Version 2
August 2007

Daily use of TortoiseCVS

A guide to the commonly used functions of the TortoiseCVS client for source code management on the CropForge collaborative software development site

IRRI

This work was funded by the Generation Challenge Programme
http://www.generationcp.org

CONTENTS

1. Introduction

The Concurrent Versioning System (CVS) enables developers to work concurrently by using the **copy-modify-merge** model. Developers download a working copy of the source code from a CVS server, make changes, and subsequently upload those changes into the CVS repository. The source code is usually in a directory tree that contains the files that make up the software project.

The main reasons for using a source code version control system like CVS are:

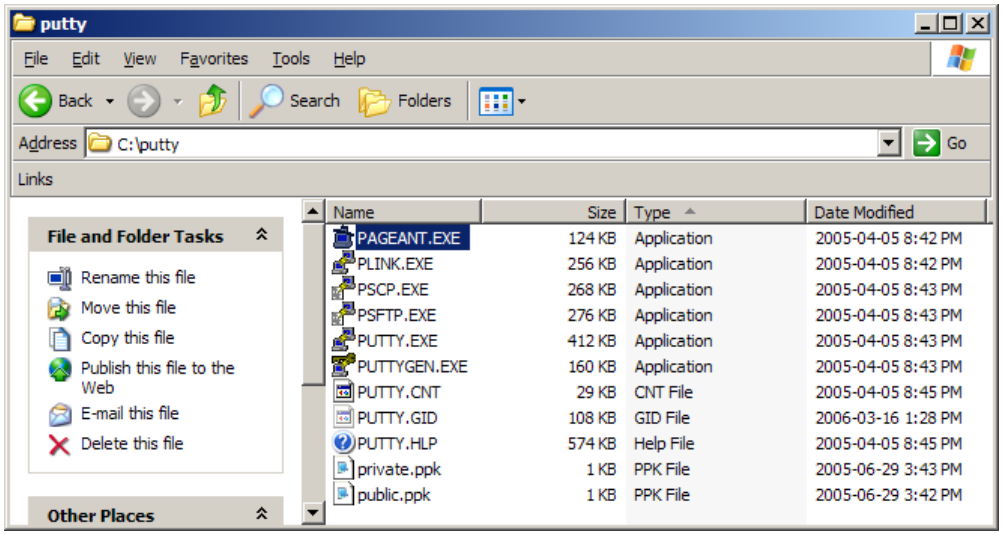
- The CVS repository serves as backup
- The CVS repository maintains a complete version history of every file
- The CVS repository facilitates multiple developers working concurrently on the same project

This manual describes the basic interaction with a CropForge CVS repository that is needed on a daily basis (e.g., update, commit, add, delete). It also addresses some more advanced topics in source code management (e.g., tagging, branching, merging) and release management.

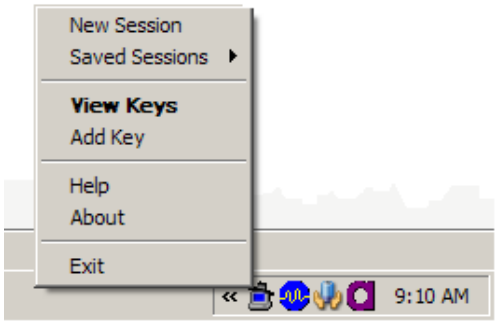
It is assumed that the TortoiseCVS software has already been installed on the client machine and that the user has an account on CropForge. The user must also be a developer in a project, and have either checked out an existing module from that project, or uploaded a new module to that project. Furthermore, it is assumed that the authentication software PuTTY has been installed, that a pair of public and private keys have been generated, and that the public key has been uploaded into the user's CropForge account. These issues are covered in separate manuals CodeOnCropForge.pdf and BeginningTortoiseCVS.pdf.

2. Loading the private authentication key

It is assumed that the PuTTY software is being used for authentication with the CropForge server. The default PuTTY installation should have resulted in a directory **c:\putty**. The public and private keys in the example below were saved under the names **private.ppk** and **public.ppk**.



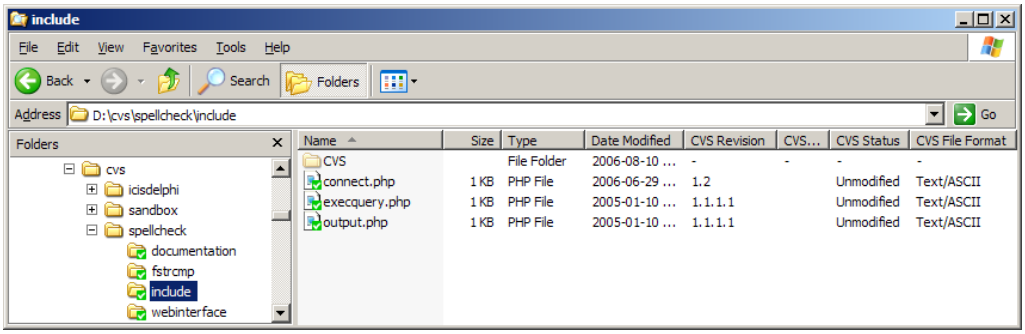
After starting the PAGENT.EXE program (PuTTY authentication agent), a corresponding icon appears in the taskbar. Right-click on the icon, select the option **Add Key**, and select the file containing your private key. In this example, the private key is stored in the file **c:\putty\private.ppk**.



