Cervisia Manual

Bernd Gehrmann
Carlos Woelz

Contents

1 Introduction **7**

2 Getting Started **8**

2.1 Accessing The Repository 8

2.2 Importing a Module Into the Repository 10

2.3 Checkout a Module From the Repository 12

2.4 The Main Screen, Viewing File Status and Updating 14

3 Working With Files **17**

3.1 Adding Files 18

3.2 Removing Files 18

3.3 Adding and Removing Folders 19

3.4 Committing Files 19

3.5 Resolving Conflicts 20

4 Obtaining Information About Files and Creating Patches **23**

4.1 Watching Differences Between Revisions 23

4.2 Creating Patches 24

4.3 Watching an Annotated View of a File 26

4.4 Browsing CVS Logs 27

4.5 Browsing the History 28

5 Advanced Usage **31**

5.1 Updating to Tag, Branch or Date 31

5.2 Tagging and Branching 33

5.3 Using Watches 33

5.4 Locking 35

6 Customizing Cervisia **36**

6.1 General 36

6.2 Diff Viewer 36

6.3 Status 37

6.4 Advanced 37

6.5 Appearance 37

7 Appendix **38**

7.1 Ignored Files 38

7.2 Further Information and Support 38

7.3 Command Reference 39

7.3.1 The File Menu 39

7.3.2 The View Menu 40

7.3.3 The Advanced Menu 41

7.3.4 The Repository Menu 41

7.3.5 The Settings and Help Menu 42

8 Credits And License **43**

List of Figures

2.1 A screenshot of Cervisia's Configure Access to Repositories dialog 9
2.2 A screenshot of Cervisia's import dialog 11
2.3 A screenshot of Cervisia's checkout dialog 13
2.4 A screenshot of Cervisia's main view 15

3.1 A screenshot of Cervisia's context menu 17
3.2 A screenshot of Cervisia's commit dialog 19
3.3 A screenshot of Cervisia's resolve dialog 21

4.1 A screenshot of Cervisia's diff dialog 24
4.2 A screenshot of Cervisia's patch dialog 25
4.3 A screenshot of Cervisia's annotate dialog 26
4.4 A screenshot of Cervisia's browse logs dialog 27
4.5 A screenshot of Cervisia's history dialog 29

5.1 A screenshot of Cervisia's update to tag dialog 32

Abstract

Cervisia provides a graphical view of CVS.

Chapter 1

Introduction

Cervisia is a user friendly version control system front-end. The aim is to support CVS in a unified interface, featuring conflict resolution, difference and history viewers, status for the working copy files, and support for most version control functions. You can get Cervisia by building the Cervisia program or installing the Cervisia package provided by your distribution.

A version control system is a tool to record, manage, and distribute different versions of files. CVS is a version control system. It allows you to share your modifications easily, as each of the contributors can work on their local copy at the same time, without fear of overwriting each others' modifications. It allows the recovery of past versions (useful for tracking bugs), the creation of branches (for experimental development or for releases of code) and more.

The main *repository* usually holds a collaborative project (commercial or not), but you can take advantage of the nice revision control features offered by CVS even for a project developed exclusively by you. It is easy to set up a local repository, and you will gain the ability to track changes that caused bugs, revert changes, avoid accidental loss of information, etc.

The repository holds the project files, and every contributor keeps their own local copy, named *working copy* or *sandbox*; one can then add their modifications to the main repository (a process called 'committing') and/or update their own copy to reflect recent changes made by other contributors.

Chapter 2

Getting Started

2.1 Accessing The Repository

In this section, we show how to use the basic version control system functionality using Cervisia to checkout modules from the repository and work with them. To do that, you must have access to the repository as a client, meaning that someone (probably the administrator of the CVS repository) gave you an account on the server machine. Alternatively, you can easily create a local repository for your own project.

> TIP
> If you plan to develop a complex project, it is a good idea to use the CVS features, even if you are the only developer. You can make all changes in the working copy, and use Cervisia (or any other CVS tool) to update and commit. This way, you will gain the ability to track changes that caused bugs, revert changes, avoid accidental loss of information, etc. Using Cervisia, it is simple to create a local repository.
>
> 1. Open the **Create New Repository (cvs init)** dialog by choosing **Repository** → **Create...**.
> 2. Press the **...** button to select the folder where you want to create the repository, or enter its location in the text box. For instance, if you want to place the repository in the `/home/user` folder, and to name it `cvsroot`, you should type `/home/user/cvsroot` in the text box, or select the `/home/user` folder using the file picker, and add `cvsroot`.
> 3. Confirm by pressing the **OK** button. Cervisia will create and initialize the new repository folder.
> 4. Now you can import your current work to the repository, or simply create a folder in the repository to start a new module from scratch.

Cervisia offers an integrated front-end to manage all your repository locations, the **Configure Access to Repositories** dialog. To display it, select the **Repository** → **Repositories...** menu item.

There are several methods to access a CVS repository. It may be reached via password authentication (:pserver:), secure shell (using :ext:), local repository (:local:), etc. The format for the repository location is (optional items appear between square brackets):

`[:method:][[user][:password]@]hostname[:[port]]/path/to/repository`

Not all these items (user, password, hostname, port) are always necessary to access the repository. The required information depends on the access method used, which can be categorized as follows:

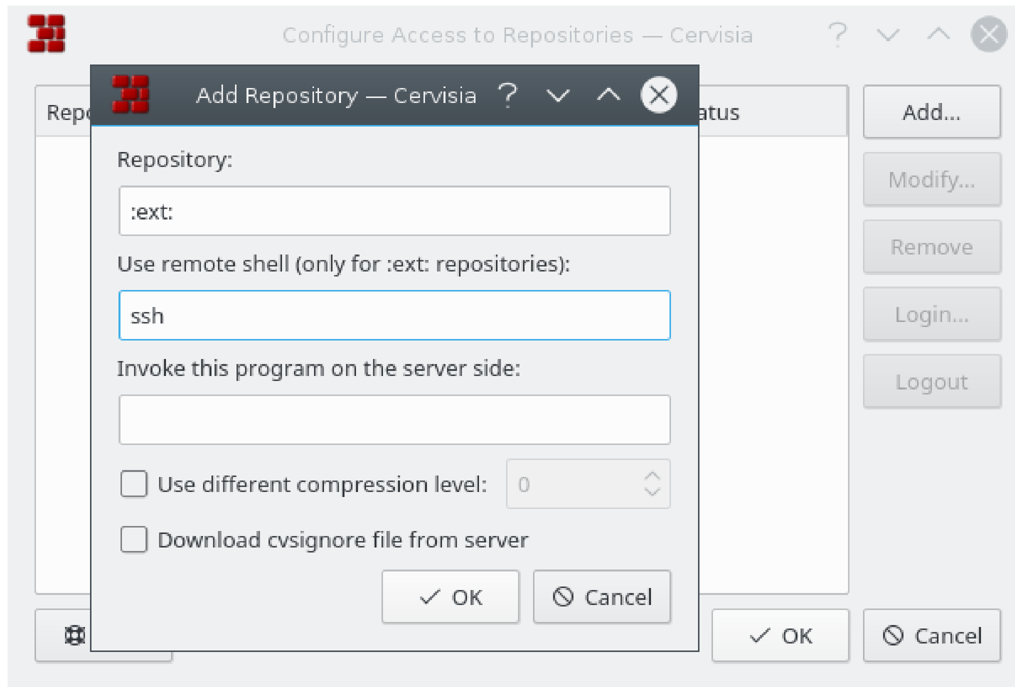

Figure 2.1: A screenshot of Cervisia's Configure Access to Repositories dialog

Local

The local access method is the default method used by CVS. Therefore, it is optional to add the :local: method to the repository location: you can enter simply the path to the folder which stores the CVS repository, and is accessible from your computer, like `/path/to/repository` or to give a real life example, `/home/cvs`.

It may physically be on a disk which is mounted via NFS, but this is an irrelevant detail. If you created a local repository, the location will be simple the path to it.

rsh

The repository location is something like `:ext:username@host.url.org:/path/to/repository`.

This method requires that you have a user account on the server machine (in this example, `host.url.org`) and use a remote shell for communication. By default, CVS uses **rsh** for this purpose; however, **rsh** has long considered to be insecure, and is widely replaced by **ssh**.

If you wish to use **ssh**, you must set the environment variable `$CVS_RSH` to **ssh** when using the **cvs** client. Cervisia supports this easily.

Note that Cervisia cannot answer possible password requests from the server machine. You must make sure that a remote login works without requiring you to enter the password. With plain vanilla **rsh**, this can be achieved for example by creating a `.rhosts` file in your home folder with a list of trusted hosts (see the **rsh** manpage).

With **ssh**, it can be achieved by copying your public key located in the file `identity.pub`, located in the `$HOME /.ssh/` folder to the server. In this case, the key must not be encrypted with a passphrase, see the **ssh** manpage.

pserver

The repository location looks like `:pserver:username@host.url.org:/path/to/repository`

This method accesses the server via a special protocol with a relatively weak authentication (`pserver` stands for password authentication). Before you can use such a server, you

need a username and password given by the CVS server administrator, and you have to login. Note that your CVS password authentication username does not necessarily match the system's username. Before accessing the CVS server, you will need to login.

Open-source projects typically offer Anonymous CVS access to their sources. This means you can easily grab the latest sources, modify, and create patches (differences) against the repository without asking for a CVS account. As a general rule, Anonymous CVS uses password authentication (:pserver:), and is a read-only repository, not allowing you to upload your changes directly.

Knowing the access method and location to the repository, you can add it to Cervisia's repositories list:

1. Open the **Configure Access to Repositories** dialog by choosing the **Repository** → **Repositories...** menu item.
2. Press the **Add...** button to open the **Add Repository** dialog.
3. Enter the repository location in the **Repository:** text box. Cervisia will automatically disable the areas of the dialog that are not relevant to the access method you entered.
4. If you are using the ext method to access the repository, enter the remote shell you wish to use (e.g. **ssh**) in the **Use remote shell (only for :ext: repositories):** text box.
5. Press **OK**. You will see the repository you just entered on the repositories list.
6. If the access method to the repository you just entered is password authentication (pserver), you will need to login before connecting the server. Click the repository on the list to select it, and press the **Login...** button. Enter your password in the upcoming dialog.

