

Bundespategericht

Postfach 90 02 53

81502 München

Jan O. Kechel

Karl-Liebkecht-Str. 37

14482 Postdam

Tel.: 0170-8341452

Email: jan@kechel.de

Datum: 25. Februar 2006

Aktenzeichen: 17 W (pat) 318/05

103 19 887.3-53

Zeichen: 103 19 887.3-53

Einsprechender: Jan O. Kechel

Patentinhaber(in): Siemens Aktiengesellschaft

Ich nehme Bezug auf das Schreiben vom 24. Februar 2006, in dem der Patentinhaber Stellung hinsichtlich meines Antrags auf Verfahrenskostenhilfe nimmt.

Teil 1: CVS - Veröffentlichung, Stand der Technik

Der Patentinhaber beantragt die Aufrechterhaltung des oben genannten deutschen Patents im vollen Umfang. Begründet wird dies mit der Behauptung, daß das als Beispiel herangezogene Versionsverwaltungssystem CVS nicht zum Stand der Technik gehörte bzw. daß kein Beleg über den Veröffentlichungszeitpunkt von CVS dem Einspruch beiliegen würde.

Dies möchte ich hiermit nachholen:

CVS wurde am 3. Juli 1986 öffentlich zugänglich gemacht. Sogar das

originale Posting in der Usenet-Gruppe mod.sources ist noch im Internet zu finden:

<http://groups.google.com/group/mod.sources/msg/2ebab72ac0744fb8>

Desweiteren bringt der Patentinhaber vor, daß mein Einspruch nicht ausreichend substantiiert und damit nicht zulässig sei, da aus dem Einspruchsschriftsatz keine abschließenden Folgerungen für das Vorliegen oder Nichtvorliegen eines Widerrufsgrundes ohne eigene Ermittlungen gezogen werden können (BGH Streichgarn, BIPMZ 87, 203).

Ich aber halte Versionsverwaltungssysteme für ein absolut grundlegendes Werkzeug jedes Informatikers. Es ist nicht glaubhaft, daß Siemens Ermittlungen anstellen müßte, um zu wissen was CVS ist, was es macht und wie es funktioniert.

Um zu demonstrieren, wie verbreitet die Kenntnisse über Versionsverwaltungssysteme sind, hier ein paar Beispiele von Vorlesungen zu diesem Thema an verschiedenen Universitäten in Deutschland (fett markierte Daten liegen vor dem Anmeldedatum des angegriffenen Patents):

TU-Berlin SS 05 "UML/Java-Praktikum", Seiten 29-31

<http://tfs.cs.tu-berlin.de/lehre/SS05/UML-Java/swentwicklung-4.pdf>

FU-Berlin WS 03/04 "Partizipation im Internet", CVS ist Teil der VL

<http://www.mi.fu-berlin.de/kvv/?veranstaltung=319>

Universität München WS 02/03 "Programmierpraktikum", CVS ist gleich am zweiten Termin Thema des Praktikums

http://www.dbs.informatik.uni-muenchen.de/Lehre/Programmierpraktikum/Skript/ProgPraktWS02_1.pdf

Universität Oldenburg, **WS 01/02** "Modul Softwareprojekt incl. Proseminar":

www-is.informatik.uni-oldenburg.de/~sauer/lehre/swp_01_02/swp_01_02.html

Diese Vorlesung enthält eine 174 Seitige Anleitung zu CVS 1.10.7 und ist mit einem Copyright aus dem Jahr **1992/1993** versehen:

http://www-is.informatik.uni-oldenburg.de/~sauer/lehre/swp_01_02/cvs.pdf

Universität Karlsruhe, **WS 01/02** "Softwareengineering", enthält Folien über CVS und RCS (siehe Einspruch Seite 4).

