

SelfLinux-0.12.2



Verwaltung eines CVS-Archivs



Autor: Karl Fogel
Formatierung: Matthias Nuessler (m.nuessler@web.de)
Lizenz: GPL

Der folgende Text enthält das Kapitel 4 der deutschen Übersetzung des Buches "Open Source Development with CVS", welche unter der GNU Public License veröffentlicht wurden.

Das SelfLinux-Team

Inhaltsverzeichnis

1 Die Rolle des Administrators

- 1.1 Beschaffen und Installieren von CVS

2 Beschaffen und Bauen von CVS unter Unix

- 2.1 Beschaffen und Installieren von CVS unter Windows
- 2.2 Beschaffen und Installieren von CVS auf einem Macintosh
- 2.3 Beschränkungen der Macintosh- und Windows-Versionen

3 Anatomie einer CVS-Distribution

- 3.1 Informationsdateien
- 3.2 Unterverzeichnisse
- 3.3 Andere Informationsquellen

4 Mit einem Archiv beginnen

- 4.1 Der Passwortauthentisierungs-Server
- 4.2 Anonymer Zugriff über den Passwortauthentisierungs-Server
- 4.3 Archivstruktur, in quälender Detailfülle erklärt
- 4.4 RCS-Format quotiert »@«-Zeichen immer
- 4.5 Was geschieht, wenn Sie eine Datei löschen
- 4.6 Das CVSROOT/-Administrationsverzeichnis

5 Finden Sie mehr heraus

1 Die Rolle des Administrators

In [Übersicht CVS](#) haben Sie genügend über CVS gelernt, um es als Projektteilnehmer effizient verwenden zu können. Wenn Sie aber ein Projektbetreuer werden wollen, müssen Sie wissen, wie man CVS installiert und Archive administriert. In diesem Kapitel ziehen wir den Vorhang auf und betrachten im Detail, wie das Archiv strukturiert ist und wie es von CVS genutzt wird. Sie werden alle wichtigen Schritte kennenlernen, die CVS während der `update`- und `commit`-Befehle durchläuft, und wie sie das Verhalten verändern können. Durch das Verständnis, wie CVS funktioniert, werden Sie auch dazu in der Lage sein, Problemen auf den Grund zu gehen und sie auf wartbare Art zu beseitigen.

Das mag sehr kompliziert klingen, doch denken Sie daran, dass sich CVS bereits als sehr langlebig erwiesen hat und für viele weitere Jahre verwendbar sein wird. Was auch immer Sie jetzt lernen, wird für eine lange Zeit nützlich sein. Zudem tendiert CVS dazu, immer unersetzbarer zu werden, je länger Sie es verwenden. Wenn Sie sich so von etwas abhängig machen (und vertrauen Sie mir, Sie werden es), ist es die Mühe wert, es gründlich kennenzulernen.

Lassen Sie uns mit diesem Gedanken im Hinterkopf gleich am Anfang beginnen: damit, CVS auf Ihr System zu bringen.

1.1 Beschaffen und Installieren von CVS

In vielen Fällen werden Sie sich CVS gar nicht erst beschaffen müssen, da es bereits auf Ihrem System vorhanden ist. Wenn Sie eine der großen Linux- oder FreeBSD-Distributionen verwenden, ist es wahrscheinlich schon in `/usr/bin` oder einer anderen Stelle installiert. Falls nicht, können Benutzer von RedHat Linux normalerweise eine RPM-Datei (RedHat Package Manager) der aktuellsten, oder nahezu aktuellsten, Version von CVS in ihrer Distribution finden. Debian-Benutzer können immer das aktuellste Debian-Paket mit diesen Befehlen installieren:

```
user@linux ~/ $ apt-get update
user@linux ~/ $ apt-get install cvs
```

Wenn CVS noch nicht auf Ihrer Maschine installiert ist, müssen Sie es wahrscheinlich aus dem Quelltext bauen. Falls Sie kein Unix-Benutzer sein sollten, finden Sie es wahrscheinlich einfacher, ein fertig gebautes, ausführbares Programm für Ihr Betriebssystem zu verwenden (mehr dazu später). Glücklicherweise ist CVS vollständig "autokonfiskiert" - d.h. es verwendet den GNU-Autokonfigurations-Mechanismus, der das Compilieren des Quelltexts erstaunlich einfach macht.

2 Beschaffen und Bauen von CVS unter Unix

Zum Zeitpunkt dieses Schreibens gibt es zwei anerkannte Server, von denen Sie CVS herunterladen können. Einer ist der FTP-Server der Free Software Foundation, <ftp://ftp.gnu.org/gnu/cvs/>, der CVS als ein offizielles GNU-Werkzeug anbietet. Der andere ist der Server von Cyclic Software. Cyclic Software ist, wenn nicht der Betreuer von CVS, dann doch der "Betreuer der Betreuer", dadurch dass sie einen Server für das Archiv und Download-Zugriff für Benutzer und Entwickler zur Verfügung stellen. Deren Releases sind von  <http://download.cyclic.com/pub/> erhältlich.

Beide Server sind gleich empfehlenswert. Im folgenden Beispiel verwende ich den Server von Cyclic Software. Wenn Sie Ihren FTP-Client (wahrscheinlich Ihren Web-Browser) darauf ansetzen, sehen Sie eine Liste von Verzeichnissen, ungefähr wie diese:

Index of /pub
<pre>cvs-1.10.5/ 18-Feb-99 21:36 - cvs-1.10.6/ 17-May-99 10:34 - cvs-1.10/ 09-Dec-98 17:26 - macintosh/ 23-Feb-99 00:53 - os2/ 09-Dec-98 17:26 - packages/ 09-Dec-98 17:26 - rcs/ 09-Dec-98 17:26 - tkcvs/ 09-Dec-98 17:26 - training/ 09-Dec-98 17:26 - unix/ 09-Dec-98 17:26 - vms/ 09-Dec-98 17:26 -</pre>

Lenken Sie Ihre Aufmerksamkeit auf die Verzeichnisse, die mit "cvs-" beginnen (Sie können die meisten anderen ignorieren). Wie Sie sehen, gibt es drei cvs-Verzeichnisse, was bedeutet, dass Sie sich bereits entscheiden müssen: Sollen Sie die als "stabil" gekennzeichnete Version oder lieber die neuere (aber nicht so gut getestete) Interimsversion verwenden? Die stabilen Versionen haben nur einen Dezimalpunkt, wie z.B. "cvs-1.10", wogegen die Interimsversionen Unterversionsnummern am Ende angefügt haben, wie z.B. "1.10.5".

Bemerkung

Der GNU-Server bietet normalerweise nur die Hauptversionen an, nicht die Interimsversionen, an, also werden Sie nicht alle Verzeichnisse finden, wenn Sie CVS von dort beziehen.

Im Allgemeinen haben sich die Interimsversionen als sehr sicher erwiesen und enthalten manchmal Korrekturen für Fehler in der Hauptversion. Die beste Vorgehensweise ist es, höchste Interims-Release zu verwenden, aber darauf vorbereitet zu sein, so oft, wie es nötig ist auf die vorhergehende Version zurückzugehen, wenn Sie auf Probleme stoßen.

Die höchste im vorigen Beispiel gelistete Version ist **cvs-1.10.6**. In diesem Verzeichnis sehen wir Folgendes:

Index of /pub/cvs-1.10.6
<pre>cvs-1.10.6.tar.gz 17-May-99 08:44 2.2M</pre>

Das ist es - der gesamte Quelltext von CVS. Laden Sie ihn einfach auf Ihre Maschine herunter, und Sie haben alles, was Sie zum Bauen von CVS benötigen. Wenn Sie bereits mit dem Standardbauprozess für GNU-Werkzeuge vertraut sind, wissen Sie an diesem Punkt, was zu tun ist, und können wahrscheinlich bis zum Abschnitt "Anatomie einer CVS-Distribution" weiter blättern. Wenn Sie allerdings nicht sicher sind, wie Sie weitermachen sollen, lesen Sie weiter ...

Die folgenden Compilieranweisungen und Beispiele nehmen an, dass Sie eine recht standardisierte Unix-Distribution haben. Es sollte mit jeder der freien Unix-Versionen (z.B. FreeBSD oder Linux) genauso wie mit den verbreiteten kommerziellen Unix-Versionen (SunOS/Solaris, AIX, HP/UX oder Ultrix) funktionieren. Geben Sie die Hoffnung nicht auf, wenn die Anleitung für Sie nicht genau wie beschrieben funktioniert. Auch wenn die Behandlung der Compilierung auf jedem einzelnen System den Rahmen dieses Buches sprengt, werde ich später in diesem Kapitel einige Verweise auf Hilfequellen geben.

Um mit dem Compilieren fortzufahren, müssen Sie auf jeden Fall zuerst die tar-Datei mit GNU **gunzip** und **tar** auspacken (wenn diese nicht auf Ihrem System installiert sind, können Sie **gunzip** von <ftp://ftp.gnu.org/gnu/gzip/> und GNUs tar-Version von <ftp://ftp.gnu.org/gnu/tar/> bekommen):

```
user@linux ~/ $ gunzip cvs-1.10.6.tar.gz
user@linux ~/ $ tar xvf cvs-1.10.6.tar
```

Sie werden viele Dateinamen auf Ihrem Bildschirm vorbeiziehen sehen.

Danach haben Sie ein neues Verzeichnis auf Ihrer Maschine - **cvs-1.10.6** - das mit dem cvs-Quelltext gefüllt ist. Wechseln Sie in dieses Verzeichnis und konfigurieren Sie CVS für Ihr System mit dem mitgelieferten **configure**-Skript:

```
user@linux ~/ $ cd cvs-1.10.6
user@linux ~/cvs-1.10.6/ $ ./configure

creating cache ./config.cache
checking for gcc... gcc
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking how to run the C preprocessor... gcc -E
(etc)
```

Wenn das **configure**-Kommando beendet ist, weiß der Quelltextbaum alles, was über Compilieren auf Ihrer Maschine wissenswert ist. Der nächste Schritt ist:

```
user@linux ~/cvs-1.10.6/ $ make
```

(Den folgenden letzten Schritt werden Sie wahrscheinlich als Superuser durchführen müssen). Sie werden eine Menge Ausgaben vorbeifliegen sehen, danach tippen Sie:

```
root@linux ~/cvs-1.10.6/ # make install
```

Es fliegen noch mehr Ausgabezeilen vorbei; wenn diese durchgelaufen sind, wird CVS auf Ihrem System installiert sein.

Standardmäßig wird das ausführbare CVS-Programm als **/usr/local/bin/cvs** installiert werden. Dies

setzt voraus, dass Sie ein vernünftiges **make**-Programm auf Ihrem System installiert haben (falls Sie kein **make** auf Ihrem System haben, können Sie das **make**-Programm des GNU-Projekts von <http://ftp.gnu.org/gnu/make/> bekommen).

Wenn Sie CVS an einer anderen Stelle als **/usr/local/bin** installieren wollen, sollten Sie den Aufruf des einleitenden Konfigurationsschrittes ändern. Beispielsweise resultiert

```
user@linux ~/cvs-1.10.6/ $ ./configure --prefix=/usr
```

in einer Installation von CVS als **/usr/bin/cvs** (es wird immer als **PREFIX/bin/cvs** installiert). Das Standardpräfix **/usr/local** ist aber für die meisten Installationen richtig.

Bemerkung

Ein Hinweis an erfahrene Benutzer: Auch wenn ältere Versionen von CVS aus mehr als einem ausführbaren Programm bestehen - da sie auch von einem installierten RCS abhängen - ist dies seit der Version 1.10 nicht mehr der Fall. Deshalb müssen Sie sich keine Gedanken über irgendwelche Bibliotheken oder ausführbaren Programme außer cvs selbst machen.

Wenn Sie nur vorhaben, CVS für den Zugriff auf entfernte Archive zu verwenden, ist die vorhergehende Anleitung ausreichend. Wenn Sie aber auch ein Archiv auf Ihrem System anbieten wollen, sind noch einige zusätzliche Schritte notwendig, die später in diesem Kapitel behandelt werden.

2.1 Beschaffen und Installieren von CVS unter Windows

Wenn Sie nicht gerade extrem darauf versessen sind, den Quelltext für Ihre ausführbaren Programme zu besitzen, müssen Sie auf Ihrer Windows-Maschine CVS nicht aus dem Quelltext compilieren. Im Gegensatz zu Unix existieren die notwendigen Compilierwerkzeuge auf Ihrem Rechner wahrscheinlich noch nicht, also würde ein Bauen aus dem Quelltext bedeuten, dass Sie sich zuerst diese Werkzeuge besorgen müssten. Da ein solches Projekt die Grenzen dieses Buches sprengt, werde ich nur Anweisungen dafür geben, wie man ein vorcompiliertes CVS-Programm bekommen kann.

Beachten Sie zuerst, dass binäre Distributionen für Windows normalerweise nur für Hauptversionen - nicht für die Interimsversionen - gemacht werden und nicht auf dem GNU-FTP-Server zur Verfügung stehen. Also müssen Sie den Server von Cyclic Software verwenden, wo Sie im Hauptversionsverzeichnis <http://download.cyclic.com/pub/cvs-1.10/> ein zusätzliches Unterverzeichnis sehen:

Index of /pub/cvs-1.10
cvs-1.10.tar.gz 14-Aug-98 09:35 2.4M windows/

worin sich eine ZIP-Datei befindet:

Index of /pub/cvs-1.10/windows
cvs-1.10-win.zip 14-Aug-98 10:10 589k

Diese ZIP-Datei enthält eine Binärdistribution von CVS. Laden Sie diese Datei herunter, und entpacken Sie sie:

```
user@linux ~/ $ unzip cvs-1.10-win.zip
Archive: cvs-1.10-win.zip
  inflating: cvs.html
  inflating: cvs.exe
  inflating: README
  inflating: FAQ
  inflating: NEWS
  inflating: patch.exe
  inflating: win32gnu.dll
```

Die README-Datei enthält detaillierte Anweisungen. Für die meisten Installationen lassen sie sich wie folgt zusammenfassen: Legen Sie alle EXE- und DLL-Dateien in ein Verzeichnis, das in Ihrem PATH liegt. Wenn Sie zusätzlich die pserver-Methode zum Zugriff auf ein entferntes Archiv verwenden möchten, müssen Sie das Folgende in Ihre **C:\AUTOEXEC.BAT**-Datei schreiben und den Rechner neu starten: **set HOME=C:** Dies teilt CVS mit, wo die **.cvspass**-Datei zu speichern ist.