   If you successfully enter your password, the **Status** column entry of the pserver repository will change from **Not logged in** to **Logged in**.
7. Press **OK** to apply your modifications, or add another location to the list. Cervisia will store as many locations as you like.

2.2 Importing a Module Into the Repository

In this section, we discuss how you can put a new project into the CVS repository. If you just want to work with an existing project which is already in a repository, you may skip this section.

There are two ways to put a project into the CVS:

- Import the files and folders to a new *module*, using Cervisia's import dialog. Modules are the top folders in the CVS repository folder tree, and are used to separate and organize the different software projects inside the repository.
- Create an empty module and add the new files and folders manually. You will have more control, but it will probably take a little more time.

> IMPORTANT
> Keep in mind that CVS was initially designed to handle text files. Many features, like revision merging, creating differences in a readable form, etc. are only performed to text files. This does not mean you cannot use CVS to keep binary files, it just means you have to *explicitly tell CVS if it is a text or binary file*. If you declare the wrong file type, you will experience problems with the CVS functionality for these files, and they may get corrupted.

Importing a project (as a new module) has some advantages: you will import all files and folders recursively, and the module will automatically be created for you. This makes importing large existing projects to the repository easier. However, there are some disadvantages: you cannot use Cervisia's import dialog to add files to existing modules, and you can either import the files as text or binary files. You can work around this limitation by creating a folder with files of only one of the types, or by informing the patterns of the files that should be ignored during the import process.

For instance, suppose your project contains text files and some PNG images (binary files) only. You can tell CVS to ignore all files with the pattern `*.png` while importing the other files as text, or you can move the images to a separate folder, and then import the remaining files (as text files). Either way, you will have to checkout the newly imported module to a new working copy, copy the missing files and folders to it, add and commit them to the repository to complete the import process.

As an alternative, you can add the files and folders manually, creating an empty module for them. To add an empty module to a repository, just create a new folder in the CVS repository root folder. The name of this new folder will be the name of the module. Checkout the new empty module. Then copy the files and folders to the working copy, add and commit to upload them to the CVS repository.

Figure 2.2: A screenshot of Cervisia's import dialog

In Figure 2.2 you can see the dialog which helps you to *import* a project as a module. To access Cervisia's import dialog, choose the **Repository** → **Import...** menu item.

Repository:

Enter or select from the drop down box the name of the CVS repository, also known as `$CVSROOT`. You must have write access to it, and the repository must be properly initialized. If the repository does not yet exist, you can create one choosing the **Repository** → **Create...** menu item.
The drop down box shows a list of the repositories you previously entered using the **Configure Access to Repositories** dialog box. If the repository is remote, make sure that authentication works. See Section 2.1 for more information.

Module:

The name of the module under which the project will be stored. After the import, the project can be checked out under this name. See Section 2.3 for more information. This is also the name of the corresponding folder in the repository.

Working Folder:

The toplevel folder of the project you want to import. The import starts from this folder and goes down recursively.

Vendor tag:

The vendor tag is historically used for tracking third-party sources. Just use your user name if you have no better idea. It does not matter much what you enter here.

Release tag:

This tag is also historically used for importing different versions of third-party software. If you are not doing this, use the word `start` or a string `FOO_1_0` where `FOO` is the name of your project and `1.0` is the version number of the imported release.

Ignore files:

If you fill out this field, an additional `-I file names` option is given go the **cvs import** command. This entry is interpreted as a whitespace-separated list of file name patterns which are ignored. In general, a cleaner and less error-prone way to control which files go into the repository is to create a folder with only the files which you want to import and start from that. Nevertheless, this entry may be useful if the project contains files which are by default ignored by CVS, e.g. files with the name `core`. In such a case, simply enter the character `!` in this field: this overrules CVS's scheme of ignored files, see Section 7.1.

Comment:

Use this field to record the comments you might have about the origin, use, development, etc. of the files you are importing.

Import as binaries

If you check this box, all files are imported in binary mode, i.e. an argument `-kb` is given to **cvs import**.

Use file's modification as time of import

If you check this box, the time of import will be the file's modification time instead of the import time.

After you have filled out this form and confirmed by pressing **OK**, the following CVS command is used:

```
cvs -d <:\coref{1}{co-repository}:>repository import -m "<:\coref{2}{co- ↩
    comment}:>" <:\coref{3}{co-module}:>module <:\coref{4}{co-vendortag}:> ↩
    vendortag <:\coref{5}{co-releasetag}:>releasetag
```

2.3 Checkout a Module From the Repository

Now that you successfully defined your repository location, and imported the initial files to the repository, it is time to retrieve the module from the CVS repository, creating your working copy.

You should also know the name of the *branch* or *tag* you want to use.

Branches of a module are parallel versions of this module. A good real-life example of the use of this feature is the release of a software project. After a major release, there are bugs in the code that should be fixed, but people want to add new features to the application too. It is very hard

to do both at the same time because new features usually introduce new bugs, making it hard to track down the old ones. To solve this dilemma, CVS lets you create a parallel version, that we will call the 'stable release branch', where you can only add bugfixes, leaving the main branch (HEAD) open for adding new features.

Tags are used to mark a version of a project. CVS stamps one version of each file with the tag, so when you checkout or update to a specific tag, you will get always the same file versions. Therefore, in opposition to branches, tags are not dynamic: you cannot develop a tag. Tags are useful to mark releases, big changes in the code, etc. Using tags, you can easily return the project to a previous state, to reproduce and track bugs, generate the release code again, etc.

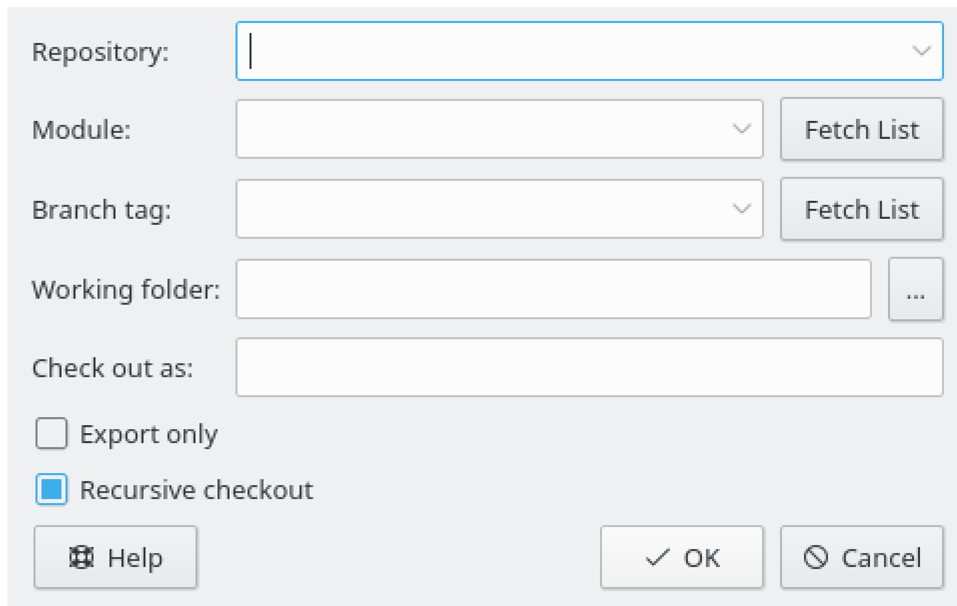

Figure 2.3: A screenshot of Cervisia's checkout dialog

Repository:

The name of the CVS repository, also known as `$CVSROOT`. The drop down box shows a list of the repositories you previously entered using the **Configure Access to Repositories** dialog box. If the repository is remote, make sure that authentication works. See Section 2.1 for more information.

Module:

The name of the module to be checked out. If you are working with an existing repository, you can probably get this name from the system administrator; or, if it is an open-source repository, you can get the module names from the project web pages. If you want to create a new module from scratch using a local repository, just create a new folder in the local repository root folder. The name of the folder will be the same as the name of the empty module.

Alternatively, if the repository has a `$CVSROOT /modules` file, you can retrieve a list of available modules by pressing the **Fetch List** button.

Note that it is possible to checkout any existing subfolder of the module, without retrieving the rest of the module. Just enter the path to the subfolder as well.

Branch tag:

The name of the branch or tag you want to check out. If you leave this field empty, Cervisia will retrieve the main (HEAD) branch.

Working folder:

The folder under which the module should be checked out. Note that the working copy toplevel folder is named after the module you are retrieving, unless you give it an alternative name in the **Check out as:** field.

Check out as:

This results in the working copy files being checked out to an alternative folder under the working folder rather than a folder named after the module.

Export only

If you check this box, the files will be exported rather than checked out. Exporting obtains a copy of the source for the module without the CVS administrative folders. For example, export may be used to prepare the source code for a release.

Recursive checkout

Checkout all files and folders recursively.

2.4 The Main Screen, Viewing File Status and Updating

When you start Cervisia, and open a working copy by choosing **File** → **Open Sandbox...**, you can see two main areas in Cervisia's main window: the top one is a hierarchical (tree) view of the current working copy; the bottom area is used to display the CVS commands Cervisia issues to perform its tasks, as well as the output generated by these commands.

By default, Cervisia does not display the files contained by the sub-folders, so you will have to click the folders you want to see. To display all files of the working copy, select **View** → **Unfold File Tree**. To close back all folders from the working copy, choose **View** → **Fold File Tree**.

According to the settings in your `.cvsignore` files, the files you usually do not want to include into the repository - such as object files - are not shown in the tree view. For each file, you see its corresponding status. In the default setting, after opening the sandbox, this is "Unknown" because Cervisia delays the fetching of information until you select the files and folders whose status you want to update or view and choose **File** → **Update** or **File** → **Status**. With this approach, you have a minimal amount of functionality available even if you do not have a permanent connection to the CVS server.

The commands in the File menu usually act only on the files which you have marked. You may also mark folders. Now choose **File** → **Status** or press **F5**. Cervisia issues a

```
cvs update -n file names
```

command to get status information for the marked files. Note that Cervisia goes recursively into subfolders only if you have the according option in the **Settings** menu set. According to the respective file's status, you now see an entry in the **Status** column:

Locally Modified

This means you have modified the file compared to the version in the repository.

