt mit einem Buchstaben. Der Buchtstabe bezeichnet einen Befehl zusammen mit dem Zustand, der nach der Befehlsausführung eingetreten ist. Die folgende Tabelle zeigt die Bedeutung dieser Buchstaben:

Buchstabe	Bedeutung
F	Der Befehl **release** wurde ausgeführt.
O	Der Befehl **checkout** wurde ausgeführt.
E	Der Befehl **export** wurde ausgeführt.
T	Der Befehl **rtag** wurde ausgeführt.
C	Der Befehl **update** wurde ausgeführt. Dabei kam es zu einem Konflikt. Dieser muss manuell behoben werden.
G	Der Befehl **update** wurde ausgeführt. Ein Merge war notwendig und konnte ohne Konflikt durchgeführt werden.
U	Der Befehl **update** wurde ausgeführt. Die Datei wurde in die lokale Arbeitskopie kopiert.
W	Der Befehl **update** wurde ausgeführt. Die Datei in der lokalen Arbeitskopie wurde gelöscht, da sie aus dem Repository entfernt worden war.
A	Der Befehl **commit** wurde ausgeführt. Die Datei wurde neu hinzugefügt.
M	Der Befehl **commit** wurde ausgeführt. Die Datei wurde verändert.
R	Der Befehl **commit** wurde ausgeführt. Die Datei wurde entfernt.

Auf den Buchstaben folgt immer das Datum der Ausführung zusammen mit der Uhrzeit. Danach wird der CVS-Benutzer angezeigt, der den Befehl ausgeführt hat. Dann kommen die Revision und der Dateiname, die allerdings bei einigen Befehlen wie **checkout** nicht angezeigt werden. Schließlich kommen noch der Pfad im Repository und der Name der lokalen Arbeitskopie. Bei Zugriffen von einem Client-Rechner steht hier oft nur ein wenig aussagekräftiges <remote>.

Das CVS-Protokoll ist weder intuitiv lesbar noch einfach nutzbar. Wer allerdings wissen muss, was zu einem bestimmten Zeitpunkt im Repository passiert ist, besitzt mit dem Protokoll ein nützliches Hilfsmittel.

7.17 Der Befehl admin

Der Befehl **admin** unterscheidet sich von anderen CVS-Befehlen; man kann ihm keine eindeutige Funktionsbeschreibung geben. Am Besten trifft noch die Umschreibung zu, dass **admin** der Ausführung administrativer Arbeiten auf dem Repository dient.

Der Befehl **admin** beherrscht eine ganze Reihe von Funktionen, von denen viele allerdings längst veraltet sind und nicht mehr verwendet werden sollten. Es ist auch damit zu rechnen, dass einige dieser Funktionen aus zukünftigen Versionen von CVS entfernt werden. So soll an dieser Stelle auch nicht der gesamte Funktionsumfang von **admin** beschrieben werden, sondern es wird nur auf einige wichtige Optionen eingegangen.

7.17.1 Das Sperren von Dateien

Sperren wird nicht empfohlen

Mit dem Befehl **admin** lässt sich erreichen, was einige andere Versionsmanagementsysteme von Haus aus praktizieren: das Sperren von Dateien. Wenn man eine Datei mittels des Befehls **admin** sperrt, dann lässt sie sich durch andere Entwickler so lange nicht mehr verändern, bis die Sperre wieder aufgehoben wird. Man sollte sich gut überlegen, ob man diesen Mechanismus verwenden möchte, denn er widerspricht den grundlegenden Prinzipien von CVS und wird nicht empfohlen. Das Sperren von Dateien soll daher an dieser Stelle auch nicht weiter beschrieben werden. Zum Sperren und Aufheben der Sperre dienen die Optionen –l, –L, –u und –U.