It is assumed that your public key has already been uploaded to the CropForge server. Whenever authentication between you and the CropForge server is needed, it will now be done automatically by the PAGENT software using your locally loaded private key and your public key available on the CropForge server.

3. TortoiseCVS and Windows Explorer

TortoiseCVS integrates itself into Windows Explorer and maintains additional information for directories and files that are under version control. The most noticeable difference is the changed appearance of icons for directories and files that are under version control. Each directory under version control contains a "hidden" subdirectory named CVS in which TortoiseCVS maintains its administrative information.

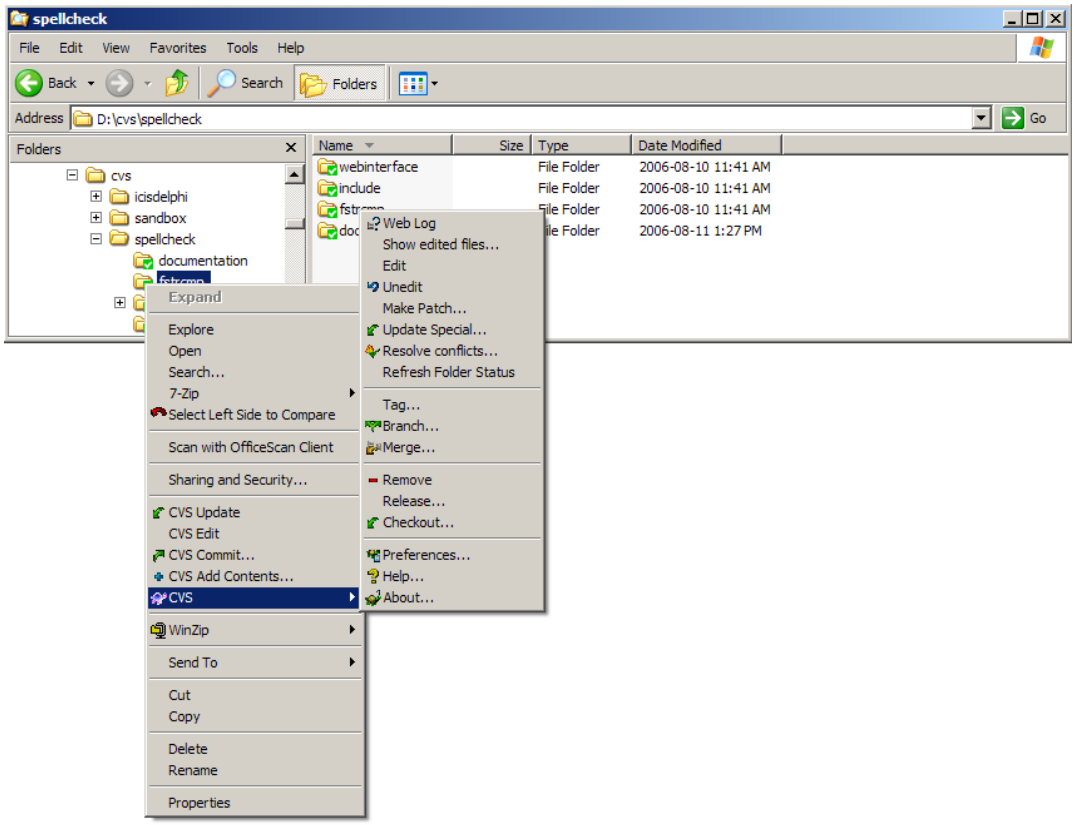


TortoiseCVS uses the following icon overlays to display the CVS status of the local files or folders within Explorer.

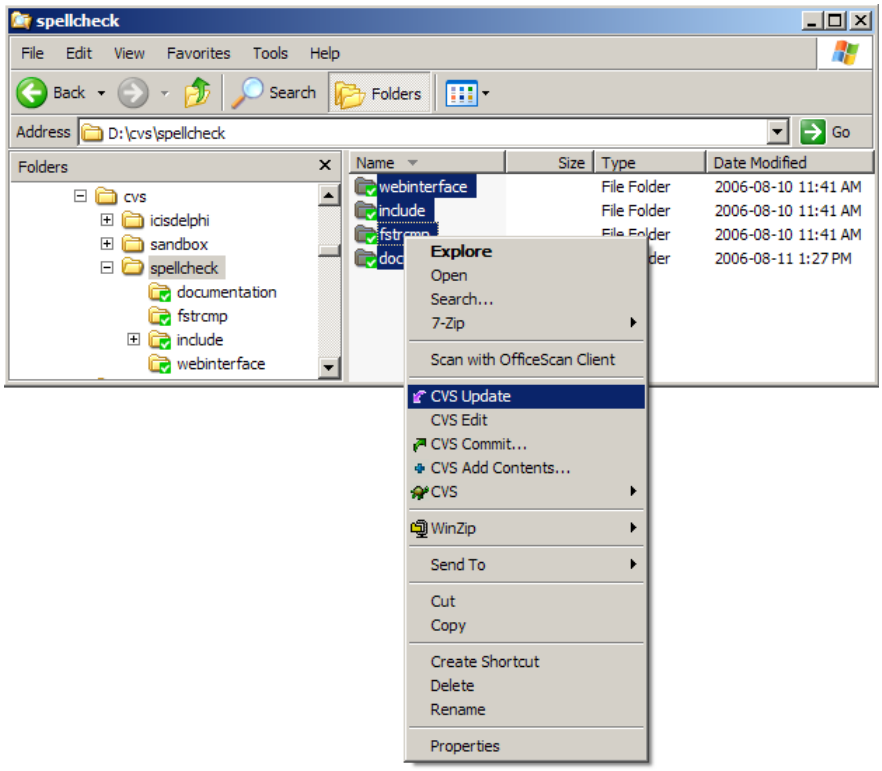
	***Not Modified***	The local version of the file or folder is up-to-date with the version in the remote CVS repository.
	***Modified***	The local version of file or folder has been modified and differs from the version in the remote CVS repository.
	***Added***	The file or folder has been added to the local copy of the source code under CVS, but has not yet been committed to the remote CVS repository.
	***Conflict***	The local file or folder has a conflict with the current file or folder in the remote CVS repository. A conflict occurs when two or more developers have changed the same few lines of a file in parallel.
	***Not in CVS***	The file or folder has not yet been added to the local copy of the source code under CVS.
	***Ignore***	The local file or folder is being ignored by CVS, thus it will not be affected by CVS operations.