Locally Added

This means the file does not exist in the repository, but in your working copy and you have scheduled it for addition. The actual insertion into the repository only happens after a commit.

Locally Removed

This means you have scheduled the file for removal, but it still exists in the repository. The actual removal happens only after a commit.

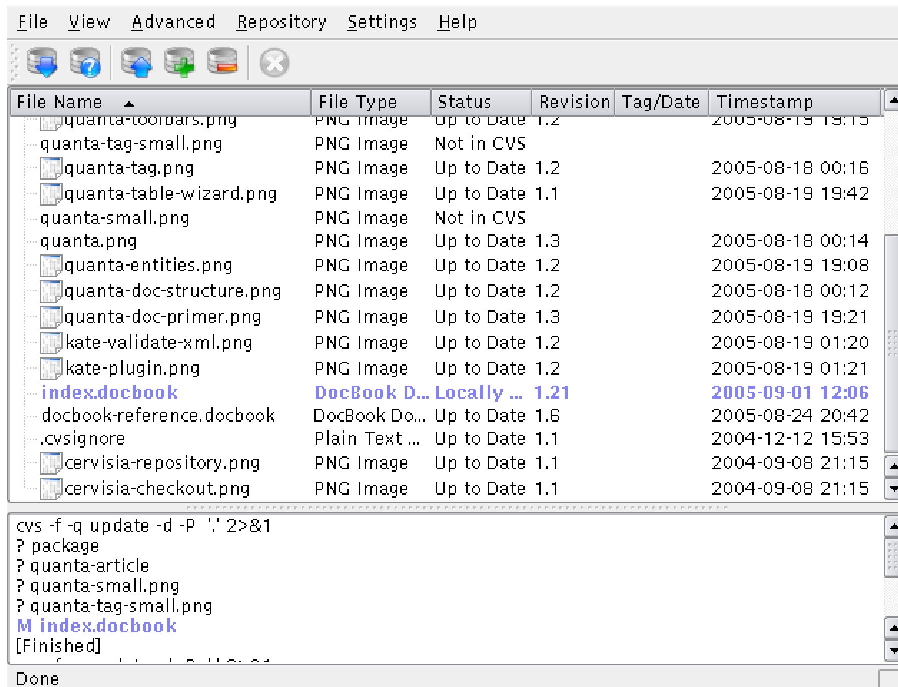

Figure 2.4: A screenshot of Cervisia's main view

Needs Update

This is shown if a newer version of the file exists in the repository, e.g. because someone committed a modification. Normally, you want to update this file so you have an up-to-date version in your folder.

Needs Patch

This is essentially the same as before; the difference is that in case of an update, the CVS server transfers only a patch instead of the whole file to you.

Needs Merge

Indicates that a merge of the revision of this file in your working copy with the version in the repository is necessary. This typically happens if you have made modifications to the file while someone else has committed his modifications. If you choose to update, the modifications in the repository are merged into your file. In case of a conflict (i.e. if someone else has changed some of the same lines like you) the new status is then ″Conflict″.

Up to Date

Indicates that the file is identical with the version in the repository.

Conflict

This is shown if this file still has conflict markers in it. Maybe you have previously updated the file and not resolved the conflicts.

Not In CVS

Indicates that the file is not registered in the CVS repository. If you want it to available for others, you should add it to the repository. If not, you may consider adding it to your `.cvsignore` file.

Now that you have got an overview of the current status of the CVS, you may want to do an update. Mark some files (or the root of the folder tree which is equivalent to marking all files in

this folder). Now choose **File** → **Update** (Of course, you could have chosen this at the beginning of the session). For some of the files the status may change now. Typically, files which had ″Needs Patch″ or ″Needs Update″ are updated. So the following new items are possible in the status column:

Updated

Shown if the file was updated from the repository.

Patched

Indicates that the CVS server has sent a patch for this file and the patch has been successfully applied. If the patch was not successful because there was a conflict between your modifications and those someone else committed to the repository, the status is now **Conflict**.

You may have noticed that according to the status of the file, its row has a different color. The colors are chosen to somehow reflect the priority of the status. For example, a file with a conflict is marked red to show you that you have to resolve a conflict before you can continue working with the file. If your folder contains a high number of files, you may nevertheless lose the overview. To get more concise information about which files have an unusual status, simply click on the header of the **Status** column. The file list is then sorted by priority, so you have all important information at the top of the list. To get back to the alphabetically sorted view, click on the header of the **File name** column.

Chapter 3

Working With Files

All commonly used CVS functionality is directly available in Cervisia's main view. Commands usually act on several files at once, namely all which currently selected. If the selection includes folders, its interpretation depends on the settings made under the **Settings** menu. For example, if **Settings** → **Commit & Remove Recursively** is checked and you choose **File** → **Commit...** while a folder is selected, then all files in the tree under that folder are committed. Otherwise, only the regular files in the folder itself are affected.

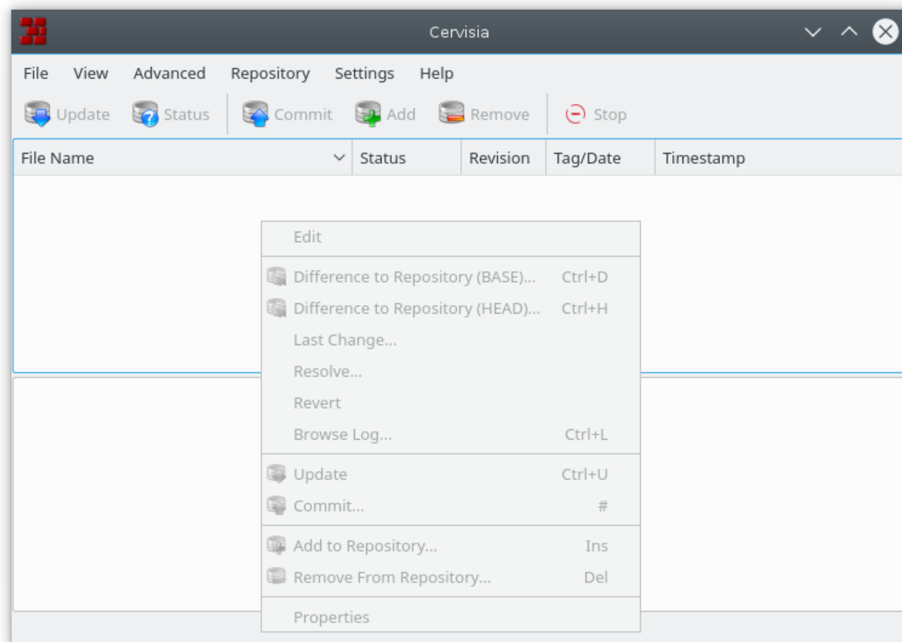

Figure 3.1: A screenshot of Cervisia's context menu

The most used actions are also available by right clicking the files in the tree view, through the context menu. Figure 3.1 shows Cervisia's main window context menu.

You can simply edit a file by double-clicking on it or selecting it and pressing **Enter**. This starts the default application that handles that file type (the default application for each file type is a KDE wide setting). If the default application is not the one you want to use, you can right click

the file and choose the **Edit With** submenu, and select one of the other applications that handle that file type.

3.1 Adding Files

Adding files to a project requires two steps: first, the files must be registered with CVS, or in other words, *added to the repository*. This is necessary, but not sufficient. In order to actually put the files into the repository, you must *commit* them. This procedure has an important advantage: you can commit the files together with modifications to other parts of the project. When doing this, one can easily see (e.g. in commit emails) that all these changes are part of a whole.

To this end, mark all files to be added in Cervisia's main view. Then, choose **File** → **Add to Repository...**, or right click the marked files and choose **Add to Repository...**. The **CVS Add** dialog will appear, listing the files you marked, and asks for confirmation. Press **OK**.

Cervisia issues a command

```
cvs add file names
```

If the operation was successful, the status column should have "Added to repository" for the added files.

> WARNING
> CVS is not designed to provide meaningful revision control for binary files. For example, merging binary files normally does not make sense. Furthermore, by default CVS performs keyword expansion (e.g. on the string `$Revision:  1.6 $`) when a file is committed. In binary files, such replacements may corrupt the file and make it completely unusable.

In order to switch the above behavior off, you should commit binary files (or other files, like Postscript or PNG images) by choosing **File** → **Add Binary...**. The **CVS Add** dialog will appear, listing the binary files you marked, and asks for confirmation. Press **OK**.

Cervisia issues a command

```
cvs add -kb file names
```

3.2 Removing Files

Like adding files, removing files is done in two steps: First, the files have to be registered as removed by choosing **File** → **Remove From Repository...** or right clicking the marked files and choosing **Remove From Repository...** from the context menu. The **CVS Remove** dialog will appear, listing the files you marked, and asking for confirmation. Press **OK**. Cervisia issues the command

```
cvs remove -f file names
```

After that, this modification to the sandbox has to be committed, possibly together with other modifications to the project.

> NOTE
> The above command only works if the file is up-to-date. Otherwise, you get an error message. This behavior is sensible: If you have modified the file compared to the version in the repository, or if someone else has made any modifications, you will first want to check if you really want to discard them.

3.3 Adding and Removing Folders

Folders are handled fundamentally different from ordinary files by CVS. They are not under revision control, i.e. you cannot tell which folders existed in the project at a certain time. Furthermore, folders can never be explicitly removed (except by removing them directly in the repository).

As a substitute, CVS follows the convention that a folder is "non-existent" in a version of the project if it is empty. This convention can be enforced by using the option `-P` to **cvs update** and **cvs checkout**. This option can be set in the menu **Settings** → **Prune Empty Folders on Update**.

A folder can be added to the repository choosing **File** → **Add to Repository...** or right clicking the marked folder and choosing **Add to Repository...** from the context menu. Note that in contrast to adding files, adding folders does not require a commit afterwards. Cervisia issues the command

```
cvs add dirname
```

3.4 Committing Files

When you have made a certain number of changes to your working copy, and you want other developers to have access to them, you *commit* them. With a commit, you place your versions of the modified files as new revisions into the repository. A subsequent update by another developer will bring your modifications into their working copy.

In order to commit a couple of files, select them in Cervisia's main view and choose **File** → **Commit...** or right click the marked files and choose **Commit...** from the context menu.