CVS unter Windows kann momentan keine Archive an entfernte Maschinen zur Verfügung stellen; es kann aber ein Client sein (um zu entfernten Repositories zu verbinden) und in lokalem Modus operieren (ein Archiv auf derselben Maschine verwenden). Größtenteils geht dieses Buch davon aus, dass CVS unter Windows als Client arbeitet. Es sollte aber nicht zu schwierig sein, ein lokales Archiv unter Windows aufzusetzen, wenn Sie die an Unix orientierte Anleitung im Rest des Kapitels gelesen haben.

Wenn Sie nur auf entfernte Archive zugreifen, müssen Sie noch nicht einmal CVS selbst verwenden. Es gibt ein Werkzeug namens WinCVS, das nur die Client-Seite von CVS implementiert. Es wird von CVS getrennt vertrieben, ist aber wie CVS unter der GNU General Public License frei verfügbar. Weitere Informationen sind von  www.wincvs.org erhältlich.

2.2 Beschaffen und Installieren von CVS auf einem Macintosh

CVS ist für den Macintosh verfügbar, allerdings nicht als Bestandteil der Hauptdistribution. Zurzeit gibt es sogar drei verschiedene Macintosh CVS-Clients:

- * MacCvs -  www.wincvs.org
- *  www.cyclic.com/maccvscient/
- * MacCVSPro -  www.maccvs.org

Ich habe ehrlich gesagt keine Ahnung, welcher der beste ist. Probieren Sie sie alle aus, nicht notwendigerweise in der angegebenen Reihenfolge, und finden Sie heraus, welchen Sie mögen. MacCVSPro scheint aktiv weiterentwickelt zu werden. MacCvs ist offenbar ein Schwesterprojekt von WinCVS und teilt sich mit ihm eine Homepage (zum Zeitpunkt dieses Schreibens erklärt eine Bemerkung auf der WinCVS-Seite: "Die Entwicklung von MacCvs wird bald weitergehen", was immer das auch bedeuten mag).

2.3 Beschränkungen der Macintosh- und Windows-Versionen

Die Macintosh- und Windows-Versionen von CVS sind generell in ihrer Funktionalität beschränkt. Sie können alle als Clients agieren, das heißt, sie können einen Archivserver kontaktieren, um eine Arbeitskopie zu erhalten, übergeben, aktualisieren und so weiter. Sie können aber nicht selbst Archive zur Verfügung stellen. Wenn Sie sie richtig einrichten, kann die Windows-Portierung ein Archiv auf der lokalen Platte verwenden, aber sie kann

immer noch keine Archive für andere Maschinen zur Verfügung stellen. Im Allgemeinen gilt, dass Sie den CVS-Server auf einer Unix-Maschine laufen lassen müssen, wenn Sie ein im Netzwerk verfügbares CVS-Archiv haben

3 Anatomie einer CVS-Distribution

Die vorhergehenden Anweisungen sind dazu gedacht, Ihnen einen schnellen Überblick zu geben, es gibt aber noch viel mehr in einer CVS Quelldistribution als nur den reinen Quelltext. Hier ist eine schnelle Übersicht des Quelltextbaums, damit Sie wissen, welche Teile nützliche Ressourcen sind und welche ignoriert werden können.

3.1 Informationsdateien

Im Hauptverzeichnis des Distributionsbaumes finden Sie verschiedene Dateien, die nützliche Informationen (und Hinweise auf weitere Informationen) beinhalten. Dies sind in ungefährer Reihenfolge der Relevanz:

NEWS - Diese Datei listet die Änderungen von einer Version zur nächsten in umgekehrter chronologischer Reihenfolge auf (das heißt, die aktuellsten zuerst). Wenn Sie CVS bereits eine Zeit lang verwendet und nur auf eine neue Version aktualisiert haben, sollten Sie sich die NEWS-Datei anschauen um zu sehen, welche neue Funktionalität verfügbar ist. Auch wenn die meisten Änderungen an CVS Rückwärtskompatibilität erhalten, kommen nicht kompatible Änderungen von Zeit zu Zeit vor. Es ist besser, hier über sie zu lesen, anstatt überrascht zu sein, wenn CVS sich nicht so verhält, wie Sie es erwarten.

BUGS - Diese Datei enthält genau das, was Sie erwarten: eine Liste bekannter Fehler in CVS. Sie sind üblicherweise nicht fatal, Sie sollten sie aber kennen, wenn Sie eine neue Version installieren.

DEVEL-CVS - Diese Datei ist die "Verfassung" von CVS. Sie beschreibt den Prozess, durch den Änderungen an der Haupt-CVS-Distribution akzeptiert werden, und die Vorgehensweisen, durch die man CVS-Entwickler wird. Sie müssen dies nicht wirklich lesen, wenn Sie CVS nur verwenden wollen; es ist jedoch hoch interessant, wenn Sie verstehen wollen, wie die größtenteils unkoordinierten Bemühungen von Menschen, die über die ganze Welt verstreut sind, zu einem funktionierenden, benutzbaren Stück Software verschmelzen. Und es ist natürlich Pflichtlektüre, wenn Sie einen Patch zu CVS einbringen wollen (sei es eine Fehlerbeseitigung oder eine neue Funktion).

HACKING - Trotz ihres Namens sagt die HACKING-Datei nicht viel über den Entwurf oder die Implementierung von CVS aus. Sie ist hauptsächlich eine Anleitung für Programmierstandards und andere technische "Administrativa" für Menschen, die darüber nachdenken, einen Patch für CVS zu schreiben. Sie kann als Ergänzung zur DEVEL-CVS-Datei angesehen werden. Nachdem Sie die grundlegende Philosophie der CVS-Entwicklung verstanden haben, müssen Sie die HACKING-Datei lesen, um dies in wirkliche Programmierpraktiken umzusetzen.

FAQ - Dies ist das Dokument, das die am häufigsten gestellten Fragen zu CVS enthält. Unglücklicherweise weist seine Betreuung einige Lücken auf. David Grubbs hat sich bis 1995 darum gekümmert, dann wurde er (vermutlich) zu beschäftigt, und es dümpelte eine Zeit lang vor sich hin. Schließlich übernahm 1997 Pascal Molli die Betreuung. Molli hatte auch nicht die Zeit, es von Hand zu betreuen, doch zumindest hat er Zeit dafür gefunden, es in sein automatisiertes FAQ-O-Matic-System einzubinden, das es der Öffentlichkeit erlaubt, die FAQ dezentralisiert zu verwalten (grundsätzlich kann jeder Einträge über ein Webformular ändern oder ergänzen). Dies war wahrscheinlich insoweit eine gute Sache, als dass die FAQ zumindest wieder betreut wurde; jedoch ist ihre gesamte Organisation und Qualitätskontrolle nicht auf demselben Niveau, wie wenn sie von einer Person betreut würde.

Die Hauptversion der FAQ ist immer auf Mollis Web-Server ( www.loria.fr/~molli/cvs-index.html, unter dem Link "Documentation") verfügbar. Die FAQ, die mit CVS-Distributionen mitgeliefert wird, wird automatisch aus dieser FAQ-O-Matic-Datenbank generiert, ist also bereits ein wenig veraltet, wenn sie die Öffentlichkeit erreicht. Dennoch kann sie durchaus hilfreich sein, wenn Sie nach Tips und Beispielen suchen, wie etwas Bestimmtes getan werden muss (z.B. das Wiedervereinigen eines großen Zweigs oder das Retten einer

gelöschten Datei). Die beste Art, sie zu benutzen, ist als Referenzdokument; Sie können sie in Ihren bevorzugten Editor laden und nach den für Sie interessanten Themen suchen. Es wäre ein Fehler zu versuchen, sie als Tutorial zu verwenden - es fehlen zu viele wichtige Tatsachen über CVS, als dass sie als komplettes Handbuch dienen könnte.

3.2 Unterverzeichnisse

Die CVS-Distribution enthält eine Reihe von Unterverzeichnissen. Im Rahmen einer normalen Installation müssen Sie nicht darin herumsuchen, wenn Sie aber im Quelltext wühlen möchten, ist es gut zu wissen, wofür jedes von ihnen gedacht ist. Hier sind sie:

```
contrib/  
diff/  
doc/  
emx/  
lib/  
man/  
os2/  
src/  
tools/  
vms/  
windows-NT/  
zlib/
```

Der Großteil davon kann ignoriert werden. Die Verzeichnisse `emx/`, `os2/`, `vms/` und `windows-NT/` enthalten alle betriebssystemspezifischen Quelltext, den Sie nur brauchen würden, wenn Sie tatsächlich einen Fehler auf Quelltextebene in CVS beseitigen wollten (eine unwahrscheinliche Situation, die aber durchaus schon vorgekommen ist). Die `diff/`- und `zlib/`-Unterverzeichnisse enthalten die CVS-internen Implementationen des diff-Programms bzw. der GNU gzip-Kompressionsbibliothek (CVS benutzt letztere, um die Menge an Daten zu reduzieren, die beim Zugriff auf entfernte Archive gesendet werden müssen).

Die Unterverzeichnisse `contrib/` und `tools/` enthalten freie Software von Dritten, die dazu gedacht ist, mit CVS benutzt zu werden. In `contrib/` finden Sie eine Ansammlung von kleinen spezialisierten Shell-Skripten (lesen Sie `contrib/README`, um herauszufinden, was sie tun). Das Unterverzeichnis `tools/` enthielt früher von Dritten beigesteuerte Software, enthält nun aber nur eine README-Datei, die unter anderem besagt:

Dieses Unterverzeichnis enthielt früher Werkzeuge, die zusammen mit CVS benutzt werden können. Im Besonderen enthielt es eine Kopie von pcl-cvs Version 1.x. Pcl-cvs ist eine Emacs-Schnittstelle für CVS. Wenn Sie pcl-cvs suchen, empfehlen wir pcl-cvs Version 2.x, zu finden unter: <http://ftp.weird.com/pub/local/>

Das hier erwähnte PCL-CVS-Paket ist sehr praktisch, und ich werde in Kapitel 10 mehr darüber schreiben.

Die Unterverzeichnisse `src/` und `lib/` enthalten den Großteil des CVS-Quelltexts, der die CVS-Interna betrifft. Die Hauptdatenstrukturen und Befehle sind in `src/` implementiert, `lib/` enthält kleine Quelltextmodule, die allgemein für CVS nützlich sind.

Das Unterverzeichnis `man/` enthält die CVS-Man-Pages (elektronische Handbucheinträge, gedacht für das Unix Online-Handbuchsystem). Wenn Sie `make install` aufrufen, werden sie den Standard Man-Pages Ihres Unix-Systems hinzugefügt, so dass Sie

```
user@linux ~/ $ man cvs
```

eingeben können und eine recht knappe Einleitung und Unterbefehlsreferenz von CVS erhalten. Obwohl sie auch als schnelles Nachschlagewerk nützlich sind, kann es sein, dass die Man-Pages nicht so aktuell oder komplett wie das Cederqvist-Handbuch sind (siehe den nächsten Abschnitt); jedoch sind die Man-Pages eher unvollständig als tatsächlich falsch, falls Sie das irgendwie beruhigt.

Das Cederqvist-Handbuch

Übrig bleibt das `doc/`-Unterverzeichnis, dessen wichtigster Teil der berühmte Cederqvist ist. Heute ist es wahrscheinlich ein wenig weit hergeholt, es "den Cederqvist" zu nennen. Auch wenn *Per Cederqvist* (von Signum Support, Linköping, Schweden) die erste Version um 1992 geschrieben hat, wurde es seitdem von vielen Menschen aktualisiert. Wenn Mitarbeiter zum Beispiel eine neue Funktionalität zu CVS hinzufügen, dokumentieren sie sie normalerweise auch im Cederqvist.

Das Cederqvist-Handbuch ist im Texinfo-Format verfasst, das vom GNU-Projekt verwendet wird, da es recht einfach ist, daraus sowohl Online- wie auch gedruckte Versionen zu erzeugen (im Info- bzw. PostScript-Format). Die Texinfo-Hauptdatei ist `doc/cvs.texinfo`, CVS-Distributionen kommen aber mit vorgenerierten Info- und PostScript-Dateien - Sie brauchen also nicht zu befürchten, die Texinfo-Werkzeuge selbst benutzen zu müssen.

Auch wenn der Cederqvist als Einführung und Tutorial verwendet werden kann, ist er wahrscheinlich als Referenzdokument am nützlichsten. Aus diesem Grund lesen ihn die meisten Menschen online, anstatt ihn auszudrucken (auch wenn die PostScript-Datei `doc/cvs.ps` ist, für Menschen mit zu viel Papier). Wenn Sie CVS zum ersten Mal auf Ihrem System installiert haben, dann müssen Sie einen zusätzlichen Schritt unternehmen, um sicherzustellen, dass das Handbuch öffentlich verfügbar ist.

Die Info-Dateien (`doc/cvs.info`, `doc/cvs.info-1`, `doc/cvs.info.2` usw.) wurden für Sie installiert, als Sie `make install` aufriefen. Auch wenn die Dateien in den Info-Verzeichnisbaum Ihres Systems kopiert wurden, kann es sein, dass Sie noch eine Zeile hinzufügen müssen, um CVS in das Info-Inhaltsverzeichnis, den "obersten Knoten", aufzunehmen. (Dies ist nur notwendig, wenn CVS zum ersten Mal auf Ihrem System installiert wurde; sonst sollte der Eintrag bereits von einer früheren Installation her im Inhaltsverzeichnis enthalten sein).

