Tilo Riemer ▪ Frank Hemer

CrossVC

Grafische Versionskontrolle mit CVS und Subversion

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Gesamtlektorat: Dr. Markus Wirtz
Satz: Open Source Press (LaTeX)
Umschlaggestaltung: Fritz Design, Erlangen
Gesamtherstellung: Kösel, Krugzell

ISBN-10 3-937514-13-9
ISBN-13 978-3-937514-13-0 http://www.opensourcepress.de

der Ankerpunkt, wiedergefunden werden kann. Es werden also keine Redundanzen (keine Kopie) erzeugt, da sich die Datei aus der Kumulation der Unterschiede, ausgehend von der Kopf-Version, zusammensetzen lässt. Gleiches gilt für das Setzen von Markierungen, womit sowohl das Setzen von Markierungen als auch das Erzeugen von Verzweigungen „billige“ Operationen sind.

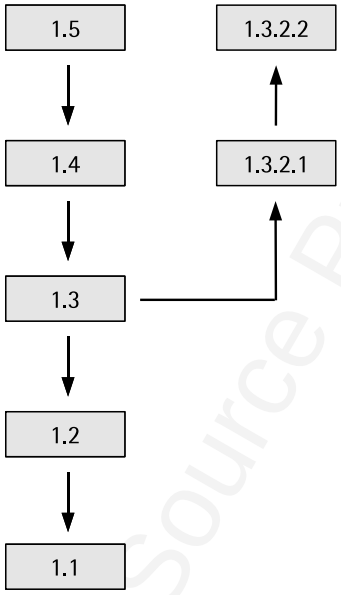


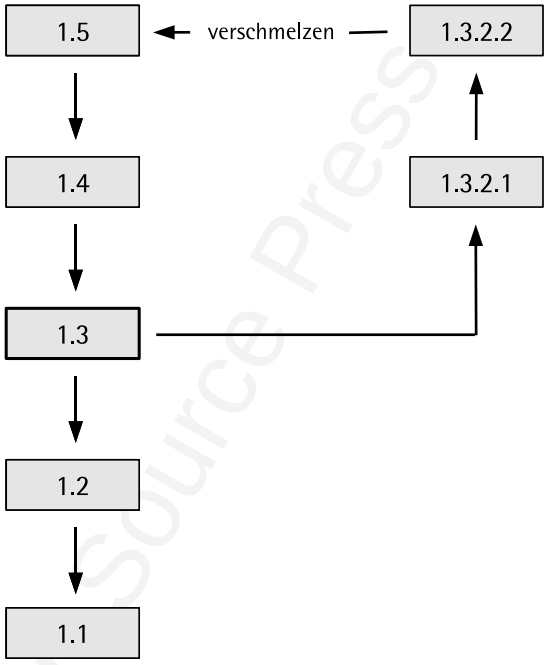
Abbildung 2.4: Patches, Zweige und RCS-Datei

2.3.6 Verzweigen und Verschmelzen

Im Zuge der Entwicklung kommt es immer wieder vor, dass Daten sowohl weiterentwickelt als auch korrigiert werden. Sollen für bestimmte Personengruppen nur die korrigierten Daten zur Verfügung stehen, während andere auf die weiterentwickelten Daten angewiesen sind, kommt es zwangsläufig zu Redundanzen. Da Korrekturen in der Regel nach und nach vorgenommen werden, müssen diese folglich sowohl in den zu korrigierenden Daten als auch in den bereits weiterentwickelten Daten vorgenommen werden. Hier kommen nun die Verzweigungen ins Spiel: Beim Verzweigen wird die Entwicklung der Daten aufgespalten, aber der Bezugspunkt dieser Spaltprodukte, besser der Verzweigungspunkt, bleibt erhalten. So ist es möglich, zu einem späteren Zeitpunkt Änderungen von einem Zweig automatisiert in den anderen Zweig zu integrieren. Dieser Vorgang wird auch als *Verschmelzen* (*merge*) bezeichnet.