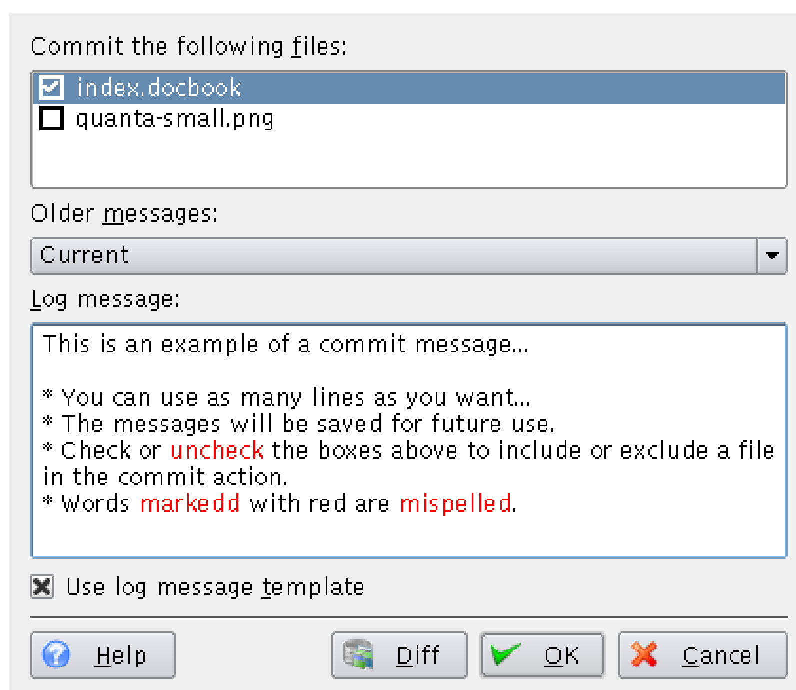

Figure 3.2: A screenshot of Cervisia's commit dialog

You get a dialog that shows you a list of the selected files on the top section and a log message for your changes below. Cervisia helps you in several ways to find a meaningful log message: first, in the file list you can double-click a file or press **Return** in order to see the changes you have

made to the file. Second, it gives you a list of log messages you have previously used in a combo box. Third, this dialog is integrated with Cervisia's changelog editor described below. When you have finished the dialog, the command

```
cvs commit -m message file names
```

is used.

> **NOTE**
> A common error you may encounter when committing is Up-to-date check failed. This indicates that someone has committed changes to the repository since you last updated; or, more technically, that your `BASE` revision is not the newest on its branch. In such a case, CVS refuses to merge your modifications into the repository. The solution is to update, resolve any conflicts and commit again. Of course, if you are working on a software project, it is normally good style to check if the program still works after you have updated - after all, there could be bad interactions between your modifications and the other modifications which break the code.

> **NOTE**
> Another popular mistake results in the error message Sticky tag 'X' for file 'X' is not a branch. This happens if you try to commit a file which you have previously brought to a certain revision or tag with the command
>
> ```
> %cvs update -r X
> ```
>
> (which is e.g. used by the menu item **Advanced** → **Update to Tag/Date...**). In such a case, the tag on the file gets sticky, i.e. further updates do not bring you to the newest revision on the branch. If you want to commit further revisions to the branch, you have to update to the branch tag before you do further commits.

With Cervisia, it is quite easy to maintain a ChangeLog file that is compliant with the format laid out in the GNU coding guidelines. To use it, choose **File** → **Insert ChangeLog Entry...**. If a file with the name `ChangeLog` exists in the toplevel folder of your sandbox, this file will be loaded and you have the possibility to edit it. To this end, at the top of the file, an entry with the current date and your user name (which can be configured as described in Section 6.1) is inserted. When you are finished the dialog can be closed by clicking **OK**, the next commit dialog you open will have the log message set to the message you last entered in the ChangeLog.

3.5 Resolving Conflicts

Conflicts may occur whenever you have made changes to a file which was also modified by another developer. The conflict is detected by CVS when you update the modified file; CVS then tries to merge the modifications committed by the other developer into your working copy. The merge fails if both your and his modifications are in overlapping parts of the file, and the CVS server issues an error message.

In Cervisia's main view, files with conflicts are indicated with "Conflict" in the status column and with a red color. It is your job now to resolve these conflicts before you commit the file. CVS will refuse to commit any files with conflicts until they have been edited. From the main view, you can of course resolve conflicts the traditional way: just double-click the file in question and edit it with your favorite editor.

CVS marks the conflicting changes by placing marks in the middle of the files, in the following manner:

```
<<<<<<<
Changes in your working copy
=======
Changes in the repository
>>>>>>> revision_number
```

You should replace this whole block with the new merged version. Of course, you have a great amount of freedom when resolving a set of conflicts: for each conflict you can decide to take one of the two alternative versions. You can also decide that both approaches are broken and rewrite a whole routine or the complete file from scratch.

Fortunately, Cervisia offers a nicer interface for handling these conflicts. This does not mean that you will never need to manually edit the files, but at least can eliminate the need to do so for the trivial conflict resolution. To use Cervisia's **CVS Resolve** dialog choose **File** → **Resolve...** or right click the marked file and choose **Resolve...** from the context menu.

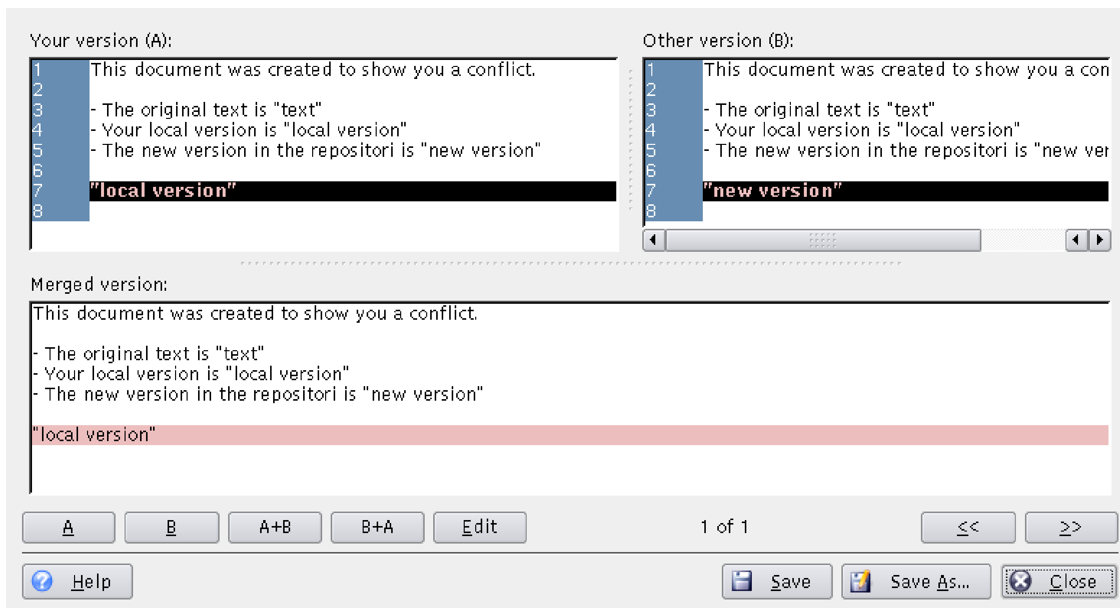

Figure 3.3: A screenshot of Cervisia's resolve dialog

On the top of the dialog, you see **Your version (A)** of the file on the left hand side and the version in the repository, **Other version (B)**, on the right hand side. The differences between them are marked in red color. Below these two versions, you can see the **Merged version**. The merged version reflects what that section will be in your working copy if you press the **Save** button.

You can go back and forward between the conflicting sections by pressing « and ». In the lower middle of the dialog you can see which section is currently marked. For example, `2 of 3` means that you are currently at the second differing section of 3 total.

Now you can decide section by section which version you want to have in the merged file. By pressing **A**, you take over the version you edited. By pressing **B**, you take over the version from the repository. By pressing **A+B**, both versions will be added, and your version will come first. **B+A** yields the same result, but the order will be different: first the repository version, then yours.

If you are not happy with any of these versions, press **Edit** to open a simple text editor where you can edit the section. When you are finished editing, press **OK** to return to the **CVS Resolve** dialog and resume solving conflicts. You will see the section you just edited in the **Merged version**, with your modifications.

To save your modifications, overwriting the working copy version, press **Save**. Note that this will save the choices not only the section you are currently viewing, but all sections in the file. If you want to save it to another file, press **Save As...**. Press **Close** to exit the dialog. If you close the dialog without saving, the changes you made will be lost.

Chapter 4

Obtaining Information About Files and Creating Patches

4.1 Watching Differences Between Revisions

There are several places in Cervisia where you can ask for a window showing the differences between revisions of a file:

- In the main view, you can choose **View** → **Difference to Repository (BASE)...**. This is based on the command **cvs diff** and shows you the differences between the version in your sandbox and the version to which you last updated (also known as `BASE`). This is in particular useful just before you commit a file, so you can find an appropriate log message.
- You can view the differences between the version in your sandbox and the version in the main development branch (also called `HEAD`) by choosing **View** → **Difference to Repository (HEAD)...**.
- You can view the differences between the last two revisions of the selected file choosing **View** → **Last Change...**.
- You can access the **Difference to Repository (BASE)...**, **Difference to Repository (HEAD)...** and **Last Change...** menu items from the main view context menu, by right-clicking the file you want to view.
- In the dialog that is shown when a you commit a set of files, you can request a difference window by selecting a file name in the selection list, either by double-clicking it or by pressing **Return**. This is quite similar to using **View** → **Difference to Repository (BASE)...** with the respective file in the main view.
- In the Browse Logs dialog, you can mark two revisions of a file and request a dialog showing the differences between them (see Section 4.4).

As you may have expected, Cervisia does not just dump the output of the **diff** command into your terminal, but shows you a graphical view as seen in Figure 4.1.

The text in the dialog is an improved variant of the text given by the diff command with the `-u` option. You can see the differing versions in two windows, with lines arranged such that you can do a side-by-side comparison. That means, where text has been added or deleted, the respective window shows empty lines with the marker `+++++` at the left hand side. Elsewhere, you can see the running number of each line in the left column.