[http://www.aifb.uni-karlsruhe.de/Lehrangebot/Winter2001-](http://www.aifb.uni-karlsruhe.de/Lehrangebot/Winter2001-02/SwEng/PPFolien/Folien_SE09.ppt)

[02/SwEng/PPFolien/Folien_SE09.ppt](http://www.aifb.uni-karlsruhe.de/Lehrangebot/Winter2001-02/SwEng/PPFolien/Folien_SE09.ppt)

Universität Bonn, **SS 99** "Softwaretechnologie", enthält in Übungsblatt 3 sogar eine extra Übungsaufgabe zu CVS, insbesondere zu den Befehlen checkout, update und commit

<http://www.informatik.uni-bonn.de/III/lehre/vorlesungen/SWT/SS99/>

Außerdem gibt es sehr viele Bücher über Versionskontrollsysteme:

- CVS Versionskontrolle und Quellcode-Management
ISBN: 3897213699, Erscheinungsjahr: 2004
- CVS kurz und gut O' Reillys Taschenbibliothek
ISBN: 389721265X, Erscheinungsjahr: 2004
- CVS - Windows- und Open Source-Projekte managen
ISBN: 3898427056, Erscheinungsjahr: 2005
- Pragmatisch Programmieren: Versionsverwaltung mit CVS
ISBN: 3446404708, Erscheinungsjahr: 2005
- Pragmatic Version Control Using CVS The Pragmatic Starter Kit 1
ISBN: 0974514004, Erscheinungsjahr: **2003**
- Open Source Development with CVS
ISBN: 1576104907, Erscheinungsjahr: **2000**
- Open Source-Projekte mit CVS

ISBN: 382661416X, Erscheinungsjahr **2003**

Sowie für andere Versionskontrollsysteme:

- Versionskontrolle mit Subversion
ISBN: 3897213907, Erscheinungsjahr: 2006
- Linux Kochbuch Praktischer Rat für Anwender und Systemadministratoren
ISBN: 3897214059, Erscheinungsjahr: 2005
- Maven Das Entwicklerheft
ISBN: 3897214377, Erscheinungsjahr: 2005
- UNIX System Administration
ISBN: 3897213478, Erscheinungsjahr: **2003**
- Applying SCCS und RCS. From Source Control to Project Control
ISBN: 1565921178, Erscheinungsjahr: **1995**
- The RCS Handbook
ISBN: 0471435651, Erscheinungsjahr: **2001**

Teil 2: Mündliche Verhandlung

Der Patentinhaber beantragt eine mündliche Verhandlung, falls seinem Antrag auf Erhaltung des Patents im vollen Umfang nicht vollständig im schriftlichen Verfahren stattgegeben wird.

Ich bin gerne bereit einer mündlichen Verhandlung beizuwohnen. Jedoch nur bzgl. des Einspruchs selber, nicht zur Verhandlung über meine VKH. Die Fahrtkosten nach München übersteigen meinen Anteil an den Kosten des Einspruchs (Einspruch: 200,- EUR, Fahrtkosten mit Zug Berlin-München-Berlin 230,- EUR). Es ist also nicht in meinem wirtschaftlichen Interesse eine mündliche Verhandlung wegen Verfahrenskostenhilfe auf mich zu nehmen.

Mit freundlichen Grüßen

Jan O. Kechel
BSc Software Engineering

Anlagen

1. unterschriebenes Doppel
2. Teile des CVS-Handbuchs der Version 1.11.3 vom **27. Dezember 2002.**

Anlage 3

CVS--Concurrent Versions System v1.11.3

<http://ximbiot.com/cvs/manual/cvs-1.11.3/cvs.html>

Falls die folgenden Auszüge dem Nachweis noch nicht genüge tun, dann bitte ich das vollständige Handbuch sowie den Source-Code zu Rate zu ziehen (<http://cvs.nongnu.org/>).

- Das Kapitel 2.3 (ab Seite 7) beschreibt, wie die Daten auf dem Server gespeichert werden (zu Einspruch gegen Anspruch 1 erste Tabellenzeile)
- Der Absatz 'Entries' (Seite 8) beschreibt die Verwendung von Zeitstempeln (im Einspruch zu Anspruch 1 zweite Tabellenzeile, im Patent als Zeitrang bezeichnet)
- Das Kapitel A8 beschreibt den commit Befehl (ab Seite 12)
- Das Kapitel A16 beschreibt den update Befehl (ab Seite 15)

1.1 What is CVS?

CVS is a version control system. Using it, you can record the history of your source files.

For example, bugs sometimes creep in when software is modified, and you might not detect the bug until a long time after you make the modification. With CVS, you can easily retrieve old versions to see exactly which change caused the bug. This can sometimes be a big help.

You could of course save every version of every file you have ever created. This would however waste an enormous amount of disk space. CVS stores all the versions of a file in a single file in a clever way that only stores the differences between versions.

CVS also helps you if you are part of a group of people working on the same project. It is all too easy to overwrite each others' changes unless you are extremely careful. Some editors, like GNU Emacs, try to make sure that the same file is never modified by two people at the same time.