When right-clicking on a directory or file under CVS version control, the Explorer menu shows additional options and a CVS sub-menu through which the TortoiseCVS functionality is accessible. Depending on the right-click context, i.e. selected files or folders, CVS operations can be performed on one or more selected files or on one or more selected directories and the files and sub-directories contained within.

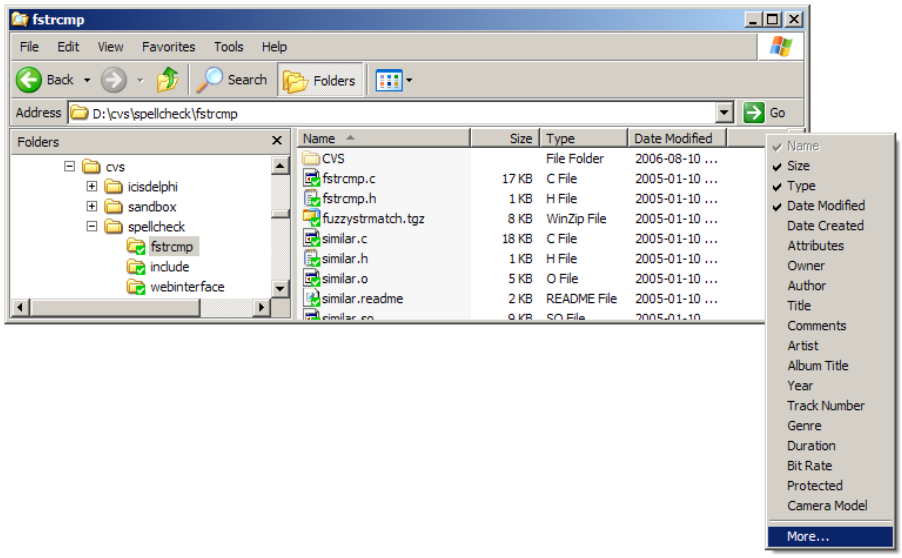
In the example below, a CVS operation is performed on the entire source code of the **fstrcmp** module.



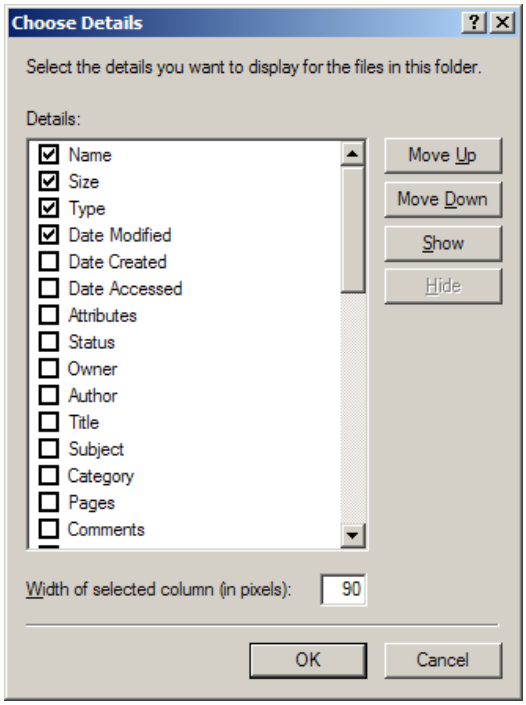
The example below shows how a CVS operation can be performed on all modules in a project. In a similar way, CVS actions can also be performed on one or more selected files.

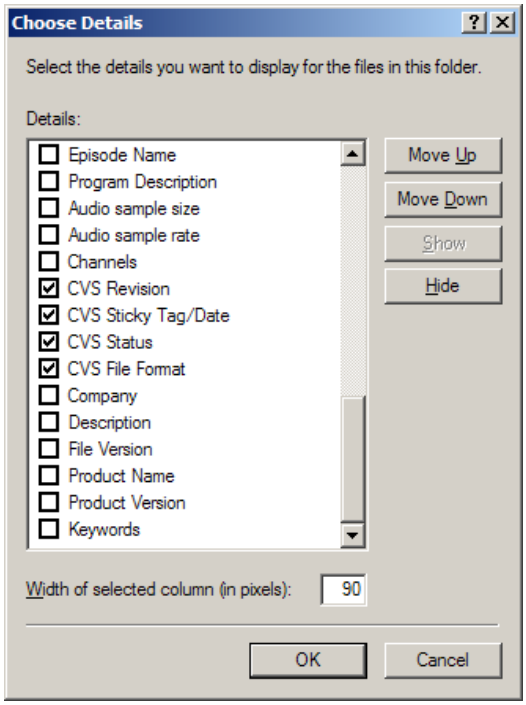


Windows Explorer can be configured to display the CVS file attributes, in addition to the normal attributes shown. In a module directory that is under CVS control, right-click on the file attributes title bar. A list of currently activated and additionally available attributes is shown.