In the second column in the right window, you can see which kind of change has been made. Possible types are `Add`, `Delete` and `Change`. The respective lines are marked in blue, green and

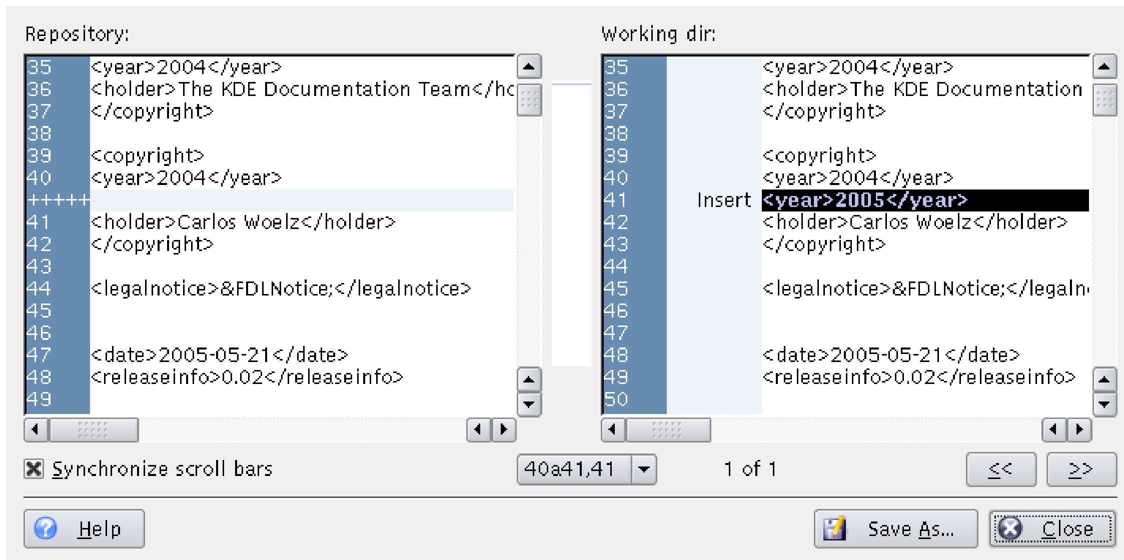

Figure 4.1: A screenshot of Cervisia's diff dialog

red color. In the middle of the dialog a compressed image of the color markers is shown. In this way, you can get a quick overview of the overall changes to the file. You can also use the position of the colored regions in the compressed image as an orientation when you using the scroll bars.

Normally, the scrollbars at the left and the right window are synchronized, i.e. if you scroll on the left hand side, the right hand side is scrolled by the same amount. You can change this by checking the box **Synchronize scroll bars**.

For information about how to customize the diff dialog, see Section 6.2.

4.2 Creating Patches

Sometimes you want to offer your modifications for review, before committing them, or you do not have write access to the repository (therefore you cannot commit). CVS offers standard formats to share the modifications in your working copy, so other people can review your changes, test them in their working copy and apply them to the CVS repository. A file containing these differences is called a *patch*, and is generated by the **cvs diff** command, in the same way as the differences in Section 4.1. Sharing patches instead of sets of files requires less bandwidth, and patches are easier to handle, as you can send only one patch file containing all the differences from many source files.

Cervisia gives you access to this feature by choosing **Advanced** → **Create Patch Against Repository...**.

> IMPORTANT
> The **Create Patch Against Repository...** action creates a patch with all modifications in all files in your working copy (sandbox) against the `BASE` repository. Therefore, the selection of files in the main view does not affect the patch that will be generated.

Another possibility is to select one file in the main view and choose **Browse Log...** from the **View** menu or right click the marked file and choose **Browse Log...** from the context menu, in order

to open the Browse log dialog. Now, select the version you want to create a patch against, as revision 'A' and press the button **Create Patch...**. This will generate a patch with the differences between the *marked file* in your working copy and the version selected as revision 'A'.

Before generating the patch, Cervisia displays a dialog allowing you to configure the output format.

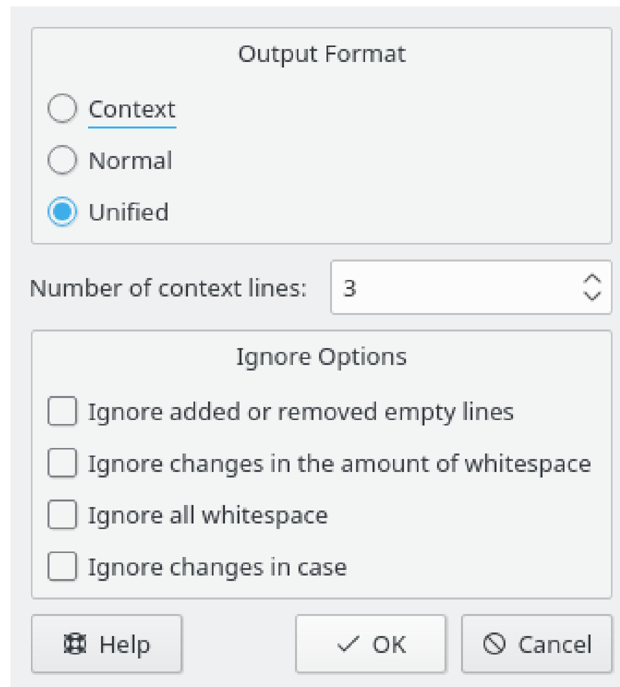

Figure 4.2: A screenshot of Cervisia's patch dialog

Output Format

There are three output formats available:

Normal: a format that can be used to cause the ed editor to automatically make another copy of the old file match the new file. In the normal output format, the characters < and > mark the changes, and there is no context information.

Unified: the most used format for exchanging patches. The unified format uses context lines in addition to line numbers to record the differences. This makes the process of applying patches more robust. This format displays the differences in a compact and readable form, with a header for each file involved, and separate sections (chunks) for each difference. The context lines available for each difference make reading the modifications easier. In the unified output format, the characters + and - mark the changes.

Context, which presents the same information as the unified format, but in a less compact way. In the context output format, the character ! marks the changes.

Number of context lines:

Set here the number of context lines for the unified or context output formats. This option is not available for the normal output format, as in this format no context information is recorded. More context information makes reading the raw output easier, and applying the patch more precise, but increases the patch size. It is recommended to use at least two context lines for proper patch operation.

Ignore Options

Check here the changes that should not be considered as differences when generating the patch.

After setting the output format, Cervisia generates the patch and displays the **Save As** dialog. Enter in this dialog the file name and location of the patch file.

4.3 Watching an Annotated View of a File

With the command **cvs annotate**, CVS offers the possibility to see - for each line in a file - who has modified a line the most recently. This view may be helpful in order to find out who has introduced a change in the behavior of a program or who should be asked about some change or bug in the code.

Cervisia gives you access to this feature, but it further enriches the information in an interactive way. You obtain an annotate view by choosing **View** → **Annotate...**. Another possibility is to press the button **Annotate** in the Browse log dialog, in which you can select which version of the file you want to display. In Figure 4.3 you can see a screenshot of the dialog.

Figure 4.3: A screenshot of Cervisia's annotate dialog

In the annotate dialog, you see in a window the latest version of the selected file (or the revision 'A'version, in case you launched the annotate dialog from the Browse log dialog). In the columns before the text, you get some information related to the latest change in each line. In the first column the line number is displayed. In the second column you see the name of the author and revision number. Finally, in the third column you see the actual content of that line.

Consequently, when a certain line appears strange to you or you assume a bug there, you can immediately see who is responsible for that line. But not only that, you can also find out *why* that line was changed. To this end, move the mouse cursor over the respective revision number. Then a tooltip appears that shows the log message and the date of the change.

4.4 Browsing CVS Logs

When you mark one file in the main view and choose **Browse Log...** from the **View** menu or right click the marked file and choose **Browse Log...** from the context menu, the **CVS Log** dialog is shown (if you mark more than one, nothing happens, as Cervisia can only generate and parse the log for one file at a time). This dialog offers functionality that is beyond viewing the file's history. Using it as a version browser you can:

- View the revision, author, date, branch, commit message, and tags for each version of the marked file.
- View a graphical tree representation showing the branching and tagging of the marked file.
- View any version of the marked file (with the default application).
- Watch an annotated view of any version of the marked file
- View the differences between any pair of versions of the marked file, including pairs with the current working copy version of the marked file.
- Create patches containing the differences between any pair of versions of the marked file, including pairs with the current working copy version of the marked file.

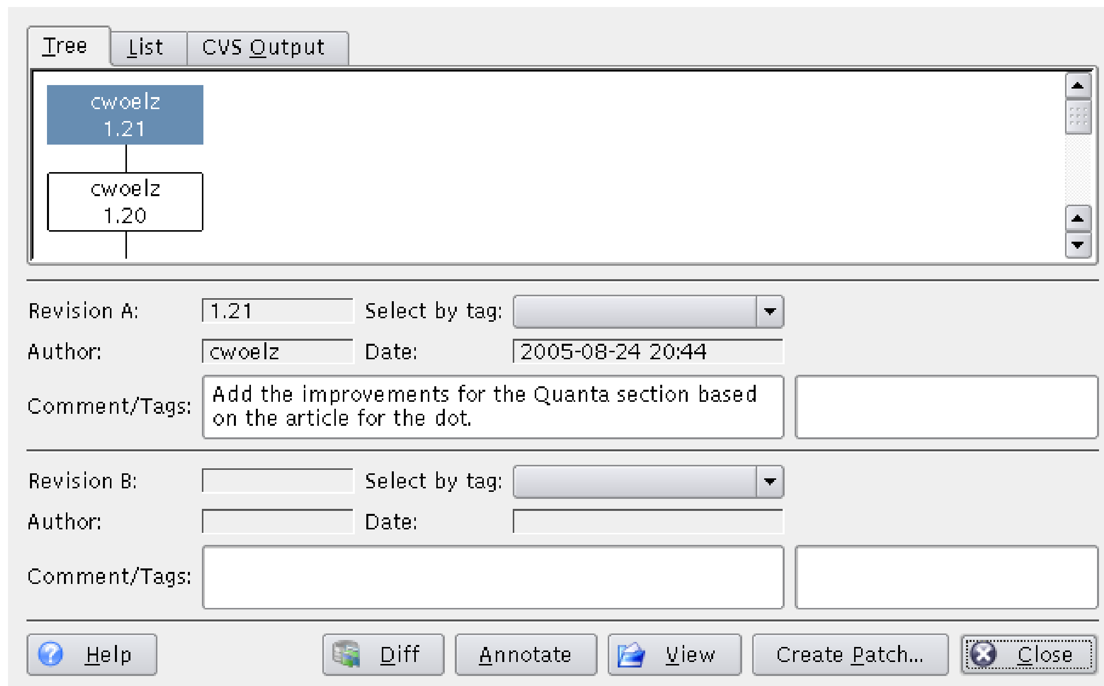

Figure 4.4: A screenshot of Cervisia's browse logs dialog

You can choose to see the history as provided by the **cvs log** command (**CVS Output**), as a **Tree**, or in **List** form. What you prefer is of course a matter of taste and it depends on what information you are interested in. The tree is an intuitive representation of what has been done on different branches by which authors. As tooltips, you can see the corresponding log messages. The list is by its nature linear and, therefore, does not give an immediate view of branches; on the other hand, it concentrates more otherwise relevant information on less screen estate, namely the time

of each change of the file and the first part of the log message. The CVS output information is complete, but long, and difficult to read. To alleviate these problems, you have the ability to search the text of the CVS output, by pressing the **Find...** button.