Sollen zwei Zweige A und B miteinander verschmolzen werden, ist es zunächst erforderlich, die den beiden Zweigen gemeinsame *Ankerversion* zu bestimmen. Die Ankerversion ist die Version, welche als *Ausgangspunkt* für die zu verschmelzenden Zweige dient, mathematisch ausgedrückt die *größte gemeinsame Version*. Diese ist bei benachbarten Zweigen identisch mit dem Verzweigungspunkt.

Abbildung 2.5: Ankerversion und Verschmelzen



Sobald ein Zweig weitere (Unter-)Zweige hat und nicht die direkt benachbarten Zweige verschmolzen werden, existieren dazwischen mehrere Verzweigungspunkte. Als *Ankerversion* gilt auch hier die *größte gemeinsame Version* der beiden zu verschmelzenden Zweige. Darüber hinaus bleibt das Vorgehen bei Unterverzweigungen jedoch unbeeinflusst.

Ist der Ankerpunkt bestimmt, wird der kumulative Patch aus den Unterschieden zwischen Ankerpunkt und Zweig A berechnet. Diesem wird der kumulative Patch aus den Unterschieden zwischen Ankerpunkt und Zweig B gegenübergestellt. Solange sich dabei keine Überschneidungen ergeben, kann der zu integrierende Patch direkt in den Zielzweig übernommen werden. Treten jedoch Überschneidungen auf, so kann lediglich der konfliktfreie Anteil automatisiert verschmolzen werden. Die sich ergebenden Konflikte müssen vom Anwender manuell aufgelöst werden.

Das Verschmelzen geschieht bei CVS ausschließlich clientseitig. Erst das Ergebnis einer solchen Verschmelzung wird durch den folgenden Commit an das Repository übertragen. Dabei geht die Information darüber, von wo nach wo verschmolzen wurde und unter Zuhilfenahme welchen Ankerpunktes dies geschah, verloren. Soll ein Zweig nun mehrfach verschmolzen werden, zum Beispiel weil nach erfolgter Verschmelzung noch weiter auf eben diesem bereits verschmolzenen Zweig gearbeitet wird, ist die gerade beschriebene Vorgehensweise so nicht wiederholbar. Da CVS die Informationen über die erfolgte Verschmelzung nicht vorliegen, kann neu Hinzugekommenes nicht von Altem und damit bereits Verschmolzenem unterschieden werden. Es würden die bereits verschmolzenen Bereiche nochmals verschmolzen und damit der Anwender mit einer Unzahl resultierender Konflikte konfrontiert.