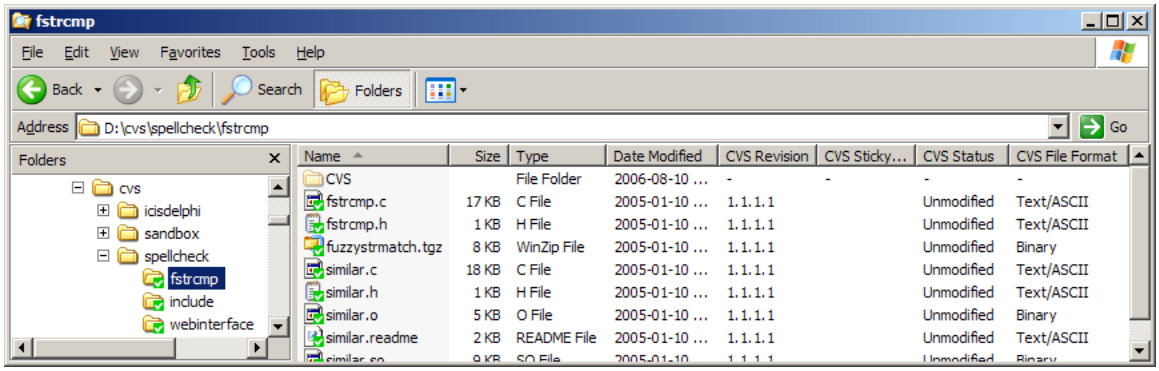


The **More…** option leads to the dialog where file attributes can be selected and arranged. Select the four CVS-specific file attributes in addition to the currently selected file attributes, then click the **OK** button.





Windows Explorer should now show the CVS file attributes in addition to the default file attributes. These additional CVS file attributes are very helpful in maintaining your local source code version. The above procedure needs to be done for every module subdirectory.



4. Updating your local copy from the CVS server

This action can be performed on one or more selected files, or on one or more selected directories and the files and sub-directories contained within.

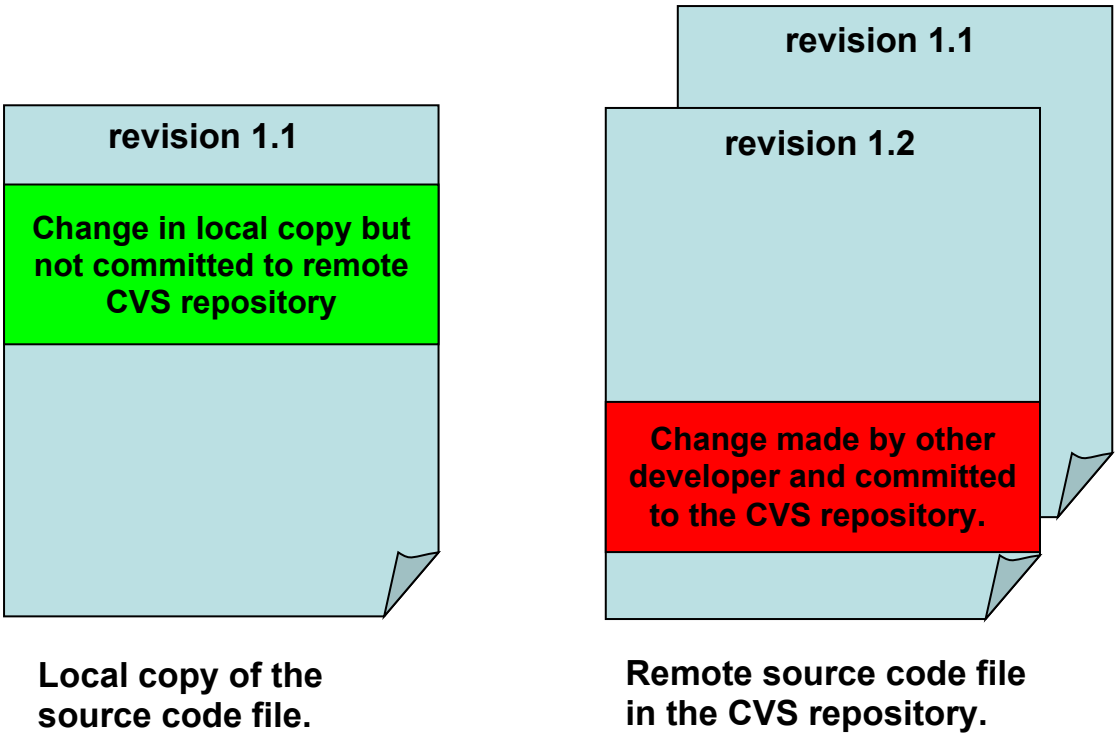
An **update** merges the differences that have occurred in the remote CVS repository since your last update into your current local copy of the source code. The local TortoiseCVS client transmits the current local revision numbers to the remote CVS server, which then generates the differences between those revisions and the latest revisions available on the remote server. These differences will be applied to your current local source code files. Such differences are usually the edits that other developers have made to the source code, and committed to the remote CVS repository.