To obtain more information about a certain revision, you can click on it either in the list or the tree view. The fields in the middle of the dialog are then filled with the complete information provided by **cvs log**. You can mark two revisions, called ‘A’and ‘B’, which are relevant if you make use of further features provided by the buttons. Revision ‘A’can be chosen with the left mouse button, revision ‘B’ with the middle one. In the list view, you can also navigate with your cursor keys. In order to mark revisions ‘A’and ‘B’, use the shortcuts **Ctrl**+**A**, **Ctrl**+**B**, respectively. Using the **CVS Output** view, you can click on the **Select for revision A** and **Select for revision B** to mark the revisions.

If you press the **Annotate** button, you get a dialog showing the text of file belonging to the revision marked as ‘A’. Every line is prefixed with the information about who edited this last time, and at which revision this happened. You can get more information about viewing annotated versions in Section 4.3.

If you press the **Diff** button, a **cvs diff** call is issued and you get a dialog in which all the modifications between the two marked revisions are shown. If you mark revision ‘A’, but not revision ‘B’, Cervisia will generate the modifications between the file version marked as revision ‘A’and the working copy version of the file. This allows you to view the differences between your version of the file and any version available in CVS. To make it easy to see the changes, different colors are used to mark lines which have been added, removed or simply changed. You can get more information about viewing differences in Section 4.1.

If you press the **Create Patch...** button, you get a dialog in which you can set the format options for generating a file containing all the modifications between the two marked revisions which are shown. If you mark revision ‘A’, but not revision ‘B’, Cervisia will generate the modifications between the file version marked as revision ‘A’and the working copy version of the file. This allows you to generate a patch, or difference file, between your version of the file and any version available in CVS. After configuring the format of the patch in the dialog, and pressing **OK**, a **cvs diff** command is issued to generate the difference file. A **Save As** dialog will pop up. Enter the file name and location of the patch file Cervisia generated, in order to save it. You can get more information about creating patches, and the patch format options in Section 4.2.

If you press the **View** button, Cervisia will retrieve the revision marked as ‘A’and display it using the default application for its file type.

Press the **Close** button to leave the dialog and return to the main view.

To generate the log that is the base for the **CVS Log** dialog, Cervisia issues the following command:

```
cvs log file name
```

4.5 Browsing the History

If the used repository has logging enabled, Cervisia can present you a history of certain events like checkouts, commits, tags, updates and releases. Choose **History** from the **View** menu, and Cervisia will issue the command

```
cvs history -e -a
```

> NOTE
> This fetches the complete logging file from the server, i.e. a list of the events for all users and all modules. This can be a huge amount of data.

Now you can see the list of events, sorted by date. In the second column, the type of the event is shown:

- Checkout - The user who is displayed in the 'Author' column has checked out a module
- Tag - A user has used the command **cvs rtag**. Note that the usage of **cvs tag** (as done by Cervisia's **Advanced** → **Tag/Branch...** command) is not recorded in the history database. This has historical reasons (see the CVS FAQ).
- Release - A user has released a module. Actually, this command is rarely used and not of much value.
- Update, Deleted - A user has made an update on a file which was deleted in the repository. As a consequence, the file was deleted in his working copy.
- Update, Copied - A user has made an update on a file. A new version was copied into working copy.
- Update, Merged - A user has made an update on a file. The modifications in the repository version on the file were merged into his working copy.
- Update, Conflict - A user has made an update on a file, and a conflict with his own modifications was detected.
- Commit, Modified - A user committed a modified file.
- Commit, Added - A user added a file and committed it.
- Commit, Removed - A user removed a file and committed it.

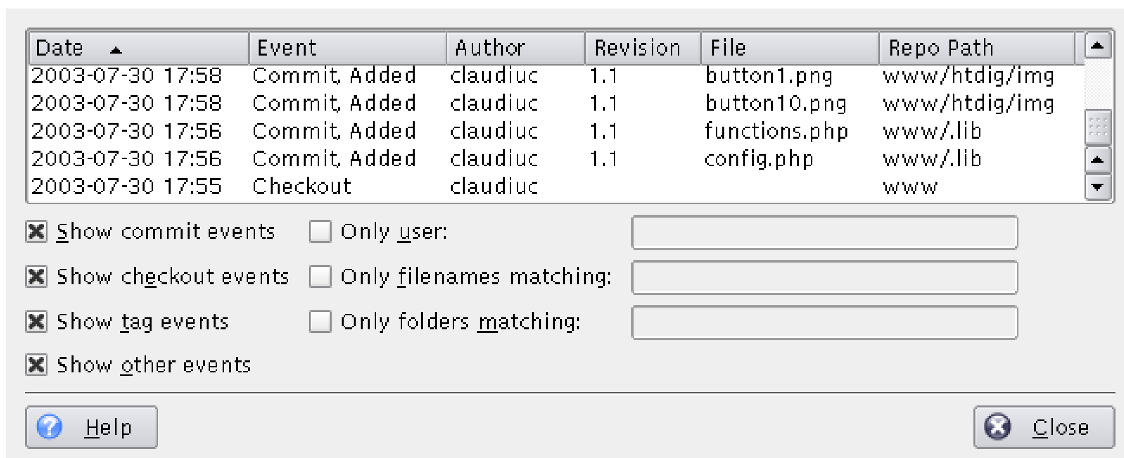

Figure 4.5: A screenshot of Cervisia's history dialog

You can sort the list by other criteria simply by clicking on the respective column header. In order to sort out the history entries you are interested in, there are various filter options activated by check boxes:

- Show commit events - shows commits
- Show checkout events - shows checkouts
- Show tag events - shows taggings
- Show other events - shows events not included in the above

- Only user - shows only events caused by a certain user
- Only file names matching - filters file names by a regular expression
- Only folders matching - filters folder names by a regular expression

Special characters recognized by the regular expression matcher are:

- `x*` matches any number of occurrences of the character `x`.
- `x+` matches one or more of occurrences of the character `x`.
- `x?` matches zero or one occurrences of the character `x`.
- `^` matches the start of the string.
- `$` matches the end of the string.
- `[a-cx-z]` matches a set of characters, e.g. here the set consisting of a,b,c,x,y,z.

Chapter 5

Advanced Usage

5.1 Updating to Tag, Branch or Date

Branches of a module are parallel versions of this module. A good real life example of the use of this feature is the release of a software project. After a major release, there are bugs in the code that should be fixed, but people want to add new features to the application too. It is very hard to do both at the same time because new features usually introduce new bugs, making it hard to track down the old ones. To solve this dilemma, CVS lets you create a parallel version, that we will call the 'stable release branch', where you can only add bugfixes, leaving the main branch (HEAD) open for adding new features.

Tags are used to mark a version of a project. CVS stamps one version of each file with the tag, so when you checkout or update to a specific tag, you will get always the same file versions; therefore, as opposed to branches, tags are not dynamic: you cannot develop a tag. Tags are useful to mark releases, big changes in the code, etc.

When you are developing or following the development of a software project, you do not necessarily work with the main branch all the time. After a release, you may want to stay with the released branch for a while, to enjoy its relative stability, fix bugs, translate the sources, etc. To do all that, you have to update to the released branch. All your files will be updated to the latest version of the files in that branch. After updating, all your new commits will be uploaded to the new branch as well.

Also, if you want to track a bug that was reported against a past tagged release, CVS offers you the possibility to retrieve the software as it was released, by updating to that tag. Besides, if you want to fetch a past version of your project, you can update your working copy to a specific date. This may be useful if an error was introduced in the project between two releases, and you have an opinion on when that was. When you update to a date or tag, the versions of your files will be the same as the versions in that specific date or the versions stamped by that tag.

> WARNING
> Before updating to a different branch or tag, make sure you committed all your changes to the branch you are working with. If your are not ready to commit your changes, but do not want to discard them, do not update to the new branch, as you may lose your changes. As an alternative, you can do a new checkout, to work in parallel with both versions.

Update to branch

Select this option to update to a branch. Enter the name of the branch in the drop down text box (or press the **Fetch List** button to retrieve the list of branches from the CVS server, and select the one you want in the drop down list).