Wenn Sie schon einmal neue Info-Dokumentation hinzugefügt haben, sind Sie vielleicht mit dem Vorgang vertraut. Finden Sie zuerst heraus, wo die Info-Seiten installiert wurden. Wenn Sie die Standardinstallation verwendet haben (in `/usr/local/`), finden sich die Info-Dateien in `/usr/local/info/cvs.info*`. Wenn Sie mit

```
user@linux ~/ $ ./configure --prefix=/usr
```

installiert haben, finden sich die Dateien in `/usr/info/cvs.*`. Wenn Sie die Dateien gefunden haben, müssen Sie dem Info-Inhaltsverzeichnis, das eine Datei namens `dir` in diesem Verzeichnis ist, eine Zeile für CVS hinzufügen (im letzteren Fall wäre dies also `/usr/info/dir`). Wenn Sie keinen root-Zugriff haben, dann bitten Sie Ihren Systemverwalter darum. Hier ist ein Ausschnitt von `dir` vor dem Hinzufügen der CVS-Dokumentation:

dir
<pre>* Bison: (bison). The Bison parser generator. * Cpp: (cpp). The GNU C preprocessor. * Flex: (flex). A fast scanner generator</pre>

Und hier ist der gleiche Bereich von **dir** danach:

dir
* Bison: (bison). The Bison parser generator. * Cpp: (cpp). The GNU C preprocessor. * Cvs: (cvs). Concurrent Versions System * Flex: (flex). A fast scanner generator

Das Format der Zeile ist sehr wichtig. Sie müssen den Stern, Leerzeichen und den Doppelpunkt in `"* Cvs:"` verwenden und die Klammern und den Punkt in `"(cvs)."` danach. Wenn eines dieser Elemente fehlt, wird das Info-Verzeichnisformat gestört, und Sie können den Cederqvist nicht lesen.

Sobald das Handbuch installiert und vom Inhaltsverzeichnis aus referenziert wurde, können Sie es mit jedem Info-kompatiblen Browser lesen. Am wahrscheinlichsten existieren auf einem typischen Unix-System zum einen der Kommandozeilen Info-Browser, der folgendermaßen aufgerufen werden kann, wenn Sie direkt zu den CVS-Seiten gelangen wollen:

```
user@linux ~/ $ info cvs
```

und zum anderen der in Emacs enthaltene Browser, der durch **M-X info** oder **C-h i** aufgerufen werden kann.

Nehmen Sie sich so viel Zeit wie nötig, um den Cederqvist auf Ihrem System richtig einzurichten, wenn Sie CVS installieren; es wird sich später viele Male lohnen, wenn Sie irgendetwas nachschlagen müssen.

3.3 Andere Informationsquellen

Zusätzlich zum Cederqvist, der FAQ und den anderen Dateien in der Distribution selbst gibt es CVS-spezifische Internet-Ressourcen. Wenn Sie einen CVS-Server administrieren wollen, sollten Sie wahrscheinlich die info-cvs-Mailingliste abonnieren. Dazu senden Sie eine E-Mail an info-cvs-request@gnu.org (die Liste selbst ist info-cvs@gnu.org). Das Nachrichtenaufkommen ist mit ca. zehn bis zwanzig E-Mails pro Tag mittel bis groß, die meisten davon sind von Menschen, die bestimmte Fragen haben. Der Großteil davon kann ungelesen gelöscht werden (außer wenn Sie ihnen helfen wollen, was immer nett ist), doch ab und zu kündigt jemand die Entdeckung eines Fehlers oder einen Patch an, der eine neue Funktionalität implementiert, die Sie sich gewünscht haben.

Sie können sich auch an der formalen Fehlerberichts-Mailingliste, die jeden eingesandten Fehlerbericht enthält, beteiligen. Dies ist wahrscheinlich nicht notwendig, außer Sie haben vor, bei der Behebung der Fehler mitzuhelfen, was großartig wäre - oder Sie sind fürchterlich paranoid und möchten über jedes Problem, das andere in CVS finden, Bescheid wissen. Wenn Sie daran teilnehmen wollen, senden Sie eine E-Mail an bug-cvs-request@gnu.org.

Es gibt auch eine Usenet-Newsgruppe, **comp.software.config-mgmt**, die Versionskontrolle und Konfigurationsmanagement allgemein behandelt und in der relativ viel über CVS diskutiert wird.

Zuletzt gibt es mindestens drei Webseiten, die sich mit CVS beschäftigen. Der Webserver von Cyclic Software,  www.cyclic.com ist seit einigen Jahren die informelle Homepage von CVS und wird es wahrscheinlich auch auf absehbare Zeit bleiben. Cyclic Software stellt auch Serverplatz und Netzzugriff für das Archiv bereit, in dem

sich die CVS-Quellen befinden. Die Webseiten von Cyclic enthalten umfassende Links zu experimentellen CVS-Patches, Werkzeuge von Dritten, die mit CVS zusammenarbeiten, Dokumentation, Mailinglisten-Archive und so ziemlich alles andere. Wenn Sie etwas, das Sie benötigen, nicht in der Distribution finden, ist www.cyclic.com der richtige Ort, um mit der Suche zu beginnen.

Zwei andere gute Server sind Pascal Mollis www.loria.fr/~molli/cvs-index.html und Sean Dreilingers <http://durak.org/cvswebsites>. Die größte Attraktion an Mollis Webseiten ist natürlich die FAQ, doch sie beinhalten auch Links zu CVS-bezogenen Werkzeugen und Mailinglisten-Archiven. Dreilingers Seiten sind auf Information zur Verwendung von CVS zur Verwaltung von Webdokumenten spezialisiert und haben auch eine CVS-spezifische Suchmaschine.

4 Mit einem Archiv beginnen

Sobald die CVS-Programme auf Ihrem System installiert sind, können Sie sie direkt als Client benutzen, um auf entfernte Archive zuzugreifen, indem Sie die in Kapitel 2 beschriebene Vorgehensweise anwenden. Wenn Sie jedoch Revisionen auf Ihrer Maschine zur Verfügung stellen wollen, müssen Sie dort ein Archiv erzeugen.

Der Befehl dafür ist:

```
user@linux ~/ $ cvs -d /usr/local/newrepos init
```

wobei `/usr/local/newrepos` ein Pfad zu der (beliebigen) Stelle ist, an der Sie das Archiv haben möchten. (Natürlich müssen Sie dafür Schreibberechtigung haben, was implizieren kann, dass Sie das Programm als Benutzer root ausführen müssen.) Es mag ein wenig gegen die Intuition sein, dass der Ort des neuen Archivs vor dem `init` Unterkommando anstatt danach angegeben wird, doch durch die Verwendung der `-d`-Option wird die Konsistenz mit den anderen CVS-Kommandos gewährleistet.

Der Befehl gibt nach seiner Ausführung keine Meldung zurück. Lassen Sie uns das neue Archiv untersuchen:

```
user@linux ~/ $ ls -ld /usr/local/newrepos
drwxrwxr-x 3 root root 1024 Jun 19 17:59 /usr/local/newrepos/

user@linux ~/ $ cd /usr/local/newrepos
user@linux /usrlocal/newrepos/ $ ls

CVSR00T

user@linux /usrlocal/newrepos/ $ cd CVSR00T
user@linux /usrlocal/newrepos/CVSR00T/ $ ls

checkoutlist config,v history notify taginfo,v
checkoutlist,v cvswrappers logininfo notify,v verifymsg
commitinfo cvswrappers,v logininfo,v rcsinfo verifymsg,v
commitinfo,v editinfo modules rcsinfo,v
config editinfo,v modules,v taginfo

user@linux /usrlocal/newrepos/CVSR00T/ $
```

Das einzige Unterverzeichnis im neuen Archiv - `CVSR00T/` - enthält verschiedene administrative Dateien, die das Verhalten von CVS kontrollieren. Später werden wir diese Dateien einzeln untersuchen; momentan haben wir einfach das Ziel, das Archiv ans Laufen zu bekommen. Hierbei bedeutet "laufen", dass Benutzer Projekte importieren, auschecken, aktualisieren und mit `commit` in das Archiv übertragen können.

Bemerkung

Verwechseln Sie die Umgebungsvariable `CVSR00T`, die in Kapitel 2 eingeführt wurde, nicht mit diesem Unterverzeichnis `CVSR00T` im Archiv. Sie haben nichts miteinander zu tun - es ist ein unglücklicher Zufall, dass beide den gleichen Namen tragen. Ersteres ist eine Methode, damit Benutzer nicht immer `-d <Archiv-Ort>` bei der Benutzung von CVS tippen müssen; Letzteres ist das administrative Unterverzeichnis eines Archivs.

Sobald das Archiv erzeugt wurde, müssen Sie sich um seine Zugriffsrechte kümmern. CVS schreibt kein bestimmtes, standardisiertes Zugriffsrechts- oder Dateibesitzerschema vor; es braucht nur Schreibzugriff auf das Archiv. Ich empfehle jedoch sehr - zum Teil aus Sicherheitsgründen, doch hauptsächlich zu Ihrem eigenen Wohl

als Administrator - die folgenden Schritte durchzuführen:

Erzeugen Sie eine Unix-Gruppe "cvs". Alle Benutzer mit Zugriff auf das Archiv sollten in dieser Gruppe sein. Hier ist z.B. die relevante Zeile der `/etc/group`-Datei meiner Maschine:

/etc/group
cvs:*:105:kfogel,sussmann,jimb,noel,lefty,fitz,craig,anonymous,jrandom

Passen Sie die Gruppenzugehörigkeit und Zugriffsrechte auf diese neue Gruppe an:

```
user@linux ~/ $ cd /usr/local/newrepos
user@linux /usr/local/newrepos/ $ chgrp -R cvs .
user@linux /usr/local/newrepos/ $ chmod ug+rwX . CVSROOT
```

Jetzt können alle Benutzer, die in der Gruppe gelistet sind, ein Projekt beginnen, indem sie `cvs import` wie in Kapitel 2 beschrieben aufrufen. `checkout`, `update` und `commit` sollten genauso funktionieren. Sie können das Archiv auch von entfernten Orten über die `:ext:`-Methode erreichen, vorausgesetzt, sie haben `rsh`- oder `ssh`-Zugriff auf die Archivmaschine. (Sie haben vielleicht bemerkt, dass die `chgrp`- und `chmod`-Kommandos in diesem Beispiel einem Benutzer "anonymous" Schreibzugriff gegeben haben, was Sie wahrscheinlich nicht erwartet haben. Der Grund dafür ist, dass auch anonyme Benutzer, die nur lesend auf das Archiv zugreifen, Schreibzugriff auf Systemebene benötigen, damit ihre CVS-Prozesse temporäre Lock-Dateien innerhalb des Archivs anlegen können. CVS erzwingt die "Nur-lesen" Restriktion des anonymen Zugriffs nicht durch Unix-Dateizugriffsrechte, sondern mit anderen Mitteln, die wir in Kürze behandeln werden.)

Wenn Ihr Archiv Projekte für die Öffentlichkeit zur Verfügung stellen soll, wobei Mitarbeiter nicht notwendigerweise Benutzerkennungen auf der Archivmaschine haben, sollten Sie nun den Passwortauthentisierungs-Server einrichten. Er ist für anonymen Nur-lese-Zugriff notwendig, und es ist wahrscheinlich der einfachste Weg, bestimmten Menschen `commit`-Zugriff zu gewähren, ohne ihnen vollständige Benutzerkennungen auf der Maschine zu geben.

4.1 Der Passwortauthentisierungs-Server

Bevor wir die Schritte durchgehen, die zum Einrichten des Passwortservers notwendig sind, betrachten wir abstrakt, wie solche Verbindungen funktionieren. Wenn ein entfernter CVS-Client die `:pserver:`-Methode verwendet, um sich mit einem Archiv zu verbinden, kontaktiert der Client in Wirklichkeit eine bestimmte Port-Nummer auf der Servermaschine - um genau zu sein, Port-Nummer 2401 (was 49 zum Quadrat ist, wenn Sie so etwas mögen). Port 2401 ist der ausgewiesene Standard-Port für den CVS-pserver; man kann auch einen anderen Port verwenden, solange Client und Server sich über die Port-Nummer einig sind.

Der CVS-Server wartet nicht wirklich auf Verbindungen zu diesem Port - der Server wird nicht gestartet, bis tatsächlich eine Verbindung ankommt. Anstelle dessen horcht der Unix-inetd (InterNET Daemon) an diesem Port und muss wissen, dass er den CVS-Server starten und mit den ankommenden Client verbinden soll, sobald er eine Verbindungsanfrage erhält.

Dies wird durch die Änderung der inetd-Konfigurationsdateien `/etc/services` und `/etc/inetd.conf` erreicht. Die `services`-Datei setzt rohe Port-Nummern in Dienstnamen um, und `inetd.conf` teilt inetd dann mit, was für einen bestimmten Dienstnamen zu tun ist.

Fügen Sie zunächst die folgende Zeile in `/etc/services` ein (nachdem Sie überprüft haben, ob sie nicht bereits existiert):

/etc/services
<code>cvspserver 2401/tcp</code>

Dann fügen Sie Folgendes in `/etc/inetd.conf` ein:

/etc/inetd.conf
<code>cvspserver stream tcp nowait root /usr/local/bin/cvs/cvs \ --allow-root=/usr/local/newrepos pserver</code>

(In der Datei selbst sollte dies alles eine lange Zeile ohne den Backslash sein). Wenn Ihr System tcp wrapper verwendet, möchten Sie etwa so etwas verwenden:

/etc/inetd.conf
<code>cvspserver stream tcp nowait root /usr/sbin/tcpd /usr/local/bin/cvs/cvs \ --allow-root=/usr/local/newrepos pserver</code>

Starten Sie jetzt `inetd` neu, damit er die Änderungen in den Konfigurationsdateien registriert. (Wenn Sie nicht wissen, wie man den Daemon neu startet, dann booten Sie den Rechner einfach neu - das wird auch funktionieren.)