It is important to note that an update does not copy and overwrite entire source code files, but that it merges changes. The utilities behind this process are called **diff** and **patch** and their use is described in more detail here: http://drupal.org/diffandpatch.

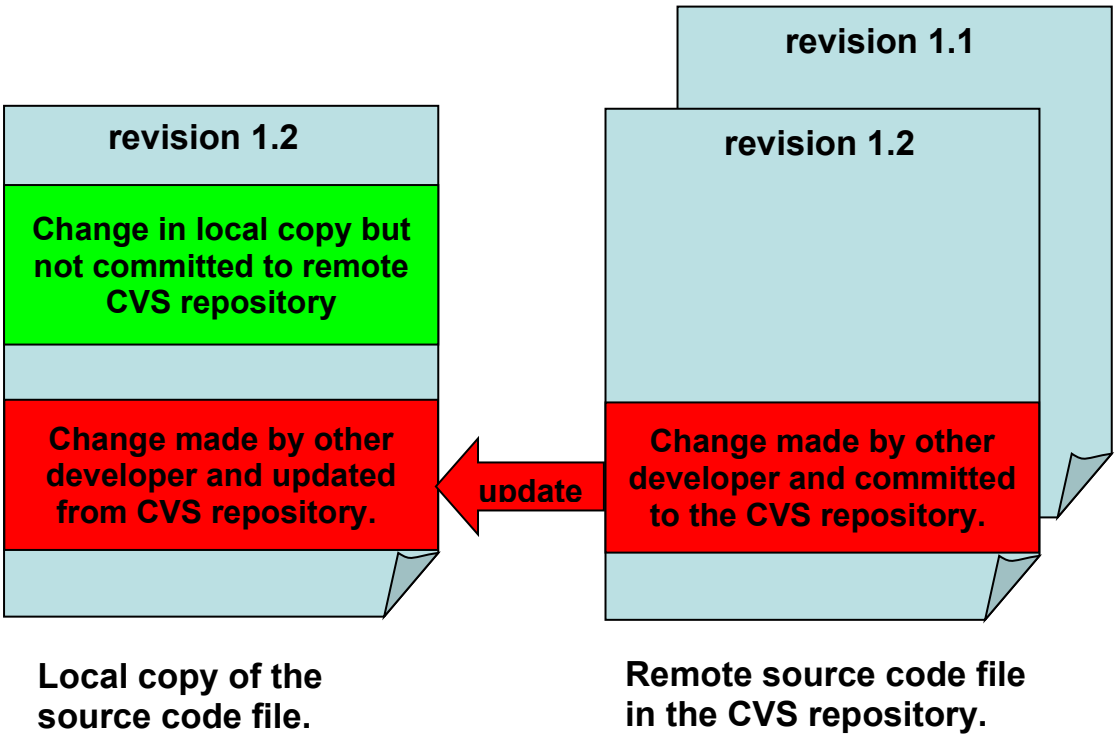
The diff-patch process is used by developers who do not have the right to commit changes to a remote CVS repository. Such developers would use the **diff** utility to create a patch file from the current CVS repository version of a file and their version of the same file. This patch file - accompanied by commit rights - is then submitted to a developer who will now be able to apply, review, and commit changes to the remote CVS repository.

The following diagram shows the different states of a source code file in the local copy (left) of the source code and in the remote CVS repository (right), before a CVS update. Note that the local revision number is still at 1.1, although changes have been made in the file (green box). Since the local changes have not been committed to the remote CVS repository, the revision number has remained unchanged.

Another developer has made changes to the file (red box) and committed these changes to the remote repository. This commit has created a new revision (1.2) of the source code file in the remote CVS repository.