Unfortunately, if someone is using another editor, that safeguard will not work. CVS solves this problem by insulating the different developers from each other. Every developer works in his own directory, and CVS merges the work when each developer is done.

CVS started out as a bunch of shell scripts written by Dick Grune, posted to the newsgroup comp.sources.unix in the volume 6 release of December, 1986. While no actual code from these shell scripts is present in the current version of CVS much of the CVS conflict resolution algorithms come from them.

In April, 1989, Brian Berliner designed and coded CVS. Jeff Polk later helped Brian with the design of the CVS module and vendor branch support.

You can get CVS in a variety of ways, including free download from the internet. For more information on downloading CVS and other CVS topics, see:

<http://www.cvshome.org/>

<http://www.loria.fr/~molli/cvs-index.html>

There is a mailing list, known as info-cvs, devoted to CVS. To subscribe or unsubscribe write to info-cvs-request@gnu.org. If you prefer a usenet group, the right group is comp.software.config-mgmt which is for CVS discussions (along with other configuration management systems). In the future, it might be possible to create a comp.software.config-mgmt.cvs, but probably only if there is sufficient CVS traffic on comp.software.config-mgmt.

You can also subscribe to the bug-cvs mailing list, described in more detail in [H. Dealing with bugs in CVS or this manual](#). To subscribe send mail to bug-cvs-request@gnu.org.

2.3 How data is stored in the working directory

While we are discussing CVS internals which may become visible from time to time, we might as well talk about what CVS puts in the `CVS' directories in the working directories. As with the repository, CVS handles this information and one can usually access it via CVS commands. But in some cases it may be useful to look at it, and other programs, such as the jCVS graphical user interface or the VC package for emacs, may need to look at it. Such programs should follow the recommendations in this section if they hope to be able to work with other programs which use those files, including future versions of the programs just mentioned and the command-line CVS client.

The ``CVS'` directory contains several files. Programs which are reading this directory should silently ignore files which are in the directory but which are not documented here, to allow for future expansion.

The files are stored according to the text file convention for the system in question. This means that working directories are not portable between systems with differing conventions for storing text files. This is intentional, on the theory that the files being managed by CVS probably will not be portable between such systems either.

``Root'`

This file contains the current CVS root, as described in [2.1 Telling CVS where your repository is](#).

``Repository'`

This file contains the directory within the repository which the current directory corresponds with. It can be either an absolute pathname or a relative pathname; CVS has had the ability to read either format since at least version 1.3 or so. The relative pathname is relative to the root, and is the more sensible approach, but the absolute pathname is quite common and implementations should accept either. For example, after the command

```
cvs -d :local:/usr/local/cvsroot checkout yoyodyne/tc
```

``Root'` will contain

```
:local:/usr/local/cvsroot
```

and ``Repository'` will contain either

```
/usr/local/cvsroot/yoyodyne/tc
```

or

```
yoyodyne/tc
```

If the particular working directory does not correspond to a directory in the repository, then ``Repository'` should contain

```
`CVSROOT/Emptydir'.
```

``Entries'`

This file lists the files and directories in the working directory. The first character of each line indicates what sort of line it is. If the character is unrecognized, programs reading the file should silently skip that line, to allow for future expansion.

If the first character is ``/'`, then the format is:

/name/revision/timestamp[+conflict]/options/tagdate
where '[' and ']' are not part of the entry, but instead indicate that the '+' and conflict marker are optional. *name* is the name of the file within the directory. *revision* is the revision that the file in the working derives from, or '0' for an added file, or '-' followed by a revision for a removed file. *timestamp* is the timestamp of the file at the time that CVS created it; if the timestamp differs with the actual modification time of the file it means the file has been modified. It is stored in the format used by the ISO C `asctime()` function (for example, 'Sun Apr 7 01:29:26 1996'). One may write a string which is not in that format, for example, 'Result of merge', to indicate that the file should always be considered to be modified. This is not a special case; to see whether a file is modified a program should take the timestamp of the file and simply do a string compare with *timestamp*. If there was a conflict, *conflict* can be set to the modification time of the file after the file has been written with conflict markers (see section [10.3 Conflicts example](#)). Thus if *conflict* is subsequently the same as the actual modification time of the file it means that the user has obviously not resolved the conflict. *options* contains sticky options (for example '-kb' for a binary file). *tagdate* contains 'T' followed by a tag name, or 'D' for a date, followed by a sticky tag or date. Note that if *timestamp* contains a pair of timestamps separated by a space, rather than a single timestamp, you are dealing with a version of CVS earlier than CVS 1.5 (not documented here).