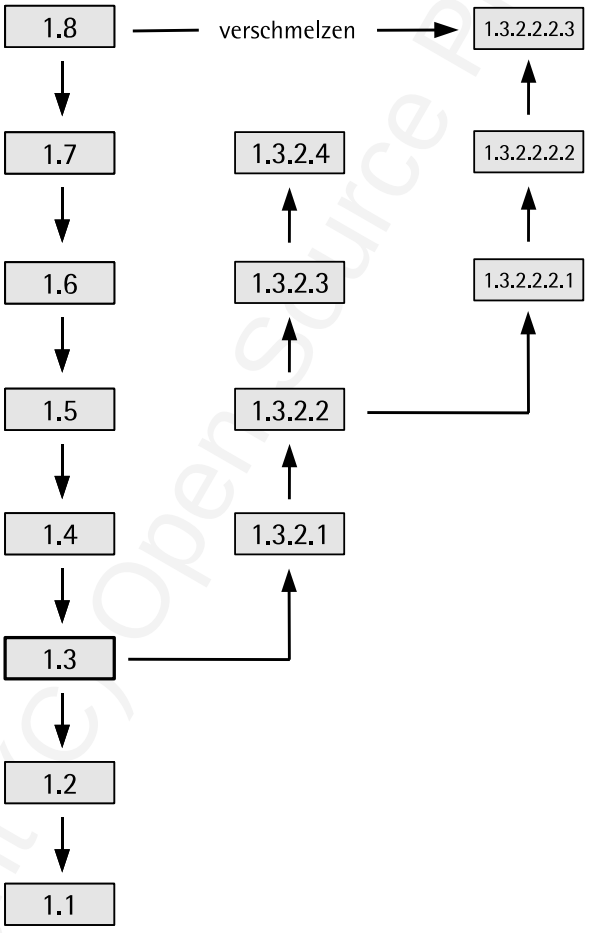


Abbildung 2.6: Ankerversion und Verschmelzen über mehrere Zweige

An dieser Stelle obliegt es dem Anwender, CVS die entscheidenden Informationen zur Verfügung zu stellen. Dies kann geschehen, indem nach erfolgter Erstverschmelzung die aktuelle Version des gerade integrierten Zweiges eine Markierung erhält. Wird diese beim wiederholten Verschmelzen manuell als Ankerpunkt angegeben, so kann, analog zum Erstverschmelzen, automatisiert vorgegangen werden.

CVSNT versucht dem Anwender das Verschmelzen zu erleichtern. Wird hier eine Verschmelzung durchgeführt, so wird clientseitig der dazu verwendete Ankerpunkt zwischengespeichert und beim folgenden Commit zum Repository übertragen. Wird eine einfache Verschmelzung durchgeführt, also lediglich ein Zweig A ein- bis mehrfach mit dem gleichen Zweig B verschmolzen und dabei nach erfolgter Verschmelzung jeweils ein Commit durchgeführt, so ist dies sehr praktisch.

Bei komplexen Verschmelzungen jedoch führt dieses Vorgehen zwangsläufig in eine Sackgasse, da immer nur der letzte Ankerpunkt zwischengespeichert wird. Wird nämlich zunächst Zweig A mit Zweig B verschmolzen und dann das Ergebnis mit Zweig C, ohne zwischendurch einen Commit durchzuführen, so wird der zwischengespeicherte Ankerpunkt der *Zweig-A→Zweig-B*-Verschmelzung während des Verschmelzens mit Zweig C vom *(Zweig-A→Zweig-B)→Zweig-C*-Ankerpunkt überschrieben. Beim darauf folgenden Commit wird nun nicht der *effektive* Ankerpunkt, sondern lediglich der zuletzt verwendete Ankerpunkt zum Repository übertragen. Dies führt dann spätestens beim nächsten Verschmelzen zu unerwarteten Ergebnissen.

Sobald also komplexe Verschmelzungen erforderlich sind, empfiehlt es sich, mit manuellen Ankerpunkten zu arbeiten. Der Mehraufwand steht sicher in keinem Verhältnis zum Aufwand, die andernfalls entstehenden und schwer nachvollziehbaren Konflikte zu lösen.

2.3.7 Zugriffskontrolle

Die Zugriffskontrolle auf das Repository geschieht in Abhängigkeit von der verwendeten Zugriffsmethode und je Verzeichnis. Es existieren Erweiterungen für die Zugriffskontrolle mit Hilfe von *Access Control Lists* (ACL), diese sind jedoch nicht fester Bestandteil von CVS.

lokal
: Die Kontrolle geschieht ausschließlich über die Berechtigungen des installierten Dateisystems. Dabei ist es irrelevant, ob der Anwender auf die einzelne RCS-Datei nur lesenden oder lesenden und schreibenden Zugriff hat. CVS setzt die Berechtigungen nach jedem Zugriff generell auf nur lesend. Gesteuert wird der Zugriff also über die

Flags des Ordners an sich. Wichtig ist dabei, dass bei geplantem nur lesendem Zugriff in der Datei `CVSROOT/config` ein separater Lock-Ordner mit lesendem und schreibendem Zugriff für alle Anwender vorgesehen wird, da bei manchen Kommandos auch bei nur lesendem Zugriff Locks (Sperre)[13] gesetzt werden.

pserver
: Die Kontrolle geschieht hier analog zu dem unter lokal beschriebenen Vorgehen. Dabei kommen die Berechtigungen entsprechend dem in `CVSROOT/passwd` zugeordneten Benutzer zur Anwendung, sofern diese Datei vorhanden ist.

ext
: In diesem Modus wird tatsächlich CVS auf dem Server-Rechner ausgeführt. Die zur Anwendung kommenden Berechtigungen richten sich nach den Berechtigungen des Anwenders auf diesem entfernten Rechner. Rein auf die Berechtigungen bezogen, entspricht ein Zugriff einem lokalen Zugriff auf dem entfernten Rechner.