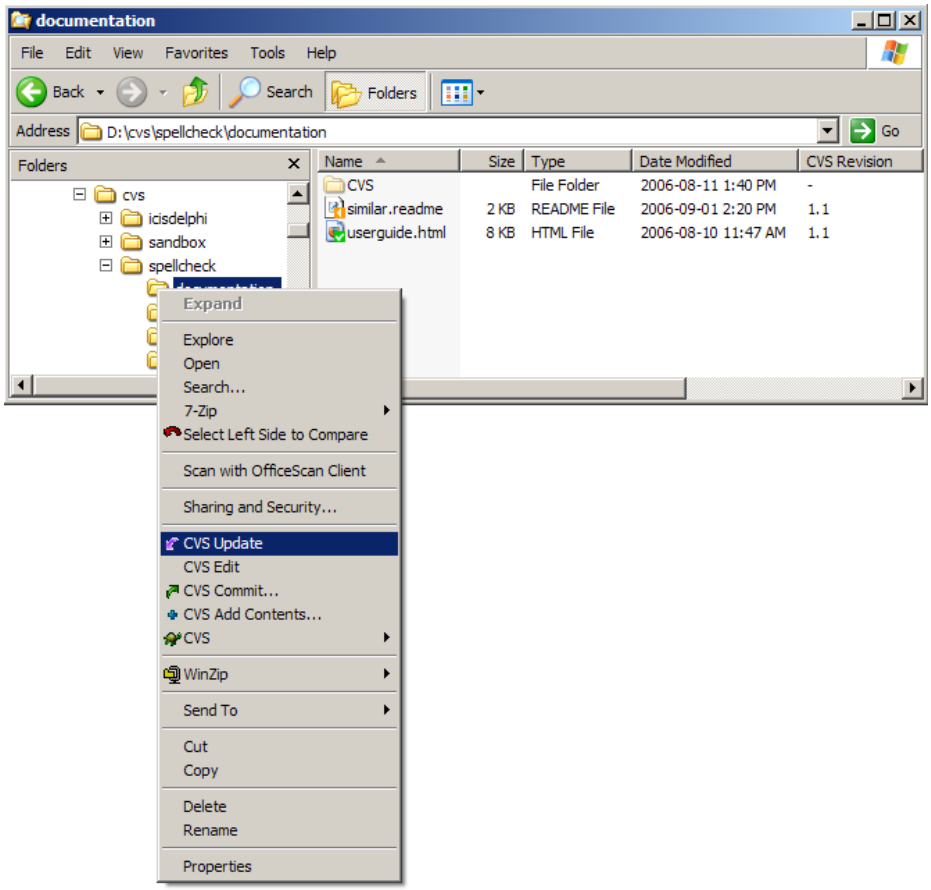


The **update** process transfers the changes derived between revision 1.1 and revision 1.2 on the remote CVS repository to the local copy. This update process causes also a change in the revision number in the local copy.



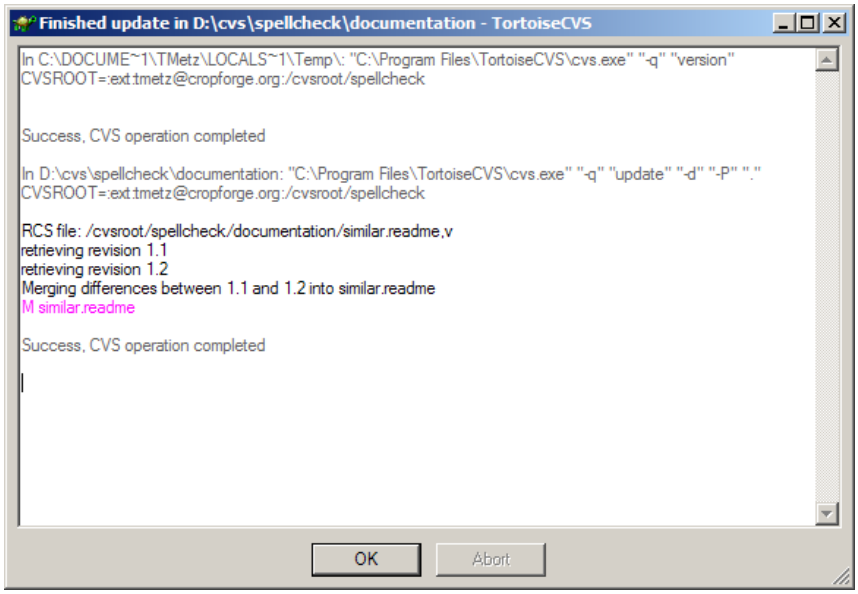
The following screenshots show the CVS update process as done in TortoiseCVS.

The module **documentation** in the project **spellcheck** contains a changed file (similar.readme) in the local copy. This change has not been committed to the remote CVS repository, as indicated by the icon overlay. In this example, we have selected the module directory to be updated, not only the file that we know has changed locally. If the file **userguide.html** has been changed by another user in the remote CVS repository, it will also be updated by this process.

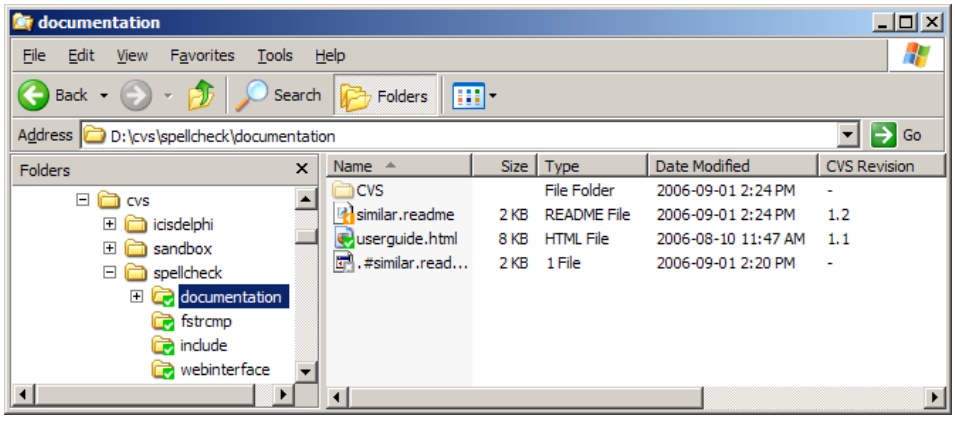


The update process generates a message window that shows the details of the communication between the CVS client and the CVS server, as well as the actions performed.

According to the message, there was a revision 1.2 of the file **similar.readme** in the remote CVS repository. Revisions 1.1 and 1.2 of this file were retrieved on the remote CVS repository, and their difference was merged into the local copy of the file. Note that the file **userguide.html** has not been updated, since there was apparently no new revision of this file in the remote CVS repository.