The timezone on the timestamp in CVS/Entries (local or universal) should be the same as the operating system stores for the timestamp of the file itself. For example, on Unix the file's timestamp is in universal time (UT), so the timestamp in CVS/Entries should be too. On VMS, the file's timestamp is in local time, so CVS on VMS should use local time. This rule is so that files do not appear to be modified merely because the timezone changed (for example, to or from summer time).

If the first character of a line in 'Entries' is 'D', then it indicates a subdirectory. 'D' on a line all by itself indicates that the program which wrote the 'Entries' file does record subdirectories (therefore, if there is such a line and no other lines beginning with 'D', one knows there are no subdirectories). Otherwise, the line looks like:

D/name/filler1/filler2/filler3/filler4

where *name* is the name of the subdirectory, and all the *filler* fields should be silently ignored, for future expansion. Programs which

modify Entries files should preserve these fields.

The lines in the ``Entries'` file can be in any order.

``Entries.Log'`

This file does not record any information beyond that in ``Entries'`, but it does provide a way to update the information without having to rewrite the entire ``Entries'` file, including the ability to preserve the information even if the program writing ``Entries'` and ``Entries.Log'` abruptly aborts. Programs which are reading the ``Entries'` file should also check for ``Entries.Log'`. If the latter exists, they should read ``Entries'` and then apply the changes mentioned in ``Entries.Log'`. After applying the changes, the recommended practice is to rewrite ``Entries'` and then delete ``Entries.Log'`. The format of a line in ``Entries.Log'` is a single character command followed by a space followed by a line in the format specified for a line in ``Entries'`. The single character command is ``A'` to indicate that the entry is being added, ``R'` to indicate that the entry is being removed, or any other character to indicate that the entire line in ``Entries.Log'` should be silently ignored (for future expansion). If the second character of the line in ``Entries.Log'` is not a space, then it was written by an older version of CVS (not documented here).

Programs which are writing rather than reading can safely ignore ``Entries.Log'` if they so choose.

``Entries.Backup'`

This is a temporary file. Recommended usage is to write a new entries file to ``Entries.Backup'`, and then to rename it (atomically, where possible) to ``Entries'`.

``Entries.Static'`

The only relevant thing about this file is whether it exists or not. If it exists, then it means that only part of a directory was gotten and CVS will not create additional files in that directory. To clear it, use the update command with the ``-d'` option, which will get the additional files and remove ``Entries.Static'`.

``Tag'`

This file contains per-directory sticky tags or dates. The first character is ``T'` for a branch tag, ``N'` for a non-branch tag, or ``D'` for a date, or another character to mean the file should be silently ignored, for future expansion. This character is followed by the tag or date. Note that per-directory sticky tags or dates are used for

things like applying to files which are newly added; they might not be the same as the sticky tags or dates on individual files. For general information on sticky tags and dates, see [4.9 Sticky tags](#).

`Checkin.prog'

`Update.prog'

These files store the programs specified by the ``-i'` and ``-u'` options in the modules file, respectively.

`Notify'

This file stores notifications (for example, for edit or unedit) which have not yet been sent to the server. Its format is not yet documented here.

`Notify.tmp'

This file is to ``Notify'` as ``Entries.Backup'` is to ``Entries'`. That is, to write ``Notify'`, first write the new contents to ``Notify.tmp'` and then (atomically where possible), rename it to ``Notify'`.

`Base'

If watches are in use, then an edit command stores the original copy of the file in the ``Base'` directory. This allows the unedit command to operate even if it is unable to communicate with the server.

`Baserev'

The file lists the revision for each of the files in the ``Base'` directory. The format is:

*Bname/rev/expansi
on*

where *expansion* should be ignored, to allow for future expansion.

`Baserev.tmp'

This file is to ``Baserev'` as ``Entries.Backup'` is to ``Entries'`. That is, to write ``Baserev'`, first write the new contents to ``Baserev.tmp'` and then (atomically where possible), rename it to ``Baserev'`.

`Template'

This file contains the template specified by the ``rcsinfo'` file (see section [C.4 Rcsinfo](#)). It is only used by the client; the non-client/server CVS consults ``rcsinfo'` directly.