2.3.8 Dateimodi/Schlüsselwörter

Spätestens beim Hinzufügen einer Datei muss der Anwender sich entscheiden, in welchem Modus diese Datei in Zukunft von CVS verwaltet werden soll. Andernfalls wurde der Modus bereits beim Import festgelegt. Der spezifizierte Modus wird als Standardmodus bei zukünftigen Checkouts auf Client-Seite angewendet. Sollte ein Anwender lokal einen anderen Modus benötigen, so lässt sich dieser Modus in der Arbeitskopie umstellen. Das Repository sowie künftige Checkouts werden davon nicht berührt. Falls versehentlich eine Datei im falschen Modus importiert oder hinzugefügt wurde, gibt es daher die Möglichkeit, den Standardmodus im Repository umzustellen. Diese Möglichkeit sollte jedoch nicht zu einem späteren Zeitpunkt verwendet werden, da der Modus selbst nicht der Versionskontrolle unterliegt. Wird der Dateimodus repositoryseitig zu irgendeinem Zeitpunkt umgestellt, so gilt der neue Modus für alle Versionen dieser Datei uneingeschränkt.

Generell wird der Modus über ein Verfahren mit der missverständlichen Bezeichnung „Schlüsselwortexpansion“ eingerichtet. Schlüsselwörter dienen dem Einbinden von Informationen über den aktuellen Stand der Version direkt in den Dateiinhalt. So wird zum Beispiel durch die Angabe von `$Revision$` innerhalb einer CVS-registrierten Datei dies in der entsprechenden Datei der Arbeitskopie durch `$Revision: 1.1$`[14] ersetzt.

[13] CVS erzeugt im Lock-Ordner Dateien zur Kennzeichnung gesperrter RCS-Dateien.
[14] Die Versionsnummer entspricht dem Versionsstand der Datei in der Arbeitskopie.

Kapitel 4

Schritt für Schritt: Ein erstes Projekt

Da CrossVC nun erfolgreich installiert wurde und Sie Einblick in die allernötigsten theoretischen CVS-Grundlagen hatten, wollen wir hier zur Tat schreiten.

Nach dem ersten Start zeigt sich CrossVC, wie in Abbildung 4.1 zu sehen. Die Oberfläche ist in drei Bereiche aufgeteilt:

1. Der *Arbeitsbereich* auf der linken Seite zeigt alle Projekte als Verzeichnisbaum an.
2. Die *Dateiansicht* rechts oben listet die Dateien des aktuellen Ordners.
3. Das *Ausgabefenster* rechts unten zeigt die Meldungen von CVS.

Genauer wird auf all dies in Kapitel 4.4 eingegangen.