Das reicht aus, um Verbindungen zu erlauben. Sie werden aber auch spezielle CVS-Passwörter - getrennt von den normalen Benutzerpasswörtern - einrichten wollen, damit jemand auf das Archiv zugreifen kann, ohne die allgemeine Systemsicherheit zu beeinträchtigen.

Die CVS Passwortdatei befindet sich im Archiv in `CVSROOT/passwd`. Sie wurde nicht standardmäßig beim Aufruf von `cvs init` erzeugt, da CVS nicht wissen konnte, dass Sie `pserver` benutzen werden. Auch wenn die Passwortdatei erzeugt wurde, kann CVS nicht wissen, welche Benutzernamen und Passwörter angelegt werden sollen. Also müssen Sie sie selbst anlegen. Hier ist eine Beispiel-`CVSROOT/passwd`-Datei:

CVSROOT/passwd
<code>kfogel:rKa5jzULzmh0o anonymous:XR4EZcEs0szik melissa:tGX1fS8sun6rY:pubcvs</code>

Das Format ist so einfach, wie es aussieht. Jede Zeile sieht wie folgt aus:

CVSROOT/passwd
<BENUTZERNAME>:<VERSCHLÜSSELTES PASSWORT>:<OPTIONALER SYSTEM BENUTZERNAME>

Der zusätzliche Doppelpunkt, gefolgt von einem optionalen Systembenutzernamen, sagt CVS, dass Verbindungen, die mit BENUTZERNAME authentisiert sind, unter der Systembenutzerkennung SYSTEM-BENUTZERNAME laufen sollen - anders gesagt, wäre diese CVS-Sitzung nur dazu in der Lage, Dinge im Archiv zu tun, die jemand tun könnte, der als SYSTEM BENUTZERNAME angemeldet wäre.

Wenn kein Systembenutzername angegeben ist, muss BENUTZERNAME einem tatsächlich existierenden Benutzernamen auf dem System entsprechen, und die Sitzung wird mit den Zugriffsrechten dieses Benutzers gestartet. In beiden Fällen sollte das verschlüsselte Passwort nicht gleich dem tatsächlichen Login-Passwort des Benutzers sein. Es sollte ein unabhängiges Passwort sein, das nur für CVS pserver-Verbindungen verwendet wird.

Das Passwort wird mit denselben Algorithmen wie die Standard-Unix-Systempasswörter in `/etc/passwd` verschlüsselt. Sie fragen sich vielleicht an dieser Stelle, wie man an eine verschlüsselte Version eines Passwortes kommt? Für Unix-Systempasswörter kümmert sich `passwd` um die Verschlüsselung in `/etc/passwd`. Unglücklicherweise gibt es kein entsprechendes `cv`s `passwd`-Kommando. (Es wurde schon mehrere Male vorgeschlagen, doch es ist noch niemand dazu gekommen, es zu schreiben - vielleicht werden Sie es tun !)

Dies ist eine Unannehmlichkeit, aber nur eine kleine. Wenn alles andere versagt, können Sie immer noch temporär ein reguläres Systempasswort eines Benutzers mit `passwd` ändern, den verschlüsselten Text von `/etc/passwd` nach `CVSROOT/passwd` kopieren und das alte Passwort wiederherstellen.

Bemerkung

Auf einigen Systemen finden sich die verschlüsselten Passwörter in `/etc/shadow` und sind nur von root lesbar.

Dieses Schema funktioniert, ist aber recht lästig. Es wäre viel einfacher, ein Kommandozeilenprogramm zu haben, das ein Klartext-Passwort als Argument erwartet und die verschlüsselte Version ausgibt. Hier ist solch ein Werkzeug, in Perl geschrieben:

/usr/local/bin/cryptout.pl
<pre>#!/usr/bin/perl srand (time()); my \$randletter = "(int (rand (26)) + (int (rand (1) + .5) % 2 ? 65 : 97))"; my \$salt = sprintf ("%c%c", eval \$randletter, eval \$randletter); my \$plaintext = shift; my \$crypttext = crypt (\$plaintext, \$salt); print "{\$crypttext}\n";</pre>

Ich habe dieses Skript als `/usr/local/bin/cryptout.pl` gespeichert:

```
user@linux ~/ $ ls -l /usr/local/bin/cryptout.pl
-rwxr-xr-x 1 root root 265 Jun 14 20:41 /usr/local/bin/cryptout.pl
user@linux ~/ $ cryptout.pl "ein Text"
sB3A79YDX5L4s
user@linux ~/ $
```

Wenn ich die Ausgabe dieses Beispiels nähme und es zum Erzeugen des folgenden Eintrags in **CVSROOT/passwd** verwendete

CVSROOT/passwd
jrandom:sB3A79YDX5L4s:craig

dann könnte sich jemand mit dem folgenden Kommando mit dem Archiv verbinden:

```
user@linux ~/ $ remote$ cvs -d
:pserver:jrandom@floss.red-bean.com:/usr/local/newrepos login
```

Sie können dann "ein Text" als ihr Passwort tippen und sind danach in der Lage, CVS-Kommandos mit den selben Rechten wie der Systembenutzer "craig" auszuführen.

Wenn jemand versucht, sich mit einem Benutzernamen und Passwort zu authentisieren, die nicht in **CVSROOT/passwd** enthalten sind, prüft CVS, ob der Benutzername und das Passwort in **/etc/passwd** vorhanden sind. Wenn sie es sind (und das Passwort natürlich übereinstimmt), erlaubt CVS den Zugriff. Es verhält sich so, um die Arbeit des Administrators zu erleichtern, damit separate **CVSROOT/passwd**-Einträge nicht für normale Systembenutzer eingerichtet werden müssen. Dieses Verhalten ist aber auch eine Sicherheitslücke, da es bedeutet, dass das Passwort dieser Benutzer im Klartext über das Netzwerk übertragen wird, sobald sie sich mit dem CVS-Server verbinden, was es für die Augen von Passwort-Sniffern potenziell verwundbar macht. Ein wenig später werden Sie sehen, wie man dieses "Rückfallverhalten" abstellt, so dass CVS nur seine eigene **passwd**-Datei verwendet. Egal ob Sie es ausschalten oder aktiv lassen, sollten Sie doch alle CVS-Benutzer, die auch Login-Benutzerkennungen haben, dazu zwingen, verschiedene Passwörter für die beiden Funktionen zu verwenden.

Auch wenn die **passwd**-Datei für den Zugriff auf das gesamte Archiv authentisiert, können Sie sie immer noch mit nur wenig zusätzlichem Aufwand für die Vergabe von projekt-spezifischen Zugriffsrechten verwenden. Im Folgenden beschreibe ich eine mögliche Methode:

Angenommen Sie möchten einigen entfernten Benutzern Zugriff auf das Projekt "foo" und anderen auf das Projekt "bar" gestatten und möchten nicht, dass Entwickler eines Projektes commit-Zugriff auf das andere haben. Sie können dies dadurch erreichen, dass Sie projektspezifische Benutzerkennungen und Gruppen im System erzeugen und dann diese Benutzerkennungen auf die **CVSROOT/passwd**-Datei abbilden:

Hier ist der relevante Auszug auf **/etc/passwd**:

/etc/passwd
cvcs-foo:*:600:600:Public CVS Account for Project Foo:/usr/local/cvs:/bin/false cvcs-bar:*:601:601:Public CVS Account for Project Bar:/usr/local/cvs:/bin/false

und aus **/etc/group**:

cvcs-foo:*:600:cvcs-foo cvcs-bar:*:601:cvcs-bar
--

und schließlich **CVSROOT/passwd**:

CVSROOT/passwd
kcunderh:rKa5jzULzmh0o:cvcs-foo jmankoff:tGX1fS8sun6rY:cvcs-foo brebard:cAXVPNZN6uFH2:cvcs-foo xwang:qp5lsf7nzRzfs:cvcs-foo dstone:JDNNF6HeX/yLw:cvcs-bar twp:glUHEM8Khcb06:cvcs-bar ffranklin:cG6/6yXbS9BHI:cvcs-bar yyang:YoEqcCeCUq1vQ:cvcs-bar

Einige der CVS-Benutzernamen werden auf die Systembenutzerkennung cvs-foo und einige auf cvs-bar abgebildet. Da CVS unter der Benutzerkennung des Systemverwalters läuft, müssen Sie nur sicherstellen, dass die relevanten Teile des Archivs nur von den zugehörigen Benutzern und Gruppen beschrieben werden können. Wenn Sie nur überprüfen, dass die Benutzerkennungen sicher gemacht wurden (kein gültiges login-Passwort, **/bin/false** als Shell), ist dieses System ausreichend sicher (aber lesen Sie weiter hinten in diesem Kapitel den Abschnitt über CVS-Zugriffsrechte!). Zudem protokolliert CVS Änderungen und Log-Mitteilungen unter dem CVS-Benutzernamen und nicht dem Systembenutzernamen, sodass Sie immer noch feststellen können, wer für eine bestimmte Änderung verantwortlich ist.

4.2 Anonymer Zugriff über den Passwortauthentisierungs-Server

Bisher haben wir nur gesehen, wie man den Passwortauthentisierungs-Server dazu verwendet, vollen Zugriff auf das Archiv zu gewähren (auch wenn man zugegebenermaßen diesen Zugriff durch sorgfältig gewählte Unix-Dateizugriffsrechte beschränken kann). Anonymen Nur-Lesezugriff zu gewähren ist ein einfacher Schritt: Sie müssen nur eine, vielleicht zwei Dateien in **CVSROOT/** anlegen. Die Namen dieser Dateien sind "readers" und "writers"⁴ - die erste Datei enthält eine Liste von Benutzernamen, die das Archiv nur lesen dürfen, die zweite eine Liste derer, die lesen und schreiben dürfen.

Wenn Sie einen Benutzernamen in **CVSROOT/readers** auflisten, hat dieser Benutzer nur Lesezugriff auf alle Projekte im Archiv. Wenn Sie einen Benutzernamen in **CVSROOT/writers** auflisten, hat dieser Benutzer Schreibzugriff, und jeder pserver-Benutzer, der nicht in writers aufgeführt ist, hat nur lesenden Zugriff (d.h.

wenn die writers-Datei überhaupt existiert, impliziert sie nur lesenden Zugriff für alle nicht in ihr enthaltenen Benutzer). Wenn der gleiche Benutzername in beiden Dateien aufgeführt ist, löst CVS den Konflikt auf konservative Weise: Der Benutzer erhält nur lesenden Zugriff.

Das Format dieser Dateien ist sehr einfach: ein Benutzer pro Zeile (vergessen Sie nicht, einen Zeilenvorschub nach dem letzten Benutzer zu tippen). Hier ist ein Beispiel für eine readers-Datei:

CVSROOT/readers
anonymous splotnik guest jbrowse

Denken Sie daran, dass sich die Dateien nur auf CVS-Benutzernamen, nicht auf Systembenutzernamen beziehen. Wenn Sie Benutzer-Aliase in der Datei **CVSROOT/passwd** verwenden (indem Sie einen Systembenutzernamen nach dem zweiten Doppelpunkt schreiben), ist der am weitesten links stehende Benutzername derjenige, der in der readers- oder writers-Datei anzugeben ist.

Nur um furchterlich genau zu sein, hier eine formale Beschreibung des Verhaltens des Servers bei der Entscheidung, ob Nur-Lese- oder Schreib-Lese-Zugriff zu gewähren ist: Wenn eine readers-Datei existiert und dieser Benutzer darin aufgeführt ist, erhält er Nur-Lesezugriff. Wenn eine writers-Datei existiert und der Benutzer in ihr nicht aufgeführt ist, erhält er ebenfalls Nur-Lesezugriff. (Dies trifft auch zu, wenn eine readers-Datei existiert und der Benutzer dort auch nicht aufgeführt ist.) Wenn die Person in beiden Dateien aufgeführt ist, erhält sie Nur-Lesezugriff. In allen anderen Fällen erhält die Person vollen Schreib-Lese-Zugriff.

Daher hat ein typisches Archiv mit anonymem CVS-Zugriff die folgende (oder ähnliche) **CVS/passwd**-Datei:

CVS/passwd
anonymous:XR4EZcEs0szik

dies (oder Ähnliches) in **etc/passwd**:

etc/passwd
anonymous::1729:105:Anonymous CVS User:/usr/local/newrepos:/bin/false

und Folgendes in **CVSROOT/readers**:

CVSROOT/readers
anonymous

und natürlich die bereits erwähnte Konfiguration in `/etc/services` und `/etc/inetd.conf`. Das ist schon alles!

Denken Sie daran, dass einige ältere Unix-Systeme keine Benutzernamen länger als acht Zeichen erlauben. Eine Art dies zu umgehen wäre, den Benutzer "anon" statt "anonymous" in `CVSROOT/passwd` und in den Systemdateien zu nennen, da oft angenommen wird, dass anon sowieso eine Abkürzung für anonymous ist. Es ist aber besser, etwa das Folgende in `CVSROOT/passwd` zu schreiben:

CVSROOT/passwd
anonymous:XR4EZcEs0szik:cvsanon

(und dann natürlich "cvsanon" in den Systemdateien zu verwenden). Auf diese Weise sind Sie dazu in der Lage, eine Archivadresse zu veröffentlichen, die "anonymous" verwendet, was mittlerweile mehr oder weniger Standard ist. Wenn man auf das Archiv dann wie folgt zugreift:

```
user@linux ~/ $ cvs -d
:pserver:anonymous@cvs.foobar.com:/usr/local/newrepos (usw...)
```

würde der Server tatsächlich als cvsanon laufen (oder wie auch immer Sie es konfiguriert haben). Man braucht dann weder zu wissen noch sich darum zu sorgen, wie Dinge auf der Serverseite eingerichtet sind - man sieht nur die veröffentlichte Adresse.