A.8 commit--Check files into the repository

- Synopsis: commit [-lnRf] [-m 'log_message' | -F file] [-r revision] [files...]
- Requires: working directory, repository.
- Changes: repository.
- Synonym: ci

Use commit when you want to incorporate changes from your working source files into the source repository.

If you don't specify particular files to commit, all of the files in your working current directory are examined. commit is careful to change in the repository only those files that you have really changed. By default (or if you explicitly specify the `-R` option), files in subdirectories are also examined and committed if they have changed; you can use the `-l` option to limit commit to the current directory only.

commit verifies that the selected files are up to date with the current revisions in the source repository; it will notify you, and exit without committing, if any of the specified files must be made current first with update (see section [A.16 update--Bring work tree in sync with repository](#)). commit does not call the update command for you, but rather leaves that for you to do when the time is right.

When all is well, an editor is invoked to allow you to enter a log message that will be written to one or more logging programs (see section [C.1 The modules file](#), and see section [C.3.5 Loginfo](#)) and placed in the RCS file inside the repository. This log message can be retrieved with the log command; see [A.13 log--Print out log information for files](#). You can specify the log message on the command line with the `-m message` option, and thus avoid the editor invocation, or use the `-F file` option to specify that the argument file contains the log message.

A.8.1 commit options

These standard options are supported by commit (see section [A.5 Common command options](#), for a complete description of them):

- l Local; run only in current working directory.
- n Do not run any module program.
- R Commit directories recursively. This is on by default.

-r *revision*

Commit to *revision*. *revision* must be either a branch, or a revision on the main trunk that is higher than any existing revision number (see section [4.3 Assigning revisions](#)). You cannot commit to a specific revision on a branch.

commit also supports these options:

-F *file*

Read the log message from *file*, instead of invoking an editor.

-f

Note that this is not the standard behavior of the ``-f'` option as defined in [A.5 Common command options](#).

Force CVS to commit a new revision even if you haven't made any changes to the file. If the current revision of *file* is 1.7, then the following two commands are equivalent:

```
$ cvs commit -f file
$ cvs commit -r 1.8 file
```

The ``-f'` option disables recursion (i.e., it implies ``-l'`). To force CVS to commit a new revision for all files in all subdirectories, you must use ``-f -R'`.

-m *message*

Use *message* as the log message, instead of invoking an editor.

A.8.2 commit examples

A.8.2.1 Committing to a branch

You can commit to a branch revision (one that has an even number of dots) with the ``-r'` option. To create a branch revision, use the ``-b'` option of the `rtag` or `tag` commands (see section [5. Branching and merging](#)). Then, either `checkout` or `update` can be used to base your sources on the newly created branch. From that point on, all commit changes made within these working sources will be automatically added to a branch revision, thereby not disturbing main-line development in any way. For example, if you had to create a patch to the 1.2 version of the product, even though the 2.0 version is already under development, you might do:

```
$ cvs rtag -b -r FCS1_2 FCS1_2_Patch product_module
```

```
$ cvs checkout -r FCS1_2_Patch product_module
```

```
$ cd product_module
```

```
[[ hack away ]]
```

```
$ cvs commit
```

This works automatically since the ``-r'` option is sticky.

A.16 update--Bring work tree in sync with repository

- update [-ACdfIPpR] [-l name] [-j rev [-j rev]] [-k kflag] [-r tag|-D date] [-W spec] files...
- Requires: repository, working directory.
- Changes: working directory.

After you've run checkout to create your private copy of source from the common repository, other developers will continue changing the central source. From time to time, when it is convenient in your development process, you can use the update command from within your working directory to reconcile your work with any revisions applied to the source repository since your last checkout or update.

A.16.1 update options

These standard options are available with update (see section [A.5 Common command options](#), for a complete description of them):

-D *date*

Use the most recent revision no later than *date*. This option is sticky, and implies ``-P'`. See [4.9 Sticky tags](#), for more information on sticky tags/dates.

-f

Only useful with the ``-D date'` or ``-r tag'` flags. If no matching revision is found, retrieve the most recent revision (instead of ignoring the file).

-k *kflag*

Process keywords according to *kflag*. See [12. Keyword substitution](#). This option is sticky; future updates of this file in this working directory will use the same *kflag*. The status command can be viewed to see the sticky options. See [B. Quick reference to CVS commands](#), for more information on the status command.