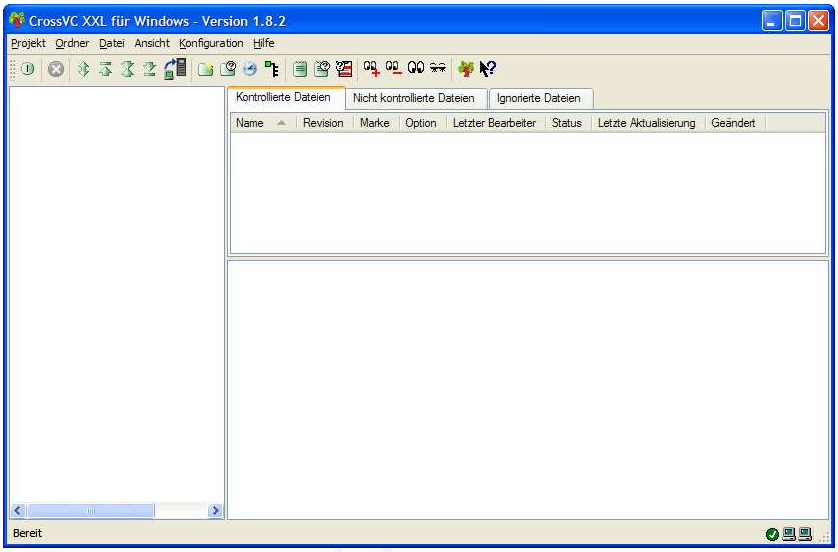


Abbildung 4.1: CrossVC nach dem ersten Start

4.1 Einrichten eines lokalen „Servers“

Bevor man mit CVS arbeiten kann, muss man ein Repository, also die *CVS-Datenbank*[1], erzeugen. Selbst wenn einem bereits ein Repository zur Verfügung steht, so empfiehlt es sich doch, zum Üben ein neues anzulegen. CrossVC unterstützt die Generierung lokaler Repositories; selbige auf entfernten Rechnern anzulegen ist Sache des verantwortlichen Administrators und deshalb nicht in CrossVC implementiert.

Selektieren Sie nun im Projektmenü den Eintrag **Lokales Repository erzeugen ...** und wählen Sie im angezeigten Dialog einen Ordner[2] aus, welcher das Repository beherbergen soll. Beachten Sie, dass Sie über Schreibrechte in diesem Verzeichnis verfügen müssen. Der Name selbst ist frei wählbar.

Nach Bestätigen des Dialogs mit **OK** erzeugt CrossVC das Repository.

[1] Es handelt sich dabei zwar im engeren technischen Sinne um keine Datenbank, da alle Daten in menschenlesbaren Dateien gespeichert werden, aber es erfüllt die Aufgaben einer Datenbank.

[2] Je nach Betriebssystem können Sie in diesem Dialog auch einen neuen Ordner anlegen.

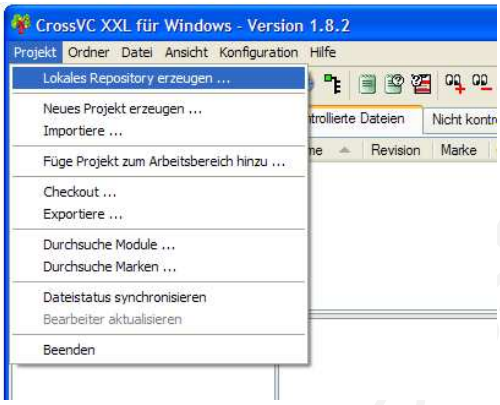


Abbildung 4.2: Erzeugen eines lokalen Repositorys

4.2 Ein Projekt ins Leben rufen

Noch ist unser frisch generiertes Repository leer. Um das zu ändern, müssen wir ein Projekt oder Modul *importieren*. Dies kann ein bereits bestehendes sein, wie die Ordner und Dateien einer Webseite. Im einfachsten Falle aber stellt schon eine leere Datei ein Projekt dar.

Fahren wir mit der einfachen Variante fort: Legen Sie einen neuen Ordner an; Name und Speicherort spielen keine Rolle. Wir benötigen ihn nur für den Import unseres Projekts. Erzeugen Sie nun in diesem Ordner eine Textdatei. Wählen Sie einen geeigneten Namen, denn CVS unterstützt das Umbenennen von Dateien nur rudimentär. . . Da jedoch alle Theorie grau ist, greifen wir auf die Entstehung von Kapitel 1.4 dieses Buches zurück. Entsprechend nennen wir die Textdatei `Kapitel1.txt`.