4.3 Archivstruktur, in quälender Detailfülle erklärt

Das neue Archiv beinhaltet immer noch keine Projekte. Lassen Sie uns den ursprünglichen Import aus Kapitel 2 noch einmal betrachten und beobachten, was im Archiv dabei vor sich geht. (Aus Gründen der Einfachheit nehmen alle Befehle an, dass die CVSROOT-Umgebungsvariable auf `/usr/local/newrepos` gesetzt ist, damit bei Importen und `checkout`-Befehlen das Archiv nicht mit der Option `-d` angegeben werden muss.)

```
user@linux ~/ $ ls /usr/local/newrepos
CVSROOT/
user@linux ~/ $ pwd
/home/jrandom/src/
user@linux ~/ $ ls
myproj/
```

```
user@linux ~/ $ cd myproj
user@linux ~/myproj/ $ cvs import -m "initial import into CVS" myproj
jrandom start

N myproj/README.txt
N myproj/hello.c
cvs import: Importing /usr/local/newrepos/myproj/a-subdir
N myproj/a-subdir/whatever.c
cvs import: Importing /usr/local/newrepos/myproj/a-subdir/subsubdir
N myproj/a-subdir/subsubdir/fish.c
cvs import: Importing /usr/local/newrepos/myproj/b-subdir
N myproj/b-subdir/random.c
No conflicts created by this import

user@linux ~/myproj/ $ ls /usr/local/newrepos

CVSROOT/ myproj/

user@linux ~/myproj/ $ cd /usr/local/newrepos/myproj
user@linux /usr/local/newrepos/myproj/ $ ls

README.txt,v a-subdir/ b-subdir/ hello.c,v

user@linux /usr/local/newrepos/myproj/ $ cd a-subdir
user@linux /usr/local/newrepos/myproj/a-subdir/ $ ls

subsubdir/ whatever.c,v

user@linux /usr/local/newrepos/myproj/a-subdir/ $ cd ..
user@linux /usr/local/newrepos/myproj/ $
```

Vor dem Import enthielt das Archiv nur seinen Verwaltungsbereich CVSROOT. Nach dem Import erschien ein neues Verzeichnis - **myproj**. Die Dateien und Unterverzeichnisse in diesem neuen Verzeichnis sehen verdächtig dem Projekt ähnlich, das wir importiert haben - mit der Ausnahme, dass die Dateien das Suffix ",v" besitzen. Dies sind Versionskontrolldateien im RCS-Format (das ",v" steht für "version"), und sie sind das Rückgrat des Archivs. Jede RCS-Datei speichert die Revisionshistorie der korrespondierenden Datei im Projekt inklusive aller Verzweigungen und Markierungen.

Sie müssen das RCS-Format nicht kennen, um CVS zu benutzen (auch wenn es eine exzellente Dokumentation dazu in der Quelltextdistribution gibt, siehe [doc/RCSFILES](#)). Ein grundlegendes Verständnis des Formates kann jedoch von großer Hilfe bei der Beseitigung von CVS-Problemen sein, also werden wir einen kurzen Blick in eine der Dateien - **hello.c,v** - werfen. Der Inhalt sieht so aus:

hello.c,v
<pre>head 1.1; branch 1.1.1; access ; symbols start:1.1.1.1 jrandom:1.1.1; locks ; strict; comment @ * @; 1.1 date 99.06.20.17.47.26; author jrandom; state Exp; branches 1.1.1.1; next; 1.1.1.1 date 99.06.20.17.47.26; author jrandom; state Exp; branches ; next; desc @@ 1.1 log @Initial revision @ text @#include <stdio.h> void main () { printf ("Hello, world!\n"); } @ 1.1.1.1 log @initial import into CVS @ text @@</pre>

Uff! Das meiste davon können Sie ignorieren; machen Sie sich zum Beispiel keine Gedanken über den Zusammenhang zwischen 1.1 und 1.1.1.1 oder der implizierten 1.1.1-Verzweigung - sie sind vom Standpunkt des Benutzers oder gar Administrators nicht wirklich von Bedeutung. Sie sollten einfach versuchen, das Format an sich zu verstehen. Am Anfang ist eine Ansammlung von Header-Dateien:

hello.c,v
<pre>head 1.1; branch 1.1.1; access ; symbols start:1.1.1.1 jrandom:1.1.1; locks ; strict; comment @ * @;</pre>

Weiter unten in der Datei sind Gruppen von Metainformationen über jede Revision (die aber immer noch nicht den Inhalt der Revision zeigen), so z.B.:

hello.c,v
1.1 date 99.06.20.17.47.26; author jrandom; state Exp; branches 1.1.1.1; next ;

Und schließlich folgen die Log-Mitteilung und der Text der eigentlichen Revision:

hello.c,v
1.1 log @Initial revision @ text @#include <stdio.h> void main () { printf ("Hello, world!\n"); } @ 1.1.1.1 log @initial import into CVS @ text @@

Wenn Sie genau hinsehen, wird Ihnen auffallen, dass der Inhalt der ersten Revision unter der Überschrift 1.1 gespeichert ist, die Log-Mitteilung aber "Initial revision" (ursprüngliche Revision) ist, wogegen die Log-Mitteilung, die wir eigentlich beim Import verwendet haben "Initial import into CVS" (ursprünglicher Import in CVS) war, die weiter unten unter Revision 1.1.1.1 erscheint. Sie brauchen sich über diese Diskrepanz im Moment keine Gedanken zu machen. Dies passiert, weil Importe ein besonderer Umstand sind: Um wiederholte Importe in dasselbe Projekt mit einem nützlichen Effekt zu versehen, platziert import die ursprüngliche Revision sowohl auf den Hauptzweig als auch auf einen speziellen Zweig (der Grund dafür wird klarer werden, wenn wir herstellerabhängige Verzweigungen in Kapitel 6 betrachten). Bis dahin können Sie 1.1 und 1.1.1.1 als identisch betrachten.

Die Datei enthüllt sogar noch mehr, sobald wir die erste Änderung an **hello.c** per **commit** übertragen:

```
user@linux ~/ $ cvs -Q co myproj
user@linux ~/ $ cd myproj
user@linux ~/myproj/ $ emacs hello.c
```

(make some changes to the file)

```
user@linux ~/myproj/ $ cvs ci -m "print goodbye too"
cvs commit: Examining .
```

```
cvcs commit: Examining a-subdir
cvcs commit: Examining a-subdir/subsubdir
cvcs commit: Examining b-subdir
Checking in hello.c;
/usr/local/newrepos/myproj/hello.c,v <-- hello.c
new revision: 1.2; previous revision: 1.1
done
```

Wenn Sie sich nun `hello.c,v` im Archiv anschauen, können Sie den Effekt des `commit`-Befehls sehen:

hello.c,v
<pre>head 1.2; access; symbols start:1.1.1.1 jrandom:1.1.1; locks; strict; comment @ * @; 1.2 date 99.06.21.01.49.40; author jrandom; state Exp; branches; next 1.1; 1.1 date 99.06.20.17.47.26; author jrandom; state Exp; branches 1.1.1.1; next ; 1.1.1.1 date 99.06.20.17.47.26; author jrandom; state Exp; branches; next ; desc @@ 1.2 log @print goodbye too @ text @#include <stdio.h> void main () { printf ("Hello, world!\n"); printf ("Goodbye, world!\n"); } @ 1.1 log @Initial revision @ text @d7 1 @ 1.1.1.1 log @initial import into CVS @ text @@</pre>

Nun ist der komplette Inhalt der Revision 1.2 in der Datei gespeichert, und der Text für Revision 1.1 wurde

durch die kryptische Formel:

hello.c,v
d7 1

ersetzt.

Der Ausdruck "**d7 1**" ist eine Differenzcodierung, die bedeutet: "Lösche eine Zeile beginnend bei Zeile 7." Anders ausgedrückt, um Revision 1.1 zu erhalten, lösche Zeile 7 in Revision 1.2! Versuchen Sie es für sich selbst durchzuarbeiten. Sie werden erkennen, dass es in der Tat Revision 1.1 erzeugt - es entfernt einfach die Zeile, die wir zur Datei hinzugefügt hatten.

Dies demonstriert das Grundprinzip des RCS-Formats: Es speichert nur die Unterschiede zwischen Revisionen und spart dadurch eine Menge Platz im Vergleich dazu, die Revisionen jeweils komplett zu speichern. Um von der aktuellsten Revision zur vorhergehenden zu gelangen, wendet es einen Patch auf die spätere Revision an mit Hilfe der gespeicherten Differenz. Natürlich bedeutet das, dass mehr Patch-Operationen durchgeführt werden müssen, je weiter Sie in den Revisionen mit Hilfe der gespeicherten Differenzen zurückgehen. (Wenn die Datei z.B. die Revision 1.7 besitzt und von CVS die Revision 1.4 angefordert wird, muss CVS Revision 1.6 durch rückwärts patchen von 1.7, dann 1.5 durch patchen von 1.6 und dann 1.4 durch patchen von 1.5 erzeugen.) Glücklicherweise sind alte Revisionen auch diejenigen, die am seltensten verlangt werden, daher funktioniert das RCS-System in der Praxis ganz gut. Je aktueller die Revision ist, desto einfacher ist es, sie zu erhalten.

Sie müssen nicht wissen, was all die Einträge der Header-Informationen am Anfang der Datei bedeuten. Die Effekte bestimmter Operationen zeigen sich jedoch sehr klar in den Headern, und eine flüchtige Bekanntschaft mit ihnen mag sich als nützlich erweisen.

Wenn Sie eine neue Revision per **commit** in den Stamm übertragen, wird die Marke "head" aktualisiert. (Im vorhergehenden Beispiel wurde sie zu 1.2, nachdem die zweite Revision zu **hello.c** per **commit** übertragen wurde.) Wenn Sie eine Datei als Binärdatei hinzufügen oder an einen symbolischen Namen binden, werden diese Operationen ebenfalls in den Headern verzeichnet. Wir fügen z.B. **foo.jpg** als Binärdatei hinzu und markieren es einige Male:

```
user@linux ~/ $ cvs add -kb foo.jpg
cvs add: scheduling file 'foo.jpg' for addition
cvs add: use 'cvs commit' to add this file permanently

user@linux ~/ $ cvs -q commit -m "added a random image; ask
jrandom@red-bean.com why"
RCS file: /usr/local/newrepos/myproj/foo.jpg,v
done
Checking in foo.jpg;
/usr/local/newrepos/myproj/foo.jpg,v <-- foo.jpg
initial revision: 1.1
done

user@linux ~/ $ cvs tag some_random_tag foo.jpg
T foo.jpg
```

```
user@linux ~/ $ cvs tag ANOTHER-TAG foo.jpg
T foo.jpg
```

Betrachten Sie nun den Header-Abschnitt von `foo.jpg,v` im Archiv:

```
foo.jpg,v
head 1.1;
access;
symbols
ANOTHER-TAG:1.1
some_random_tag:1.1;
locks; strict;
comment @# @;
expand @b@;
```

Beachten Sie das `b` in der `expand`-Zeile am Ende - es kommt daher, dass wir beim Hinzufügen der Datei die Option `-kb` verwendet haben, und bedeutet, dass die Datei keiner Schlüsselwort- oder Zeilenendeersetzung unterzogen wird, die normalerweise während Checkouts und Aktualisierungen geschehen würden, wenn es eine Textdatei wäre. Die Markierungen erscheinen im Abschnitt `symbols`, eine Markierung pro Zeile - beide sind zur ersten Revision hinzugefügt, da wir diese in beiden Fällen markiert hatten. (Dies erklärt auch, warum Markierungsnamen nur Buchstaben, Zahlen, Bindestriche und Unterstriche enthalten dürfen. Wenn die Markierung selbst Punkte oder Doppelpunkte enthielte, könnten die Aufzeichnungen darüber in der RCS-Datei zweideutig sein, da es keine Möglichkeit gibt, die Grenze zwischen der Markierung und der ihr angefügten Revisionsnummer zu erkennen.)

4.4 RCS-Format quotiert »@«-Zeichen immer

Das `@`-Symbol wird in RCS-Dateien als Feldbegrenzer verwendet. Dies bedeutet, dass ein im Text oder der Log-Mitteilung vorkommendes `"@"`-Zeichen quotiert⁵ werden muss (ansonsten würde CVS es fälschlicherweise als Endemarkierung für dieses Feld interpretieren). Es wird durch Verdoppelung quotiert - d.h. CVS interpretiert `@@` immer als "Textzeichen `@`", nie als "Ende des aktuellen Feldes". Als wir `foo.jpg` per `commit` übertragen haben, war die Log-Mitteilung

```
"added a random image; ask jrandom@red-bean.com why"
```

Das wird in `foo.jpg,v` wie folgt gespeichert:

foo.jpg,v
1.1 log @added a random image; ask jrandom@@red-bean.com why @

Das @-Symbol in **jrandom@@red-bean.com** wird automatisch zurückverwandelt, sobald CVS die Log-Mitteilung erstellt:

```
user@linux ~/ $ cvs log foo.jpg
RCS file: /usr/local/newrepos/myproj/foo.jpg,v
Working file: foo.jpg
head: 1.1
branch:
locks: strict
access list:
symbolic names:
ANOTHER-TAG: 1.1
some_random_tag: 1.1
keyword substitution: b
total revisions: 1; selected revisions: 1
description:
-----
revision 1.1
date: 1999/06/21 02:56:18; author: jrandom; state: Exp;
added a random image; ask jrandom@red-bean.com why
=====
user@linux ~/ $
```

Sie müssen dies nur bedenken, wenn Sie einmal RCS-Dateien von Hand editieren müssen (was selten ist, aber durchaus einmal vorkommen kann) - denken Sie dann daran, die @-Zeichen in Revisionsinhalten und Log-Mitteilungen zu verdoppeln. Wenn Sie das nicht tun, wird die RCS-Datei durcheinander geraten und wahrscheinlich ein seltsames und unerwünschtes Verhalten zeigen.