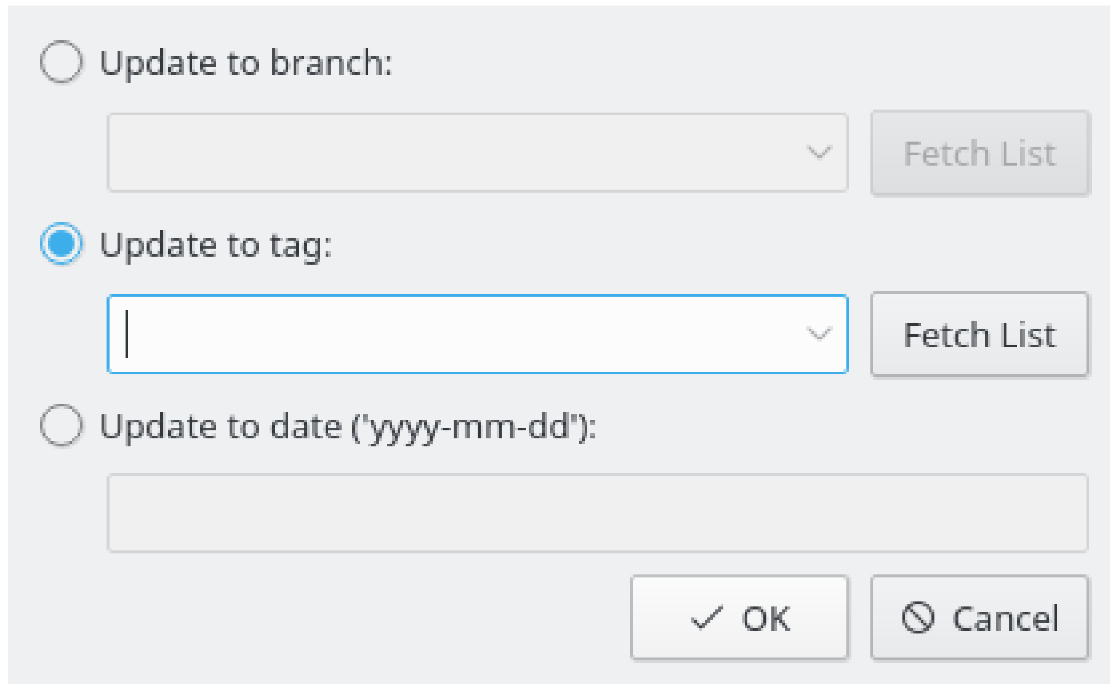

Figure 5.1: A screenshot of Cervisia's update to tag dialog

Update to tag

Select this option to update to a tag. Enter the name of the tag in the drop down text box (or press the **Fetch List** button to retrieve the list of tags from the CVS server, and select the one you want in the drop down list).

Update to date

Select this option to update to a date. In the field below, you can enter a wide variety of date formats. One possible format is `yyyy-mm-dd` where `yyyy` is the year, `mm` is the month (numerically) and `dd` is the day. Alternatives are some English phrases like `yesterday` or `2 weeks ago`.

> NOTE
> Updating to a tag or date make them 'sticky', i.e. you cannot commit further modifications on that files (unless the tag is a branch tag). In order to get back to the main branch, use the menu item **Advanced** → **Update to HEAD**.

The command issued to update to a branch or tag is:

```
cvs update -r tag
```

The command issued to update to a date is:

```
cvs update -D date
```

The command issued to update to the main branch (HEAD) is:

```
cvs update -A
```

5.2 Tagging and Branching

We discuss here only the technical aspects of tagging and branching. If you are only a *user*, not the administrator of the repository, you will probably not be confronted with the problem. If however you are your own administrator, you should first read about the non-technical problems that accompany branching, in order to get an impression of how time-consuming and error-prone maintaining different branches of a project can be. The appendix includes some references about this topic.

Simple tagging is something you usually do when a release is made, so that you can at any time easily get back to the project state at that time. Tags are usually given a name consisting of the project name and the version number. For example, Cervisia 1.0 is available under the tag `CERVISIA_1_0`. Cervisia enforces CVS's strict rules about what constitutes valid tag name. It must begin with a letter and may contain letters, digits, hyphens and underscores.

Normally, you will want to tag the whole project (although CVS of course allows you to tag only a subset). To this end, mark the toplevel folder in the view and choose **Advanced** → **Tag/Branch**. Now enter the name of the tag, press **Return** and you are done.

Creating a branch is not significantly more difficult: In the tag dialog, check the box **Create branch with this tag**. You can also delete an existing tag: Choose **Advanced** → **Delete Tag** in the main view.

Another aspect of branching is the merging of modifications from a branch to the current branch. If you are going to do this, choose **Advanced** → **Merge...**. The dialog that appears now gives you two options:

Either you may merge all modifications done on a branch to the current branch. In that case, check the box **Merge from branch** and fill in the branch you want to merge from. Cervisia will then execute the command

```
cvs update -j branchtag
```

The other possibility is that you want to merge only the modifications made between two tags on a branch. This usually happens when you merge from the same branch to the trunk several times. In that case, check the box **Merge modifications** and enter (in the correct order) the two relevant tags. This will result in a command

```
cvs update -j branchtag1 -j branchtag2
```

5.3 Using Watches

A watch is the conventional name for CVS's feature to notify users of the repository whenever a file has been changed or a developer has started editing a file. The usage of watches requires that the file `$CVSROOT /CVSROOT/notify` has been set up properly. This is not discussed here; if you need further information on the setup from the administrator's point of view, read one of the books listed in the appendix.

Cervisia's main support of watches are six menu items.

In order to add a watch to one or several files, use **Advanced** → **Add Watch...**. In the dialog you get, you can choose to get notified for any of the types of events that CVS supports. For example, if you only want to get notified when a file is committed, check the boxes **Only** and **Commits**. If you want to get notified about any event related to the marked files, check the box **All**. The command line used when you accept the dialog is

```
cvs watch add -a commit file names
```

or with a similar option, depending on the events you chose to watch.

If you are not interested in some files anymore, you can remove your watches on them. To this end, use **Advanced** → **Remove Watch...**. In the dialog you get here, the same options are offered as in the form you filled out when adding the watch. When you confirm this dialog, Cervisia issues the command

```
cvs watch remove file names
```

possibly with an option `-a` for the chosen events.

Finally, you can get a list of the people who are watching a couple of files. Choose **Advanced** → **Show Watchers**. Using this menu item will result in a command

```
cvs watchers file names
```

In the normal usage scenario of CVS, each developer works separately in his checked out sandbox. When he wants to modify some file, he can just open it in his editor and start working on it. Nobody else will know about this work until the file gets committed.

For some developer groups, this is not the preferred model of cooperation. They want to get notified about someone working on a file *as soon as* he starts with it. This can be achieved by some further CVS commands. Before you start editing a file, select it in Cervisia's main window and choose **Advanced** → **Edit Files**. This will execute the command

```
cvs edit file names
```

This will send out a notification to everyone who has set an `edit` watch on this file. It will also register you as an *editor* of the file. You can obtain a list of all editors of a certain file by using **Advanced** → **Show Editors**. This is equivalent to typing on the command line

```
cvs editors file names
```

An editing session is automatically ended when you commit the affected file. At that moment, an `unedit` notification gets sent out to all people who have registered a respective watch on the file. Of course, sometimes you may not want to commit the file, but abort the editing session and revert to the previous version of the file. This is done by using **Advanced** → **Unedit Files**. Note that Cervisia will not ask you for confirmation; that means if you use this menu item, all your work done since you used **Advanced** → **Edit Files** will be lost. Precisely, Cervisia uses the command line

```
echo y | cvs unedit file names
```

So far, we have only discussed the case where edits and unedits are used voluntarily be the developers. In addition CVS supports a model which *enforces* the usage of these commands. The responsible command to switch to this model is **cvs watch on** which we will not explain further because it is mostly used by the administrator of the repository. However, the important point from the developer's point of view is that when the project enforces edits, working copies are checked out *readonly*. That means you cannot edit a file by default (unless you use tricks like **chmod**). Only when you use **Advanced** → **Edit Files**, the file becomes writable. It is made read-only again when you commit the file or use **Advanced** → **Unedit Files**.

Cervisia's editor interface helps you with projects that enforce watches also in a different way. If you just started an editor with a readonly file by double-clicking on it or by using **File** → **Edit**, you would not be able to save your modifications later. This has of course a reason: Whenever you want to change a file, you should run **cvs edit** before, so that all people watching the file get a notification that you are working on it.

In such a case, it is advisable to check the option **Settings** → **Do cvs edit Automatically When Necessary**. Now, whenever you edit a file by double-clicking it, Cervisia will run **cvs edit** before the editor is actually executed. Then you can edit your file as usual. When you have finished your work, commit your files, and the committed files are read-only again.

5.4 Locking

The development model usually followed when CVS is used is called *unreserved checkouts*. Each developer has his own sandbox where he can edit files as he likes. If when the watch features - like **cvs edit** - are used, multiple developers can work on files synchronously. Changes done by a different developer are merged into the local sandbox when an update is performed.

Other revision control systems - like RCS and SourceSafe use a different model. When a developer wants to a edit a file, he has to *lock* it. Only one developer at a time can a lock a file. When he has finished editing, the lock is released. On the one hand, with this model, conflicts can never happen. On the other hand, two developers cannot work on the same file at the same time, even when their changes do not affect each other. This can be a bottleneck. We are not going to discuss the organizational benefits of both approaches. Nevertheless we mention that although CVS has some support for locking, it is not the preferred way of working with CVS. You should not use these features unless you are sure that your project manager allows them.

With Cervisia, you lock files as follows. Select the desired files in the main view. Then choose **Advanced** → **Lock Files**. This runs the command

```
cvs admin -l file names
```

The reverse effect is achieved by using **Advanced** → **Unlock Files**. This runs the command

```
cvs admin -u file names
```

Chapter 6

Customizing Cervisia

Cervisia can be customized in various ways to your needs and preferences. Some options which you may want to change regularly are directly available in the **Settings** menu.

6.1 General

User name for the change log editor:

Whenever you use the menu item **File** → **Insert ChangeLog Entry...**, a new ChangeLog entry is generated with the current date and your username. Normally, it is considered good style to insert your full name and your email address into each of your ChangeLog entries. Cervisia automatically adds the full name and email address entered here.

Path to CVS executable, or 'cvs':

Here you can set the name (or path) to the **cvs** command line client. By default, the CVS executable found in your `$PATH` is used by Cervisia.

6.2 Diff Viewer

Number of context lines in the diff dialog:

For the diff dialog, Cervisia uses the option `-U` to **diff**. This lets **diff** show only a limited number of lines around each difference region (context lines). Here you can set the argument to `-U`.

Additional options for cvs diff:

Here you can add additional arguments to the **diff**. A popular example is `-b` which lets **diff** ignore changes in the amount of whitespace.

Tab width in diff dialog:

In the diff dialog, tab characters present in your file or in the output of the **diff** command are expanded into a fixed number of space characters. By default, each tab is replaced by eight spaces, but here you can setup a different number.