After the update, the local copy of the file now shows the revision number 1.2. The overlay icon still indicates that the file **similar.readme** has been changed locally and is not yet in sync with the remote CVS repository. Note that the file **userguide.html** has remained at revision 1.1.



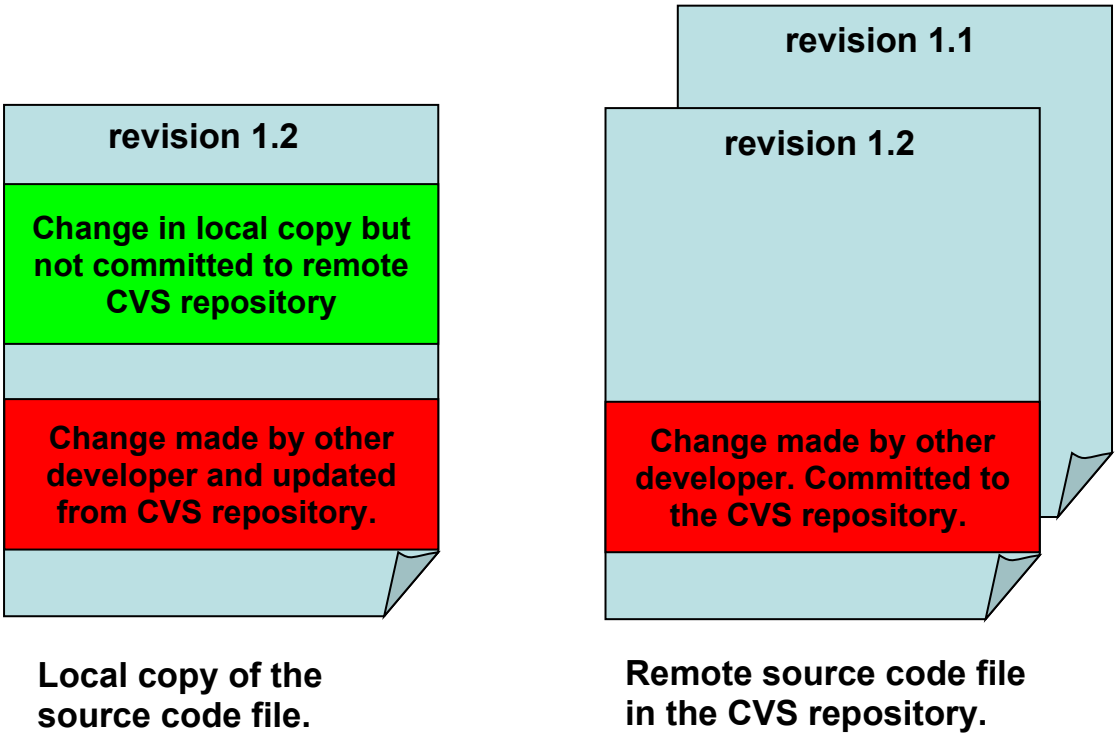
At this stage, we have incorporated the changes made by other developers into our local copy of the source code, but we have not yet uploaded (committed) our changes to the remote CVS repository. A **commit** operation must be performed in order to upload these changes (see **Section 5: Committing your local changes to the CVS server**).

5. Committing your local changes to the CVS server

This action can be performed on one or more selected files, or one or more selected directories and the files and subdirectories contained within.

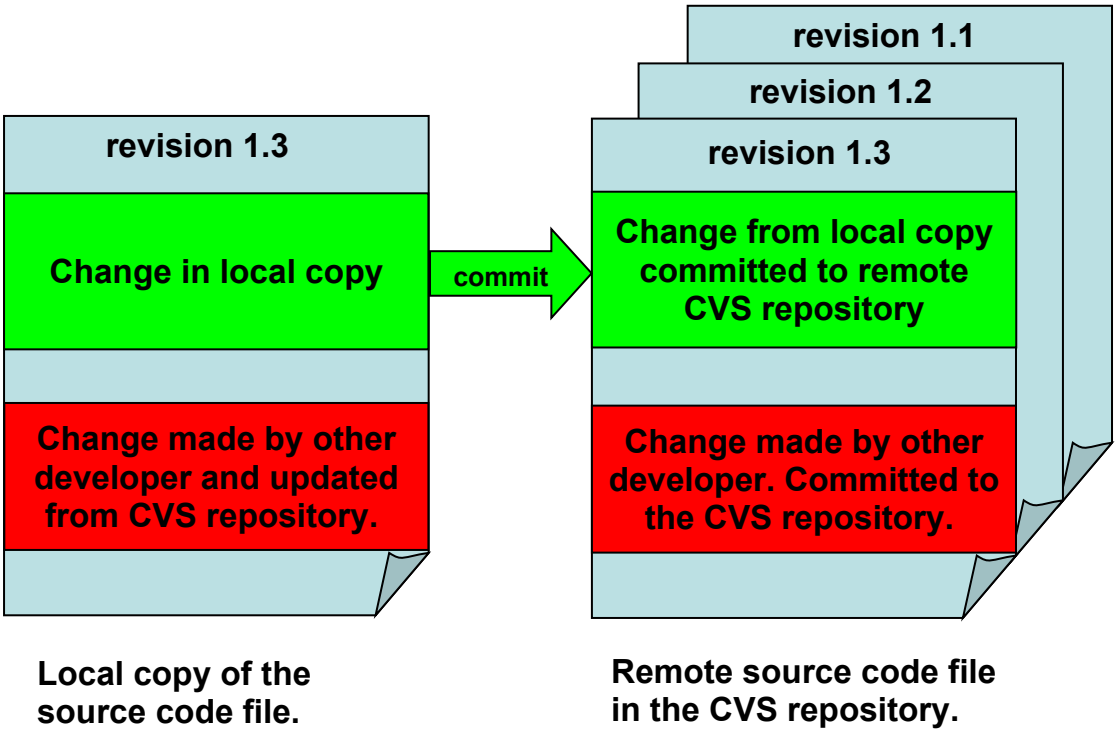
A **commit** merges the changes that were made in your local copy of the source code into the remote CVS repository. The commit process first compares the local revision number of each affected file with the latest revision number of the corresponding file in the remote CVS repository. If the revision numbers are the same, the commit process proceeds. Otherwise, an update process is performed first. This sequence ensures that commits are only performed on files that have the same revision numbers.