Wo wir über von Hand zu editierende RCS-Dateien reden - lassen Sie sich nicht von den Zugriffsrechten im Archiv in die Irre führen:

```
user@linux ~/ $ ls -l
total 6
-r--r--r-- 1 jrandom users 410 Jun 20 12:47 README.txt,v
drwxrwxr-x 3 jrandom users 1024 Jun 20 21:56 a-subdir/
drwxrwxr-x 2 jrandom users 1024 Jun 20 21:56 b-subdir/
-r--r--r-- 1 jrandom users 937 Jun 20 21:56 foo.jpg,v
-r--r--r-- 1 jrandom users 564 Jun 20 21:11 hello.c,v
user@linux ~/ $
```

(Für diejenigen Leser, die nicht so vertraut mit der Ausgabe des ls-Kommandos unter Unix sind - die Zeilen "-r--r--r--" auf der linken Seite bedeuten im Wesentlichen, dass die Dateien gelesen, aber nicht geändert werden können.) Auch wenn die Dateien für jeden als nur lesbar erscheinen, muss man die Zugriffsrechte auf das Verzeichnis in Betracht ziehen:

```
user@linux ~/ $ ls -ld .  
drwxrwxr-x 4 jrandom users 1024 Jun 20 22:16 ./  
user@linux ~/ $
```

Das Verzeichnis **myproj/** selbst und alle seine Unterverzeichnisse sind für den Benutzer (jrandom) und die Gruppe (users) beschreibbar. Das bedeutet, dass CVS (als Benutzer jrandom oder von jemandem in der Gruppe users ausgeführt) Dateien in diesen Verzeichnissen erzeugen und löschen kann, auch wenn schon vorhandene Dateien nicht direkt editiert werden können. CVS editiert eine RCS-Datei, indem es eine separate Kopie davon anlegt deshalb sollten Sie auch alle Ihre Änderungen an einer temporären Kopie vornehmen und danach die existierende RCS-Datei durch die neue ersetzen. (Fragen Sie aber bitte nicht, warum die Dateien selbst nur lesbar sind - das hat historische Gründe, die mit der Arbeitsweise von RCS als eigenständigem Programm zu tun haben.)

Übrigens ist es wahrscheinlich nicht in Ihrem Sinn, wenn die Gruppe der Dateien "users" ist, wenn man bedenkt, dass das oberste Verzeichnis des Archivs explizit der Gruppe "cvs" zugewiesen wurde. Sie können das Problem beheben, indem Sie das folgende Kommando innerhalb des Archivs ausführen:

```
user@linux ~/ $ cd /usr/local/newrepos  
user@linux ~/ $ chgrp -R cvs myproj
```

Unglücklicherweise regeln die üblichen Unix-Dateierzeugungsregeln nur, welcher Gruppe neue Dateien im Archiv zugeordnet werden, also werden Sie ab und zu die Kommandos **chgrp** oder **chmod** auf bestimmte Dateien oder Verzeichnisse im Archiv ausführen müssen. Es gibt keine festgelegten und einfachen Regeln dafür, wie Archivzugriffsrechte strukturiert werden sollten; es hängt davon ab, wer an welchem Projekt arbeitet.

4.5 Was geschieht, wenn Sie eine Datei löschen

Wenn Sie eine Datei aus einem Projekt löschen, verschwindet diese nicht einfach. CVS muss dazu in der Lage sein, solche Dateien wiederherzustellen, wenn Sie einen älteren Schnappschuss des Projektes anfordern. Stattdessen wird die Datei sozusagen auf den "Dachboden" (Attic) geräumt:

```
user@linux ~/src/myproj/ $ pwd  
/home/jrandom/src/myproj  
user@linux ~/src/myproj/ $ ls /usr/local/newrepos/myproj/  
README.txt,v a-subdir/ b-subdir/ foo.jpg,v hello.c,v  
user@linux ~/src/myproj/ $ rm foo.jpg  
user@linux ~/src/myproj/ $ cvs rm foo.jpg  
cvs remove: scheduling 'foo.jpg' for removal  
cvs remove: use 'cvs commit' to remove this file permanently  
user@linux ~/src/myproj/ $ cvs ci -m "Removed foo.jpg" foo.jpg
```

```
Removing foo.jpg;
/usr/local/newrepos/myproj/foo.jpg,v <-- foo.jpg
new revision: delete; previous revision: 1.1
done

user@linux ~/src/myproj/ $ cd /usr/local/newrepos/myproj/
user@linux /usr/local/newrepos/myproj/ $ ls

Attic/ README.txt,v a-subdir/ b-subdir/ hello.c,v

user@linux /usr/local/newrepos/myproj/ $ cd Attic
user@linux /usr/local/newrepos/myproj/Attic/ $ ls

foo.jpg,v

user@linux / $
```

In jedem Archivverzeichnis eines Projekts bedeutet die Existenz eines **Attic/**-Unterverzeichnisses, dass mindestens eine Datei aus diesem Verzeichnis gelöscht wurde. (Dies bedeutet, dass Sie keine Verzeichnisse namens **Attic/** in Ihrem Projekt verwenden sollten.) CVS bewegt aber nicht einfach die RCS-Datei nach **Attic/**; es wird auch eine neue Version dieser Datei mit einem speziellen Revisionsstand namens "dead" (engl. = tot) per **commit** erzeugt. Dies ist der entsprechende Ausschnitt aus **Attic/foo.jpg,v**:

Attic/foo.jpg,v
1.2 date 99.06.21.03.38.07; author jrandom; state dead; branches; next 1.1;

Wenn die Datei später wieder zum Leben erweckt wird, kann CVS aufzeichnen, dass die Datei irgendwann "tot" gewesen ist und nun wieder zum Leben erweckt wurde.

Wenn Sie also eine gelöschte Datei wiederherstellen wollen, können Sie sie nicht einfach aus **Attic/** herausnehmen und wieder in das Projekt zurückbewegen. Stattdessen müssen Sie ungefähr Folgendes in einer Arbeitskopie durchführen:

```
user@linux ~/src/myproj/ $ pwd
/home/jrandom/src/myproj

user@linux ~/src/myproj/ $ cvs -Q update -p -r 1.1 foo.jpg > foo.jpg
user@linux ~/src/myproj/ $ ls

CVS/ README.txt a-subdir/ b-subdir/ foo.jpg hello.c

user@linux ~/src/myproj/ $ cvs add -kb foo.jpg

cvs add: re-adding file foo.jpg (in place of dead revision 1.2)
cvs add: use 'cvs commit' to add this file permanently
cvs ci -m "revived jpg image" foo.jpg
```

```
Checking in foo.jpg;
/usr/local/newrepos/myproj/foo.jpg,v <-- foo.jpg
new revision: 1.3; previous revision: 1.2
done

user@linux ~/src/myproj/ $ cd /usr/local/newrepos/myproj/
user@linux /usr/local/newrepos/myproj/ $ ls

Attic/ a-subdir/
README.txt,v b-subdir/ hello.c,v

user@linux /usr/local/newrepos/myproj/ $ ls Attic/
user@linux /usr/local/newrepos/myproj/ $
```

Es gibt noch einiges weitere Wissenswerte über das RCS-Format, doch das reicht aus, damit ein CVS-Administrator ein Archiv betreuen kann. Es kommt sehr selten vor, dass tatsächlich eine RCS-Datei editiert werden muss; Sie werden normalerweise nur Zugriffsrechte im Archiv anpassen müssen, zumindest meiner Erfahrung nach. Wenn CVS aber doch einmal beginnt, sich wirklich seltsam zu verhalten (das ist selten, aber nicht absolut unmöglich), sollten Sie tatsächlich einmal in die RCS-Dateien sehen, um herauszufinden, was vor sich geht.

4.6 Das CVSR00T/-Administrationsverzeichnis

Die Dateien in **newrepos/CVSR00T/** sind nicht Teil eines Projektes, sondern werden dazu verwendet, das Verhalten von CVS im Archiv zu steuern. Der beste Weg, diese Dateien zu editieren, ist es, eine Arbeitskopie von **CVSR00T/** genau wie ein reguläres Projekt auszuchecken:

```
user@linux ~/ $ cvs co CVSR00T

cvs checkout: Updating CVSR00T
U CVSR00T/checkoutlist
U CVSR00T/commitinfo
U CVSR00T/config
U CVSR00T/cvswrappers
U CVSR00T/editinfo
U CVSR00T/logininfo
U CVSR00T/modules
U CVSR00T/notify
U CVSR00T/rcsinfo
U CVSR00T/taginfo
U CVSR00T/verifymsg
```

Wir betrachten die Dateien in ungefährender Reihenfolge ihrer Wichtigkeit. Jede der Dateien kommt mit einem erklärenden Kommentar am Anfang. (Die Kommentierungskonvention ist in allen Dateien gleich: Ein "#" -Zeichen am Anfang der Zeile kennzeichnet einen Kommentar, und CVS ignoriert solche Zeilen beim Lesen der Datei.) Denken Sie daran, dass alle Änderungen an der Arbeitskopie der administrativen Dateien erst dann für CVS wirksam werden, wenn Sie die Änderungen per **commit** übertragen.

Tipp

Wenn Sie extreme Sicherheitsbedenken haben, sollten Sie die Unix-Zugriffsrechte auf CVSR00T unterschiedlich zu den Zugriffsrechten anderswo im Archiv halten, um genaue Kontrolle darüber zu haben, welcher Benutzer Änderungen an CVSR00T vornehmen kann. Wie Sie ein wenig später sehen werden, gibt die

Fähigkeit, Dateien in CVSROOT zu verändern, im Prinzip jedem CVS-Benutzer - auch entfernten - die Möglichkeit, beliebige Kommandos auf der Archivmaschine auszuführen.

Die Datei config

Die Konfigurationsdatei legt gewisse globale Verhaltensparameter fest. Sie hat ein sehr genau einzuhaltendes Format:

config
PARAMETER=WERT

(usw.) ohne zusätzlich erlaubte Leerzeichen. Hier ist ein Beispiel für eine Konfigurationsdatei:

config
SystemAuth=yes TopLevelAdmin=no PreservePermissions=no

(Ein fehlender Eintrag ist gleichbedeutend mit "no".)

Der Parameter SystemAuth legt fest, ob CVS in der System-passwd-Datei nachsehen soll, wenn es einen gegebenen Benutzernamen nicht in **CVSROOT/passwd** finden kann. CVS-Distributionen werden mit diesem Parameter auf "no" gesetzt ausgeliefert - zum Wohle Ihrer Systemsicherheit.

TopLevelAdmin steuert, ob CVS beim Checkout einer Arbeitskopie auch ein **CVS/-**Verzeichnis anlegt. Dieses **CVS/-**Verzeichnis wäre nicht innerhalb der Arbeitskopie, sondern auf gleicher Verzeichnisebene. Es mag bequem sein, dies zu aktivieren, wenn Sie (und die Benutzer Ihres Archivs) dazu neigen, viele verschiedene Projekte aus dem gleichen Archiv auszuchecken. Ansonsten sollten Sie es ausgeschaltet lassen, da es verwirren kann, ein unerwartetes zusätzliches **CVS/-**Verzeichnis zu sehen.

PreservePermissions kontrolliert, ob Dateizugriffsrechte und ähnliche Metadaten in der Revisionshistorie erhalten bleiben. Dies ist eine etwas obskure Funktionalität, und es lohnt sich wahrscheinlich nicht, sie im Detail zu beschreiben. Schauen Sie unter dem Knoten "Special Files" im Cederqvist nach, wenn Sie daran interessiert sind.

Tipp

"Knoten" (Node) ist Texinfo-Jargon für einen bestimmten Ort innerhalb eines Info-Dokuments. Um zu einem bestimmten Knoten beim Lesen eines Info-Dokuments zu gelangen, tippen Sie einfach an einer beliebigen Stelle des Dokuments "g" gefolgt vom Namen des Knotens.

LockDir ist auch eine selten benutzte Funktion. Unter bestimmten Umständen möchten Sie CVS dazu bewegen, dass es seine Lock-Dateien an einer anderen Stelle als direkt in den Projektunterverzeichnissen anlegt, um Probleme mit Zugriffsrechten zu vermeiden. Diese Lock-Dateien halten CVS davon ab sich selbst auf die Füße zu treten, wenn mehrere Operationen gleichzeitig auf demselben Archiv durchgeführt werden. Normalerweise müssen Sie sich nicht darum kümmern, doch manchmal können Benutzer Probleme haben, ein Archiv zu aktualisieren oder auszuchecken, weil sie keine Lock-Datei erzeugen können. (CVS muss auch für nur lesende

Operationen eine Lock-Datei anlegen, um Situationen zu vermeiden, in denen eine Instanz von CVS liest, während eine andere Instanz schreibt.) Normalerweise ist die Lösung dafür, die Archivzugriffsrechte zu ändern; wenn das aber nicht machbar ist, kann die LockDir-Funktionalität nützlich sein.