7.17.2 Nachträgliches Ändern von Log Messages

Manchmal schreibt man im Laufe der Entwicklung beim Commit etwas in eine Log Message hinein, das sich im Nachhinein als wenig sinnvoll erweist. Statt nun für immer mit der verunglückten Beschreibung leben zu müssen oder gar zu versuchen, diese durch direktes Bearbeiten der Datei im Repository zu ändern, kann man den Befehl **admin** mit der Option –m verwenden, um nachträglich Log Messages zu bearbeiten. Ein Beispiel:

```
cvs admin -m 1.2:"Neue Nachricht" TextPrinter.java
```

CVS gibt aus:

```
RCS file: /var/lib/cvs/TextPrinter1/TextPrinter.java,v
done
```

Führt man anschließend den Befehl **cvs log** auf der Datei **TextPrinter.java** aus, so findet man in der Ausgabe folgende Stelle:

```
Revision : 1.2
Date : 2003/11/19 11:24:57
Author : 'frank'
State : 'Exp'
Lines : +1 0
Description :
Neue Nachricht
```

Die alte Nachricht ist also durch den neuen Text ersetzt worden. Man kann den Text für jede Revision einzeln überschreiben, sollte ihn allerdings immer in Hochkommata einbetten, falls der Text Leerzeichen enthält.

Die nachträgliche Bearbeitung von Log Messages durch den Befehl **admin** wirkt sich auf bereits vorhandene Einträge in Sourcecode-Dateien, die durch Schlüsselwortersetzung und die Variable `$Log$` entstanden sind, nicht aus. In diesem Fall schreibt CVS immer nur den jeweils neuesten Eintrag in die Sourcecode-Datei und verändert alte Einträge nicht.

7.17.3 Die Schlüsselwortersetzung ändern

Wie es schon in Abschnitt 7.14, »Schlüsselwortersetzung«, beschrieben worden ist, lässt sich mit dem Befehl **admin** auch die Schlüsselwortersetzung nachträglich umschalten. So schaltet

```
cvs admin -ko header.java
```

die Schlüsselwortersetzung für die Datei **header.java** nachträglich ab. Alle Optionen zur Schlüsselwortersetzung sind in Abschnitt 7.14 aufgelistet.

7.18 Datumsformate in CVS

Es gibt einige Befehlsoptionen in CVS, die die Angabe eines Datums und gegebenenfalls einer Uhrzeit verlangen. Als Beispiel sei an dieser Stelle die Option –D der Befehle **update** und **checkout** genannt. Mit diesen Befehlen lässt sich der Revisionsstand von Dateien aus dem Repository abrufen, wie er zu einem bestimmten Datum und einer Uhrzeit in der Vergangenheit ausgesehen hat.

CVS kennt viele Formate

Bei der Angabe von Datum und Uhrzeit lässt CVS mehrere Formate zu. Dabei werden zwei Formate bevorzugt: Daten die nach ISO 8601 formatiert sind und Daten, die den Internet-Mail Standards RFC822 und RFC1123 folgen. Nach der ISO 8601 wird ein Datum rückwärts formatiert. Die einzelnen Teile des Datums können durch Bindestriche getrennt werden, bei der Uhrzeit wird ein Doppelpunkt verwendet:

```
2003-03-14 13:39
```

Das Datum kann auch verkürzt werden, indem die Uhrzeit weggelassen wird:

```
2003-03-14
```

Das alternative Format nach RFC822 und RFC1123 wird vorwärts notiert, der Monat durch drei Buchstaben in englischer Schreibweise abgekürzt:

```
14 Mar 2003 13:39
```

Auch hier erlaubt es CVS Teile wegzulassen. Man kann die Uhrzeit weglassen:

```
14 Mar 2003
```

und auch das Jahr

```
14 Mar
```

In diesem Fall verwendet CVS für fehlende Angaben die Bestandteile des aktuellen Datums.

Neben diesen beiden auf internationalen Normen basierenden Formaten versteht CVS eine ganze Reihe weiterer Formate. Diese sind jedoch nicht dokumentiert und sollten daher auch nicht verwendet werden.

7.19 Zusammenfassung

Dieses Kapitel hat einige fortgeschrittene Themen zu CVS betrachtet und damit Hintergrundwissen zu vielen Bereichen dieses Versionsmanagementsystems geliefert. Es wurden unter anderem einige bisher nicht beschriebene Befehle vorgestellt, die internen Verwaltungsinformationen von CVS betrachtet, die Verwaltung von Webseiten mit CVS beschrieben, Vendor Branches und überwachtes Arbeiten erklärt sowie typische Problemquellen bei der Arbeit mit CVS untersucht. Nicht alle Funktionen von CVS werden von jedem Entwickler eingesetzt. Trotzdem ist es wichtig, die Möglichkeiten von CVS zu kennen. Dieses Kapitel hat trotz seiner Themenbreite einen Bereich völlig ausgelassen: die Administration von CVS. Das soll im nächsten Kapitel nachgeholt werden!

Index