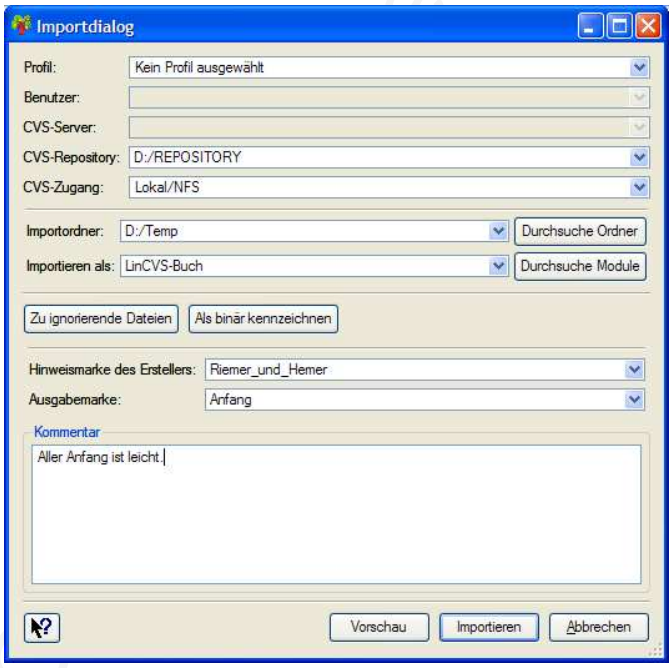
Wählen Sie aus dem Projektmenü den Eintrag **Importiere . . .** aus. CrossVC präsentiert Ihnen daraufhin den Importdialog, wie in Abbildung 4.3 zu sehen.

Ignorieren Sie zunächst die Profilauswahlbox. Da wir mit einem lokalen Repository arbeiten, sind auch Nutzer und Server für uns nicht von Interesse. Im Feld **CVS-Repository** tragen Sie bitte den absoluten Pfad zu dem eben angelegten Repository ein, unter Linux/Unix also etwas wie `/home/riemer/REPOSITORY`, unter Windows etwas wie `D:/Repository`[3]

[3] Wollen Sie CVS unter Windows auf der Kommandozeile benutzen, muss als Trennzeichen zwischen Ordnern ein Backslash (\) statt des / verwendet werden, also `D:\...` CrossVC konvertiert die Trennzeichen bei Bedarf.

und unter Mac OS X möglicherweise `/Users/riemer/REPOSITORY`. Wählen Sie `Lokal/NFS`[4] als **CVS-Zugang** aus. Als **Importordner** geben Sie den vollen Pfad zu dem temporär angelegten Ordner an, der `Kapitel1.txt` enthält. Sie können mittels der nebenstehenden Schaltfläche hierzu auch einen Auswahldialog benutzen. **Importieren als** erwartet den zukünftigen Namen des Projekts. Nennen wir es naheliegenderweise `CrossVC-Buch`. Unter **Hinweismarke** des Erstellers tragen wir `Riemer_und_Hemer` ein, als **Ausgabemarke** `Anfang`. CVS kann keine Marken verarbeiten, die Leerzeichen oder Tabulatoren enthalten. CrossVC unterbindet deshalb die Eingabe dieser Zeichen in den entsprechenden Eingabefeldern. Zu guter Letzt geben wir als **Kommentar** etwas wie `Aller Anfang ist leicht.` ein.

Abbildung 4.3: Import eines Projekts



Klicken Sie jetzt auf die **Importieren**-Schaltfläche. Im Ausgabefenster sollten daraufhin drei Zeilen erscheinen:

```
cmd: cvs -d D:\Repository -r import ...
N CrossVC-Buch/Kapitel1.txt
No conflicts created by this import
```

[4] Eigentlich müsste hier ergänzenderweise Samba stehen, da man ein lokales Repository auch auf einem Sambaserver anlegen kann.

In diesem Falle wurde unser erstes Projekt erfolgreich importiert.[5] Das Projekt enthält eine neue Datei, gekennzeichnet durch das `N` zu Beginn der zweiten Zeile. Es traten keine Konflikte auf, was bei einem Import wie in unserem Beispiel zu erwarten war. In Kapitel 9.3.3 werden wir auf konfliktträchtige Importe zu sprechen kommen.