External diff frontend:

When you use any of the features which show the diff dialog, like **View** → **Difference to Repository...**, Cervisia invokes its internal diff frontend. If you prefer a different one, like Kompare, TkDiff, or xxdiff, enter its file name and path here.

6.3 Status

When opening a sandbox from a remote repository, start a File->Status command automatically

When you check this option, the **File** → **Status** command is started whenever you open a remote sandbox. This command may need some time and also needs a connection to the server for remote repositories (making it unusable for offline usage).

When opening a sandbox from a local repository, start a File->Status command automatically

When you check this option, the **File** → **Status** command is started whenever you open a local sandbox.

6.4 Advanced

Timeout after which a progress dialog appears (in ms):

Practically all CVS commands started in a sandbox which belongs to a remote repository need a connection to the CVS server. This is affected by delays from the network connection or a high load on the server. For this reason, for commands like **View** → **Difference to Repository...** Cervisia opens a dialog which indicates that the command is still running and which allows you to abort it. Furthermore, this dialog is used to show you error messages from CVS. As this dialog may become annoying after some time, it is shown only after a certain timeout which is 4 seconds by default. Here you can change this value.

Default compression level:

The **cvs** client compresses files and patches when they are transferred over a network. With the command line option `-z`, the compression level can be set. You can setup Cervisia to use this option by configuring the level here. The value set here is used only as a default; additionally there is a per-repository setting available in **Repository** → **Repositories...**.

Utilize a running or start a new ssh-agent process

Check this box if you use ext (**rsh**) repositories, the **ssh** remote shell to communicate with the repository and ssh-agent to manage your keys.

6.5 Appearance

Font for Protocol Window...

Press this button to open the **Select Font** dialog, to set the font used in the protocol window (this is the window showing the output of the **cvs** client).

Font for Annotate View...

Press this button to open the **Select Font** dialog, to set the font used in the annotate view.

Font for Diff View...

Press this button to open the **Select Font** dialog, to set the font used in diff dialogs.

Colors

Press the colored buttons to open the **Select Color** dialog, to set the color used for **Conflict**, **Local change**, or **Remote change**, in the main view or **Diff change**, **Diff insertion**, or **Diff deletion**, in Cervisia's built-in diff frontend.

Split main window horizontally

Cervisia's main window is normally split vertically into a window with the file tree above and one with the CVS output below; alternatively, you can arrange them horizontally.

Chapter 7

Appendix

7.1 Ignored Files

In its main file tree, Cervisia does not display all files which are actually there. This is analog to **cvs** itself and helps to avoid clutter caused by uninteresting stuff like object files. Cervisia tries to mimic **cvs**'s behavior as close as possible, i.e. it gets ignore lists from the following sources:

- A static list of entries which includes things like `*.o` and `core`. For details, see the CVS documentation.
- The file `$HOME /.cvsignore`.
- The environment variable `$CVSIGNORE`.
- The `.cvsignore` file in the respective folder.

cvs itself additionally looks up entries in `$CVSROOT /CVSROOT/cvsignore`, but this is a file on the server, and Cervisia should be able to start up offline. If you are working with a group that prefers to use an ignore list on the server, it's probably a good idea to take a look which patterns are listed there and to put them into the `.cvsignore` file in your home folder.

7.2 Further Information and Support

- CVS comes with a complete set of documentation in the form of info pages, known as "The Cederqvist". If it is properly installed, you can browse it by typing in **`info:/cvs`** into the locationbar of Konqueror, or you can just choose **Help** → **CVS Manual** in Cervisia. An on-line PDF version of the Cederqvist is available on the web.

  As this book is maintained together with CVS, it is normally the most up-to-date reference; nevertheless, considering other documentation for learning to use CVS is recommended, in particular the following.

- Karl Fogel has written the excellent book Open Source Development with CVS. About half of this book is about the development process of Open Source software. The other half is a technical documentation of CVS. Thankfully, the technical part of the book has been made freely redistributable under the GPL, so that you can download a HTML version of it. A list of errata is available on the web page mentioned above.
- CVS issues are discussed on a dedicated mailing list.
- There is USENET group `comp.software.config-mgmt` dedicated to configuration management in general. CVS is only marginally a topic in this group, but nevertheless it may be interesting for discussing merits of various revision control systems compared to CVS.

7.3 Command Reference

7.3.1 The File Menu

File → Open Sandbox... (Ctrl+O)
Opens a sandbox in the main window. See Section 2.4.

File → Recent Sandboxes
Opens one of the sandboxes that were in use recently.

File → Insert ChangeLog Entry...
Opens the ChangeLog editor, prepared such that you can add a new entry with the current date. See Section 3.4.

File → Update (Ctrl+U)
Runs 'cvs update' on selected files and changes the status and revision numbers in the listing accordingly. See Section 2.4.

File → Status (F5)
Runs 'cvs -n update' on selected files and changes the status and revision numbers in the listing accordingly. See Section 2.4.

File → Edit
Opens the selected file in KDE's default editor for the selected file's type.

File → Resolve...
Opens a dialog for the selected file which allows you to resolve merge conflicts in it. See Section 3.5.

File → Commit... (#)
Allows you to commit the selected files. See Section 3.4.

File → Add to Repository... (Ins)
Allows you to add the selected files to the repository. See Section 3.1.

File → Add Binary...
Allows you to add the selected files to the repository as binaries (**cvs `add-kb`**). See Section 3.1.

File → Remove from Repository... (Del)
Allows you to remove the selected files from the repository. See Section 3.2.

File → Revert
Discards any local changes you have made to the selected files and reverts to the version in the repository (Option `-C` to **cvs update**).

File → Quit (Ctrl+Q)
Quits Cervisia.

7.3.2 The View Menu

View → Stop (Escape)

Aborts any running subprocesses.

View → Browse Log... (Ctrl+L)

Shows the log browser of the selected file versions. See Section 4.4.

View → Annotate... (Ctrl+A)

Shows an annotated view of the selected file, i.e. a view where you can for each line see which author modified it last. See Section 4.3.

View → Difference to Repository (BASE)... (Ctrl+D)

Shows the differences between the selected file in the sandbox and the revision you last updated (BASE). See Section 4.1.

View → Difference to Repository (HEAD)... (Ctrl+H)

Shows the differences between the selected file in the sandbox and the revision you last updated (HEAD). See Section 4.1.

View → Last Change...

Shows the differences between the revision of the selected file you last updated (BASE) and the revision before. See Section 4.1.

View → History...

Shows the CVS history as reported by the server. See Section 4.5.

View → Hide All Files

Determines whether only folders are shown in the main tree view. See Section 2.4.

View → Hide Unmodified Files

Determines whether unknown and up to date files are hidden in the main tree view. See Section 2.4.

View → Hide Removed Files

Determines whether removed files are hidden in the main tree view. See Section 2.4.

View → Hide Non-CVS Files

Determines whether files not in CVS are hidden in the main tree view. See Section 2.4.

View → Hide Empty Folders

Determines whether folders without visible entries are hidden in the main tree view. See Section 2.4.

View → Unfold File Tree

Opens all branches in the file tree so that you can see all files and folders. See Section 2.4.

View → Fold File Tree

Closes all branches in the file tree. See Section 2.4.

7.3.3 The Advanced Menu

Advanced → Tag/Branch...
Tags or branches the selected files. See Section 5.2.

Advanced → Delete Tag...
Removes a given tag from the selected files. See Section 5.2.

Advanced → Update to Tag/Date...
Brings the selected files to a given tag or date, making it sticky. See Section 5.1.

Advanced → Update to HEAD...
Brings the selected files to the respective HEAD revision. See Section 5.1.

Advanced → Merge...
Merges either a given branch or the modifications between two tags into the selected files. See Section 5.2.

Advanced → Add Watch...
Adds a watch for a set of events on the selected files. See Section 5.3.

Advanced → Remove Watch...
Removes a watch for a set of events from the selected files. See Section 5.3.

Advanced → Show Watchers
Lists the watchers of the selected files. See Section 5.3.

Advanced → Edit Files
Runs **cvs edit** on the selected files. See Section 5.3.

Advanced → Unedit Files
Runs **cvs unedit** on the selected files. See Section 5.3.

Advanced → Show Editors
Runs **cvs editors** on the selected files. See Section 5.3.

Advanced → Lock Files
Locks the selected files. See Section 5.4.

Advanced → Unlock Files
Unlocks the selected files. See Section 5.4.

Advanced → Create Patch Against Repository...
Creates a patch from the modifications in your sandbox. See Section 4.2.

7.3.4 The Repository Menu

Repository → Create...
Opens a dialog which allows you to create a new local repository. See Section 2.1.

Repository → Checkout...
Opens a dialog which allows you to checkout a module from a repository. See Section 2.3.

Repository → Import...
Opens a dialog which allows you to import a package into the repository. See Section 2.2.

Repository → Repositories...
Configures a list of repositories you often use and how to access them. See Section 2.1.

7.3.5 The Settings and Help Menu

Apart from the common KDE **Settings** and **Help** menus described in the Menu chapter of the KDE Fundamentals documentation Cervisia has these application specific menu entries:

Settings → Create Folders on Update

Determines whether updates create folders in the sandbox which were not there before (Option `-d` to **cvs update**).

Settings → Prune Empty Folders on Update

Determines whether updates remove empty folders in the sandbox. (Option `-P` to **cvs update**).

Settings → Update Recursively

Determines whether updates are recursive (Option `-r` to **cvs update**).

Settings → Commit & Remove Recursively

Determines whether commits and removes are recursive (Option `-r` to **cvs add**, **cvs remove** resp.).

Settings → Do cvs edit Automatically When Necessary

Determines whether **cvs edit** is executed automatically whenever you edit a file.

Help → CVS Manual

Opens the CVS info pages in the KHelpCenter.

Chapter 8

Credits And License

Program copyright

- 1999-2002 Bernd Gehrmann bernd@mail.berlios.de
- 2002-2008 the Cervisia authors

Documentation Copyright 1999-2002 Bernd Gehrmann bernd@mail.berlios.de and 2004 Carlos Woelz carloswoelz@imap-mail.com

This documentation is licensed under the terms of the GNU Free Documentation License.

This program is licensed under the terms of the GNU General Public License.