-l

Local; run only in current working directory. See section [6. Recursive behavior](#).

-P

Prune empty directories. See [7.5 Moving and renaming directories](#).

-p

Pipe files to the standard output.

-R

Update directories recursively (default). See section [6. Recursive behavior](#).

-r *rev*

Retrieve revision/tag *rev*. This option is sticky, and implies ``-P'`. See [4.9 Sticky tags](#), for more information on sticky tags/dates.

These special options are also available with update.

`-A`

Reset any sticky tags, dates, or ``-k'` options. See [4.9 Sticky tags](#), for more information on sticky tags/dates.

`-C`

Overwrite locally modified files with clean copies from the repository (the modified file is saved in ``.#file.revision'`, however).

`-d`

Create any directories that exist in the repository if they're missing from the working directory. Normally, update acts only on directories and files that were already enrolled in your working directory.

This is useful for updating directories that were created in the repository since the initial checkout; but it has an unfortunate side effect. If you deliberately avoided certain directories in the repository when you created your working directory (either through use of a module name or by listing explicitly the files and directories you wanted on the command line), then updating with ``-d'` will create those directories, which may not be what you want.

`-I name`

Ignore files whose names match *name* (in your working directory) during the update. You can specify ``-I'` more than once on the command line to specify several files to ignore. Use ``-I !'` to avoid ignoring any files at all. See section [C.5 Ignoring files via cvsignore](#), for other ways to make CVS ignore some files.

`-Wspec`

Specify file names that should be filtered during update. You can use this option repeatedly.

spec can be a file name pattern of the same type that you can specify in the ``.cvswrappers'` file. See section [C.2 The cvswrappers file](#).

`-jrevision`

With two ``-j'` options, merge changes from the revision specified with the first ``-j'` option to the revision specified with the second ``j'` option, into the working directory.

With one ``-j'` option, merge changes from the ancestor revision to

the revision specified with the ``-j'` option, into the working directory. The ancestor revision is the common ancestor of the revision which the working directory is based on, and the revision specified in the ``-j'` option.

Note that using a single ``-j tagname'` option rather than ``-j branchname'` to merge changes from a branch will often not remove files which were removed on the branch. See section [5.9 Merging can add or remove files](#), for more.

In addition, each ``-j'` option can contain an optional date specification which, when used with branches, can limit the chosen revision to one within a specific date. An optional date is specified by adding a colon (`:`) to the tag: ``-jSymbolic_Tag:Date_Specifier'`.

See section [5. Branching and merging](#).

A.16.2 update output

update and checkout keep you informed of their progress by printing a line for each file, preceded by one character indicating the status of the file:

U *file*

The file was brought up to date with respect to the repository. This is done for any file that exists in the repository but not in your source, and for files that you haven't changed but are not the most recent versions available in the repository.

P *file*

Like ``U'`, but the CVS server sends a patch instead of an entire file. This accomplishes the same thing as ``U'` using less bandwidth.

A *file*

The file has been added to your private copy of the sources, and will be added to the source repository when you run commit on the file. This is a reminder to you that the file needs to be committed.

R *file*

The file has been removed from your private copy of the sources, and will be removed from the source repository when you run commit on the file. This is a reminder to you that the file needs to be committed.

M *file*

The file is modified in your working directory.

`M' can indicate one of two states for a file you're working on: either there were no modifications to the same file in the repository, so that your file remains as you last saw it; or there were modifications in the repository as well as in your copy, but they were merged successfully, without conflict, in your working directory.

CVS will print some messages if it merges your work, and a backup copy of your working file (as it looked before you ran update) will be made. The exact name of that file is printed while update runs.

C file

A conflict was detected while trying to merge your changes to *file* with changes from the source repository. *file* (the copy in your working directory) is now the result of attempting to merge the two revisions; an unmodified copy of your file is also in your working directory, with the name ``.#file.revision'` where *revision* is the revision that your modified file started from. Resolve the conflict as described in [10.3 Conflicts example](#). (Note that some systems automatically purge files that begin with ``.#'` if they have not been accessed for a few days. If you intend to keep a copy of your original file, it is a very good idea to rename it.) Under VMS, the file name starts with ``_#'` rather than ``.#'`.

? file

file is in your working directory, but does not correspond to anything in the source repository, and is not in the list of files for CVS to ignore (see the description of the ``-I'` option, and see section [C.5 Ignoring files via cvsignore](#)).