Die modules-Datei

In der Datei **modules** können Sie Aliase und andere Gruppierungen für im Archiv befindliche Projekte festlegen. Die grundlegendste Zeile in modules sieht wie folgt aus:

modules
MODUL_NAME MODUL_IM_ARCHIV

zum Beispiel

modules
mp myproj asub myproj/a-subdir

(Die Pfadangaben auf der rechten Seite sind relativ zum Anfang des Archivs.). Dies gibt Entwicklern einen anderen Namen, über den sie ein Projekt oder einen Teil eines Projektes per **checkout** erhalten können:

```
user@linux ~/ $ cvs co mp
cvs checkout: Updating mp
U mp/README.txt
U mp/foo.jpg
U mp/hello.c
cvs checkout: Updating mp/a-subdir
U mp/a-subdir/whatever.c
cvs checkout: Updating mp/a-subdir/subsubdir
U mp/a-subdir/subsubdir/fish.c
cvs checkout: Updating mp/b-subdir
U mp/b-subdir/random.c
```

oder

```
user@linux ~/ $ cvs -d /usr/local/newrepos/ co asub
cvs checkout: Updating asub
U asub/whatever.c
cvs checkout: Updating asub/subsubdir
U asub/subsubdir/fish.c
```

In beiden Fällen wurde der Name des für die Arbeitskopie erzeugten Verzeichnisses gleich dem Namen des Moduls gesetzt. Im Fall von **asub** hat CVS sogar das dazwischen liegende **myproj/-**Verzeichnis weggelassen und direkt ein Verzeichnis **asub/** erzeugt, auch wenn es aus dem **myproj/a-subdir**-Unterverzeichnis des Projektes kam. Aktualisierungen, Commits und alle anderen CVS-Befehle verhalten sich auf diesen Arbeitskopien wie gewohnt - das einzig Ungewöhnliche daran sind ihre Namen.

Wenn Sie Dateinamen hinter die Verzeichnisnamen schreiben, können Sie ein Modul definieren, das nur aus einigen der Dateien in einem gegebenen Archivverzeichnis besteht. Zum Beispiel würden

```
readme myproj README.txt
```

und

```
no-readme myproj hello.c foo.jpg
```

die folgenden Checkouts zulassen:

```
user@linux ~/ $ cvs -q co readme
U readme/README.txt
user@linux ~/ $ cvs -q co no-readme
U no-readme/hello.c
U no-readme/foo.jpg
```

Sie können ein Modul definieren, das mehrere Archivverzeichnisse umfasst, indem Sie die Option **-a** (für "Alias") verwenden, doch denken Sie daran, dass diese Verzeichnisse unter ihrem ursprünglichen Namen ausgecheckt werden. Zum Beispiel lässt folgende Zeile

```
twoproj -a myproj yourproj
```

zu, dass Sie Folgendes tun können (unter der Annahme, dass sowohl **myproj/** wie auch **yourproj/** im Archiv sind):

```
user@linux ~/ $ cvs co twoproj
U myproj/README.txt
U myproj/foo.jpg
U myproj/hello.c
U myproj/a-subdir/whatever.c
U myproj/a-subdir/subsubdir/fish.c
```

```
U myproj/b-subdir/random.c
U yourproj/README
U yourproj/foo.c
U yourproj/some-subdir/file1.c
U yourproj/some-subdir/file2.c
U yourproj/some-subdir/another-subdir/blah.c
```

Der Name "twoproj" war ein praktischer Überbegriff für beide Projekte, er betraf aber nicht die Namen der Arbeitskopien. (Nebenbei gibt es keine Notwendigkeit, dass sich Aliasmodule auf mehrere Verzeichnisse beziehen; wir hätten **twoproj** weglassen können - in diesem Fall wäre **myproj** immer noch unter dem Namen "myproj" ausgecheckt worden.)

Module können auch auf andere Module verweisen, indem man sie mit einem kaufmännischen "und"-Symbol ("&") versteht:

```
mp myproj
asub myproj/a-subdir
twoproj -a myproj yourproj
tp &twoproj
```

Ein Checkout von **tp** hat nun genau den gleichen Effekt wie ein Checkout von **twoproj**.

Es gibt einige weitere Tricks, die Sie mit Modulen anstellen können, die meisten davon werden seltener als die gerade vorgestellten verwendet. Weitere Informationen darüber finden Sie im Knoten modules im Cederqvist.

Die Dateien commitinfo, loginfo und rcsinfo

Die meisten der anderen administrativen Dateien stellen Eingriffsmöglichkeiten in verschiedene Teile des **commit**-Prozesses zur Verfügung (z.B. die Möglichkeit, Log-Mitteilungen oder Dateizustände zu überprüfen, bevor der **commit** erlaubt wird, oder die Möglichkeit, eine Gruppe von Entwicklern zu informieren, sobald ein **commit** in einem bestimmten Verzeichnis des Archivs stattfindet).

Die Dateien haben eine gemeinsame Syntax. Jede Zeile hat die Form:

```
REGULÄRER_AUSDRUCK ZU_STARTENDES_PROGRAMM
```

Der reguläre Ausdruck wird auf das Verzeichnis angewandt, in dem der **commit** stattfindet (dabei ist der Verzeichnisname relativ zum Basisverzeichnis des Archivs). Wenn eine Übereinstimmung erkannt wird, wird das dafür vorgesehene Programm ausgeführt. Dem Programm werden die Namen aller vom **commit** betroffenen Dateien übergeben; es kann beliebige Dinge mit diesen Namen machen, auch die Dateien öffnen und ihren Inhalt begutachten. Wenn das Programm mit einem Rückgabewert ungleich null zurückkehrt, wird verhindert, dass der **commit** stattfindet.

Die **Commitinfo**-Datei stellt eine grundlegende Eingriffsmöglichkeit zur Verfügung und bei jedem Commit ausgewertet. Hier sind einige Beispiele für **Commitinfo**-Zeilen:

Commitinfo
<pre>^a-subdir.* /usr/local/bin/check-asubdir.sh ou /usr/local/bin/validate-project.pl</pre>

Tipp

Reguläre Ausdrücke sind ein System, um Klassen von Zeichenketten exakt zu beschreiben. Wenn Sie nicht mit regulären Ausdrücken vertraut sind, hier eine kurze Zusammenfassung, die für unsere Zwecke ausreicht: "foo" passt auf alle Dateien, deren Name den String "foo" enthält, und "foo.*bar" passt auf alle Dateien, deren Name "foo", gefolgt von einer beliebigen Anzahl Zeichen und gefolgt vom String "bar", enthält. Dies funktioniert, da normale Teilstrings auf sich selbst passen, aber "." und "*" spezielle Zeichen sind. "." passt auf jedes Zeichen, und "*" bedeutet, dass eine beliebige Anzahl des vorhergehenden Zeichens (inklusive Null) passen darf. Die Zeichen "^" und "\$" bedeuten, dass der passende String am Anfang bzw. Ende stehen muss; deshalb passt "^foo.*bar.*baz\$" auf jeden String, der mit "foo" beginnt, irgendwo in der Mitte "bar" enthält und mit "baz" endet. Wir werden hier nicht tiefer in die Materie einsteigen; diese Zusammenfassung ist eine sehr stark abgekürzte Teilmenge der kompletten Syntax der regulären Ausdrücke.

Also passt jeder commit in das Verzeichnis `myproj/a-subdir/` auf die erste Zeile, was die Ausführung des Skripts `check-asubdir.sh` veranlassen würde. Ein commit in einem Projekt, dessen Name (tatsächlicher Archivverzeichnisname, nicht notwendigerweise Modulname) die Zeichenfolge "ou" enthält, würde das `validate-project.pl`-Skript aufrufen, außer der `commit` hätte schon auf die vorhergehende `a-subdir`-Zeile gepasst.

Anstelle eines regulären Ausdrucks können die Begriffe DEFAULT oder ALL verwendet werden. Die Zeile DEFAULT (oder die erste DEFAULT-Zeile, falls es mehrere gibt) wird genommen, wenn kein regulärer Ausdruck passt, und die Programme aller ALL-Zeilen werden zusätzlich zu denen aller passenden Zeilen gestartet.

Bemerkung

Die dem Programm übergebenen Dateinamen beziehen sich nicht auf RCS-Dateien - sie zeigen auf normale Dateien, deren Inhalt exakt identisch mit den Arbeitskopien ist, die per `commit` übertragen wurden. Das einzige Ungewöhnliche daran ist, dass CVS sie temporär innerhalb des Archivs platziert hat, damit sie für Programme verfügbar sind, die auf der Archivmaschine laufen.

Die `loginfo`-Datei ist ähnlich zu `commitinfo`, aber anstelle von Operationen auf den Inhalt der Datei wird mit der Log-Mitteilung gearbeitet. Die linke Seite der `loginfo`-Datei enthält reguläre Ausdrücke inklusive möglicher DEFAULT- und ALL-Zeilen. Das auf der rechten Seite stehende Programm erhält die Log-Mitteilung auf seiner Standardeingabe; es kann mit der Eingabe machen, was immer es mag.

Das Programm auf der rechten Seite kann auch eine beliebige Zahl von Kommandozeilenparametern übergeben bekommen. Eines dieser Argumente kann ein spezielles "%" -Zeichen sein, das zur Laufzeit von CVS wie folgt expandiert wird:

```
%s -----> Name(n) der Datei(en), die per commit übertragen wird
%V -----> Revisionsnummer vor dem commit
%v -----> Revisionsnummer nach dem commit
```

Die Ersetzung beginnt immer mit dem Pfad zum Verzeichnis (aus Gründen der Rückwärtskompatibilität), gefolgt von der Information pro Datei. Wenn zum Beispiel die Dateien `foo`, `bar` und `baz` per `commit` übertragen würden, würde `%s` ersetzt durch:

```
myproj foo bar baz
```

wogegen `%V` durch ihre alten Revisionsnummern ersetzt würde:

```
myproj 1.7 1.134 1.12
```

und `%v` zu ihren neuen Revisionsnummern:

```
myproj 1.8 1.135 1.13
```

Es darf in der Logininfo-Datei pro Zeile nur ein `"%"`-Ausdruck vorkommen. Wenn Sie mehr als eines dieser Zeichen verwenden möchten, müssen Sie sie nach dem `"%"`-Zeichen in geschweifte Klammern setzen - dies ersetzt sie durch eine Liste von durch Komma-separierten Unterlisten, von denen jede die entsprechende Information für eine Datei des commits enthält. Zum Beispiel würde `%{sv}` ersetzt durch

```
myproj foo,1.8 bar,1.135 baz,1.13
```

wogegen `%{sVv}` durch

```
myproj foo,1.7,1.8 bar,1.134,1.135 baz,1.12,1.13
```

ersetzt würde. (Sehen Sie genau hin, damit Sie die Kommas von den Punkten in diesem Beispiel unterscheiden können.)

Hier ist ein Beispiel für eine `logininfo`-Datei:

loginfo
<pre>^myproj&#36; /usr/local/newrepos/CVSR00T/log.pl -m myproj-devel@foobar.com %s ou /usr/local/bin/ou-notify.pl %{sv} DEFAULT /usr/local/bin/default-notify.pl %{sVv}</pre>

In der ersten Zeile steht, dass jeder **commit** in das Unterverzeichnis **myproj** des Archivs das Skript "**log.pl**" aufruft, ihm eine E-Mail-Adresse übergibt (an die **log.pl** eine Mail mit der Log-Mitteilung senden wird), gefolgt vom Archiv und allen Dateien im **commit**.

In der zweiten Zeile wird festgelegt, dass jeder **commit** in ein Unterverzeichnis, das den String "ou" enthält, das (imaginäre) Skript "**ou-notify.pl**" aufrufen wird und ihm das Archiv gefolgt von den Dateinamen und neuen Revisionsnummern aller Dateien im **commit** übergeben wird.

Die dritte Zeile ruft das (genauso imaginäre) Skript **default-notify.pl** für jeden **commit** auf, der nicht auf die vorherigen zwei Zeilen gepasst hat, und übergibt diesem alle möglichen Informationen (Pfad zum Verzeichnis, Dateinamen, alte Revisionen, neue Revisionen).

Commit-E-Mails

In der **loginfo**-Datei werden commit-E-Mails konfiguriert - automatisierte E-Mails, die an jeden Mitarbeiter des Projektes gehen, wenn ein **commit** stattfindet. (Es mag gegen die Intuition sein, dass dies in **loginfo** und nicht in **commitinfo** geschieht, doch der Grund dafür ist, dass man die Log-Mitteilung in die E-Mail schreiben will). Das Programm, das die Mails versendet - **contrib/log.pl** in der CVS Quelltextdistribution - kann an einer beliebigen Stelle in Ihrem System installiert werden. Ich kopiere es normalerweise in das **CVSR00T**/-Unterverzeichnis des Archivs, aber das ist reine Geschmackssache.

Es kann sein, dass Sie **log.pl** ein wenig anpassen müssen, um es auf Ihrem System ans Laufen zu bekommen, eventuell müssen Sie die erste Zeile auf die Position Ihres Perl-Interpreters anpassen und vielleicht die folgende Zeile ändern:

log.pl
<pre>&#36;mailcmd = " Mail -s 'CVS update: &#36;modulepath'";</pre>

um Ihr bevorzugtes Mail-Programm aufzurufen, das vielleicht "Mail" heißt - oder vielleicht auch nicht. Sobald Sie es nach Ihren Wünschen angepasst haben, können Sie Zeilen wie diese in Ihre **loginfo**-Datei schreiben:

loginfo
<pre>listerizer CVSR00T/log.pl %s -f CVSR00T/commitlog -m listerizer@red-bean.com RoadMail CVSR00T/log.pl %s -f CVSR00T/commitlog -m roadmail@red-bean.com bk/*score CVSR00T/log.pl %s -f CVSR00T/commitlog -m bkscore-devel@red-bean.com</pre>

Das %s wird durch die Namen der Dateien ersetzt, die per `commit` übertragen wurden; die Option `-f` zu `log.pl` übergibt den Dateinamen, an den die Log-Mitteilung angehängt wird (damit `CVSR00T/commitlog` eine stetig wachsende Datei von Log-Mitteilungen wird), und die Option `-m` übergibt eine E-Mail-Adresse, an die `log.pl` eine Nachricht über den `commit` senden wird. Die Adresse ist normalerweise eine Mailingliste, Sie können aber die `-m`-Option so oft in einer `log.pl`-Befehlszeile verwenden, wie Sie sie benötigen.