Das temporäre Verzeichnis kann jetzt gelöscht werden, denn es wird nicht weiter gebraucht.

Anwender von CrossVC XXL können alternativ über das Menü **Projekt | Neues Projekt erzeugen ...** zunächst ein leeres Projekt im Arbeitsbereich generieren. Tragen Sie dazu als **CVS-Repository** den Pfad zum zuvor neu erstellten Repository ein. Als **Name** für das Projekt verwenden Sie analog dazu `CrossVC-Buch`. Als **Ort** wählen Sie einen beliebigen Pfad mit Schreibberechtigung, beispielsweise `D:/CrossVC-Buch`. Bei dieser Vorgehensweise haben Sie den im nächsten Kapitel beschriebenen Schritt übersprungen, benötigen allerdings noch die Textdatei `Kapitel1.txt`. Das Hinzufügen von Dateien ist Gegenstand des Kapitels 7.5.

4.3 Eine Arbeitskopie erzeugen

Uns steht nun ein Repository inklusive Projekt zur Verfügung. Direkt werden wir letzteres normalerweise nicht manipulieren, sondern immer den Weg über eine Arbeitskopie gehen. Um eine solche zu erhalten, wählen Sie aus dem Projektmenü den Eintrag **Checkout ...** Ein Pendant zum Importdialog wird daraufhin angezeigt (Abbildung 4.4).

Füllen Sie die Felder im oberen Bereich des Dialogs genauso aus wie im Falle des Imports. **Checkout nach** erfordert die Eingabe des Ordners, in welchem die Arbeitskopie gespeichert werden soll. In diesem Beispiel ist dies `D:/`. Unser **CVS-Modul** heißt `CrossVC-Buch`, `Checkout als` ist für uns in dieser Phase noch bedeutungslos. Lassen Sie die übrigen Einstellungen unverändert.

Klicken Sie auf **OK**. Das Ausgabefenster sollte jetzt Folgendes anzeigen:

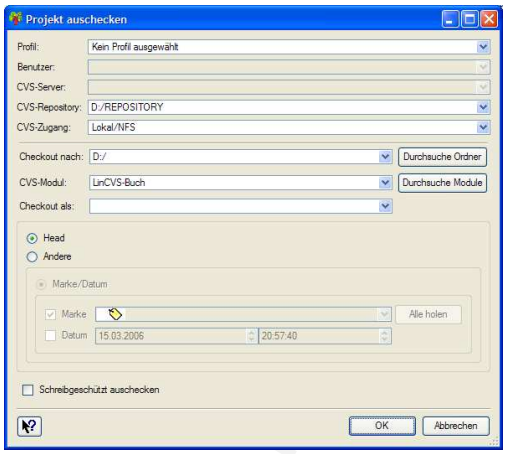
```
cmd: cvs -d D:\Repository  co -P CrossVC-Buch
U CrossVC-Buch/Kapitel1.txt
```

Die Ausgabe zeigt, dass ein Projekt `CrossVC-Buch` ausgecheckt bzw. aktualisiert wurde, welches wiederum eine Datei `Kapitel1.txt` enthält.

Bejahen Sie anschließend die Frage, ob das angeforderte Projekt zum Arbeitsbereich hinzugefügt werden soll.

[5] Wer neugierig ist, kann mit dem Dateimanager seiner Wahl einen Blick ins Repository werfen. Dort gibt es nun zwei Ordner, `CVSROOT` und `CrossVC-Buch`.

Abbildung 4.4: Checkout einer Arbeitskopie



4.4 Darstellung der Arbeitskopie in CrossVC

Vorbei ist es mit der gähnenden Leere, die CrossVC uns bisher bot, denn das erste Projekt erfreut des Betrachters Auge. Was genau gibt es zu sehen? Werfen wir hierzu einen Blick auf Abbildung 4.5.

Abbildung 4.5: Darstellung der Arbeitskopie