Die Dateien `verifymsg` und `rcsinfo`

Manchmal möchte man vielleicht nur ein Programm haben, das automatisch überprüft, ob die Log-Mitteilung einem bestimmten Standard entspricht und den `commit` verweigert, wenn dieser Standard nicht erfüllt ist. Dies kann man durch die Verwendung von "`verifymsg`", eventuell mit etwas Hilfe von "`rcsinfo`", erreichen.

Die Datei `verifymsg` besteht aus der gewohnten Kombination von regulären Ausdrücken und Programmen. Das Programm erhält die Log-Mitteilung auf der Standardeingabe; es sollte dann einige Prüfungen durchführen, um sicherzustellen, dass die Log-Mitteilung bestimmten Kriterien entspricht, und dann mit einem Status gleich oder ungleich null enden. Im zweiten Fall wird der `commit`-Befehl fehlschlagen.

Auch die linke Seite von `rcsinfo` hat die normalen regulären Ausdrücke, die rechte Seite zeigt aber auf Vorlagedateien⁶ anstelle von Programmen. Eine Vorlagedatei kann in etwa so aussehen:

```
Condition:
Fix:
Comments:
```

oder aus einer anderen Ansammlung von Feldern bestehen, die ein Entwickler ausfüllen muss, um eine gültige Log-Mitteilung zu erzeugen. Diese Vorlage ist nicht sehr nützlich, wenn jeder explizit mit der `-m`-Option per Commit übertragen wird, viele Entwickler ziehen es aber vor, dies nicht zu tun. Sie verwenden stattdessen

```
user@linux ~/ $ cvs commit
```

und warten darauf, dass CVS automatisch einen Texteditor startet (der in der `EDITOR`-Umgebungsvariablen angegeben ist). Dort schreiben sie ihre Log-Mitteilung, speichern die Datei und verlassen den Editor, worauf CVS mit dem Commit fortfährt.

In dieses Szenario würde eine `rcsinfo`-Vorlage in den zu editierenden Text eingefügt werden, bevor der Benutzer mit der Eingabe beginnen kann, also würden die Felder zusammen mit einer Erinnerung, sie auszufüllen, vorgegeben.

Wenn der Benutzer dann Dateien per Commit überträgt, wird das entsprechende Programm in `verifymsg` aufgerufen. Es sollte dann prüfen, ob die Meldung diesem Format entspricht, und sein Rückgabewert gibt das Ergebnis dieser Prüfung wieder (wobei der Wert null Erfolg bedeutet).

Als Hilfestellung für die Prüfprogramme wird der Pfad zur Vorlage auf der `rcsinfo`-Datei als letzter Parameter an die Kommandozeile des Programms in `verifymsg` angehängt; auf diese Art kann das Programm - wenn gewünscht - die Prüfung auf der Vorlage selbst basieren.

Wenn jemand eine Arbeitskopie auf einer entfernten Maschine auscheckt, dann wird auch die entsprechende `rcsinfo`-Vorlagedatei an den Client gesendet (sie wird im `CVS/`-Unterverzeichnis der Arbeitskopie gespeichert). Dies bedeutet jedoch, dass der Client die Änderungen nicht sieht, ohne das Projekt neu

auszuchecken (ein einfaches Update funktioniert nicht).

Denken Sie daran, dass das Schlüsselwort ALL in der `verifymsg`-Datei nicht unterstützt wird (DEFAULT allerdings wird unterstützt). Das macht es einfacher, Standardverifikationsskripte durch solche zu ersetzen, die spezifisch für ein Unterverzeichnis sind.

Die Datei taginfo

Was `loginfo` für Log-Mitteilungen ist, ist `taginfo` für Markierungen. Die linke Seite von `taginfo` besteht, wie gehabt, aus regulären Ausdrücken und die rechte Seite aus Programmnamen. Jedem Programm werden automatisch Argumente übergeben, wenn `cvstag` aufgerufen wird, und zwar in der folgenden Reihenfolge:

- * arg 1: Name des Tags
- * arg 2: Operation ("add" => tag, "mov" => tag -F, "del" => tag -d)
- * arg 3: Archiv
- * arg 4, 5 usw.: Dateirevision [Dateirevision ...]

Wenn das Programm nicht den Wert null zurückgibt, wird die Markierung abgebrochen.

Wir haben bisher die -F-Option von tag noch nicht behandelt, doch sie tut genau das, was oben impliziert ist: eine Markierung von einer Revision zu einer anderen zu bewegen. Wenn die Markierung "Das_funktioniert" an Revision 1.7 einer Datei gebunden ist und Sie sie stattdessen zu Revision 1.11 hinzufügen möchten, müssen Sie folgenden Befehl ausführen

```
user@linux ~/ $ cvs tag -r 1.11 -F Das_funktioniert foo.c
```

der die Markierung von 1.7 entfernt (oder wo auch immer es vorher in dieser Datei gewesen war) und zu 1.11 hinzufügt.

Die Datei cvswrappers

Die Datei `cvswrappers` ermöglicht es, einige Dateien abhängig von ihrem Dateinamen als Binärdateien zu behandeln. CVS nimmt beispielsweise nicht an, dass alle .jpg-Dateien auch wirklich JPG-Bilddaten sind, also verwendet es nicht automatisch -kb, wenn es JPG-Dateien hinzufügt. Dennoch könnte es für einige Projekte nützlich sein, einfach alle JPG-Dateien zu Binärdateien zu erklären. Diese Zeile in `cvswrappers` tut genau das:

```
*.jpg -k 'b'
```

Das 'b' ist separat und in Anführungszeichen, da dies nicht die einzige mögliche RCS-Schlüsselwort-Ersetzungsmethode ist; man könnte auch 'o' angeben, was bedeutet, dass Schlüsselwörter mit \$-Zeichen nicht ersetzt werden, sondern eine Zeilenendeersetzung stattfinden soll. 'b' ist jedoch der am häufigsten verwendete Parameter.

Es gibt noch einige andere Modi, die in der wrappers-Datei angegeben werden können, doch diese sind für so seltene Situationen gedacht, dass es sich nicht lohnt, sie hier zu dokumentieren (sprich: Ihr Autor hat sie noch nie verwenden müssen). Siehe den Knoten "Wrappers" im Cederqvist, falls Sie Genaueres wissen wollen.

Die Datei editinfo

Diese Datei ist überflüssig, auch wenn sie immer noch in Distributionen vorhanden ist. Ignorieren Sie sie

einfach.

Die Datei notify

Diese Datei wird zusammen mit der "watch"-Funktionalität von CVS verwendet, die in Kapitel 6 beschrieben wird. Nichts davon hat Sinn, bis Sie verstanden haben, was watches sind (sie sind eine nützliche, aber nicht essenzielle Funktion) - schlagen Sie also in Kapitel 6 nach, um Details über diese Datei und watches zu erfahren.

Die Datei checkoutlist

Wenn Sie in **CVSR00T/** hineinschauen, werden Sie sehen, dass die Arbeitskopien der Dateien zusammen mit ihren RCS-Revisionsdateien vorhanden sind:

```
user@linux ~/ $ ls /usr/local/newrepos/CVSR00T
checkoutlist config,v history notify taginfo
checkoutlist,v cvswrappers loginfo notify,v taginfo,v
commitinfo cvswrappers,v loginfo,v passwd verifymsg
commitinfo,v editinfo modules rcsinfo verifymsg,v
config editinfo,v modules,v rcsinfo,v
```

CVS beachtet nur die Arbeitsversionen und nicht die RCS-Dateien, um herauszufinden, wie es sich verhalten soll. Deshalb aktualisiert CVS automatisch alle geänderten Dateien im Archiv, sobald Sie einen **commit**-Befehl auf Ihrer Arbeitskopie durchführen (die ja schließlich auf einer anderen Maschine ausgecheckt sein kann). Sie erfahren dies dadurch, dass CVS eine Meldung am Ende solcher Commits schreibt:

```
user@linux ~/ $ cvs ci -m "added mp and asub modules" modules
Checking in modules;
/usr/local/newrepos/CVSR00T/modules,v <-- modules
new revision: 1.2; previous revision: 1.1
done
cvs commit: Rebuilding administrative file database
```

CVS kennt die standardisierten administrativen Dateien selbst und baut sie - falls nötig - in **CVSR00T/** neu. Wenn Sie eigene Dateien in **CVSR00T/** ablegen (wie z.B. Programme oder **rcsinfo**-Vorlagedateien), müssen Sie CVS explizit befehlen, sie genauso zu behandeln.

Dies ist der Zweck der **checkoutlist**-Datei. Ihr Format unterscheidet sich von dem der meisten anderen hier betrachteten Dateien:

```
DATEINAME FEHLERMELDUNG_WENN_DATEI_NICHT_AUSGEHECKT_WERDEN_KANN
```

zum Beispiel

```
log.pl unable to check out / update log.pl in CVSR00T
bugfix.tmpl unable to check out / update bugfig.tmp in CVSR00T
```

Bestimmte Dateien in **CVSROOT** unterliegen traditionell nicht der Revisionskontrolle. Eine davon ist die **history**-Datei, die eine Aufzeichnung aller Aktionen im Archiv zur Verwendung durch den **cvshistory**-Befehl enthält (der Checkouts, Aktualisierungen und Tag-Aktivitäten für eine bestimmte Datei oder ein Projektverzeichnis auflistet). Wenn Sie die History-Datei löschen, hört CVS damit auf, diesen Log zu führen.

Tipp

Manchmal ist die **history**-Datei die Ursache für Probleme mit Dateizugriffsrechten. Der einfachste Weg, diese zu lösen, ist es, die Datei für alle beschreibbar zu machen oder sie einfach zu löschen.

Eine andere administrative Datei, die nicht der Revisionskontrolle unterliegt, ist **passwd**. Der Grund dafür ist, dass das Auschecken über das Netz die Passwörter offen legen könnte (auch wenn sie verschlüsselt sind). Sie müssen auf Grund Ihrer Sicherheitssituation selbst entscheiden, ob Sie **passwd** zur checkoutlist hinzufügen oder nicht; standardmäßig ist es nicht darin enthalten.

Zwei abschließende Bemerkungen zum **CVSROOT**/-Verzeichnis: Wenn Sie einen genügend großen Fehler machen, ist es möglich, eine administrative Datei per **commit** zu übertragen, die dermaßen fehlerhaft ist, dass weitere Commits nicht mehr stattfinden können. Wenn Sie das tun, werden Sie natürlich keine korrigierte Version der administrativen Datei per **commit** übertragen können! Die Lösung ist, die Arbeitskopie des Archivs von Hand zu editieren, um das Problem zu beheben; es kann sein, dass das ganze Archiv nicht benutzbar ist, bis sie den Fehler behoben haben.

Aus Gründen der Sicherheit sollten Sie nur vertrauenswürdigen Benutzern Schreibzugriff auf Ihr **CVSROOT**/-Verzeichnis gestatten (mit "vertrauenswürdig" meine ich, dass Sie sowohl ihren Absichten wie auch ihren Fähigkeiten, ihr Passwort nicht zu verraten, trauen). Die ***info**-Dateien ermöglichen es, beliebige Programme aufzurufen, sodass jeder, der Dateien in das **CVSROOT**/-Verzeichnis per **commit** überträgt oder diese editiert, grundsätzlich jedes Programm auf dem System ausführen kann. Daran sollten Sie immer denken.

5 Finden Sie mehr heraus

Auch wenn dieses Kapitel versucht, eine vollständige Einführung in die Installation und Administration von CVS zu geben, habe ich einige Dinge ausgelassen, die entweder zu selten verwendet werden, als dass sie einer Erwähnung wert wären, oder die bereits gut im Cederqvist-Handbuch beschrieben sind. Die zweite Kategorie beinhaltet die Konfiguration weiterer Arten des entfernten Zugriffs: RSH/SSH, kserver (Kerberos 4) und GSSAPI (das neben vielem anderen Kerberos 5 enthält). Für RSH/SSH-Verbindungen muss nichts Spezielles getan werden; es muss nur sichergestellt sein, dass sich der fragliche Benutzer mittels RSH oder SSH auf der Archivmaschine anmelden kann. Wenn sie es können, CVS sowohl auf dem Client wie auf dem Server installiert ist und sie die richtigen Zugriffsrechte haben, um das Archiv direkt auf der Servermaschine zu verwenden, sollten sie dazu in der Lage sein, auf das Archiv von anderen Rechnern mittels der :ext:-Methode zuzugreifen.

Beschreibungen bestimmter spezialisierter Funktionen von CVS habe ich auf spätere Kapitel verlagert, damit sie in einem Kontext eingeführt werden können, in dem ihre Nützlichkeit klar wird. Allgemeine Tipps zur Fehlerbehebung in CVS finden sich in Kapitel 8.

Auch wenn es nicht notwendig ist, das gesamte Cederqvist-Handbuch zu lesen, sollten Sie sich damit vertraut machen; es wird eine unschätzbare Referenz darstellen. Wenn Info aus irgendwelchen Gründen auf Ihrer Maschine nicht funktioniert und Sie das Handbuch nicht ausdrucken möchten, können Sie es online anschauen unter  <http://durak.org/cvswebsites/doc/> oder  www.loria.fr/~molli/cvs/doc/cvs_toc.html

1. Anm. d. Übers.: »\« , im Deutschen auch Rückstrich genannt
2. Anm. d. Übers.: Wenn Sie allerdings - wie unter Unix üblich - ein Mehrbenutzersystem vor sich haben, werden Sie sich durch das Neustarten des Systems keine Freunde machen. Normalerweise sollte das Neustarten nicht notwendig sein.
3. Anm. d. Übers.: Programme, die - normalerweise unerlaubt - Daten auf dem Netzwerk abhören und nach Passwörtern durchsuchen.
4. Anm. d. Übers.: »Leser« bzw. »Schreiber«
5. Anm. d. Übers.: »Quotieren« = besonders kennzeichnen
6. Anm. d. Übers.: Sog. »Templates«