The diagram below shows the different states of a source code file in the local copy (left) of the source code and in the remote CVS repository (right), before a CVS commit. Note that the local revision number is the same as the latest revision number in the remote CVS repository. We assume that an update has already been performed that merged the changes made by another developer (red box) into your local copy of the file. In the exceptional case that you are the only developer committing changes to a remote CVS repository, you will never have to perform updates, but only commits.



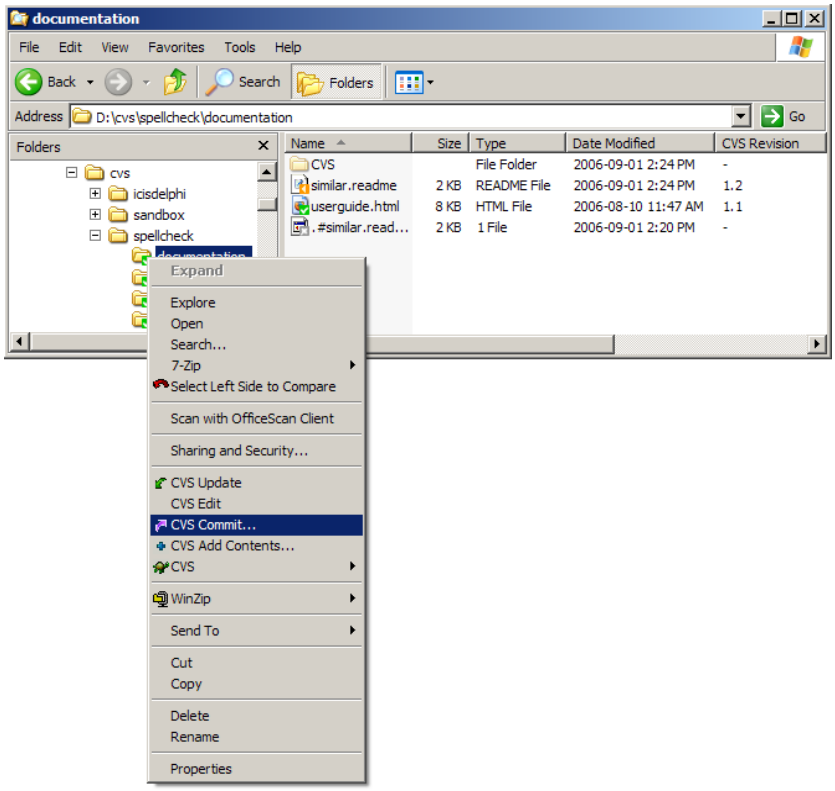
After the commit process has verified that your local revision number is the same as the latest revision number of the corresponding file in the remote CVS repository, your local changes (green box) are merged into the corresponding file in the remote CVS repository. The commit process causes

an increment in the revision numbers in your local copy as well as in the remote CVS repository. After a successful commit, the remote copy is the same as your local copy. Other developers will now have to update their local versions before being able to commit their latest changes.

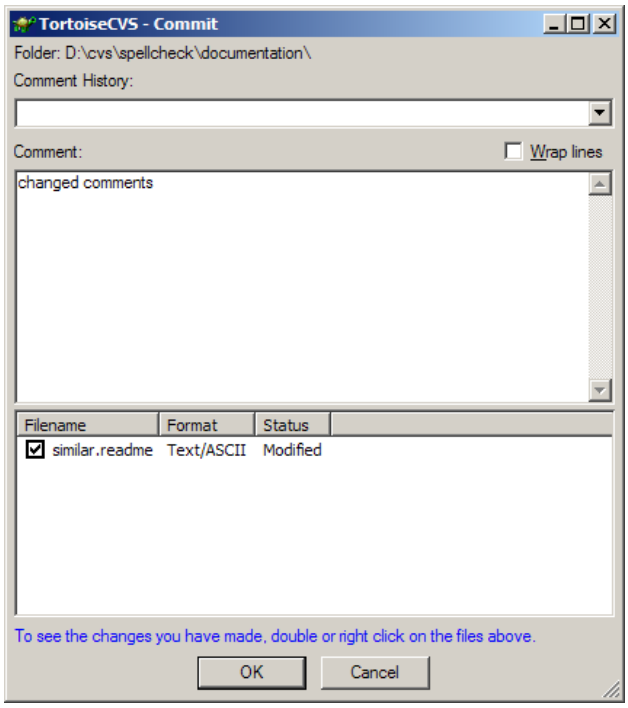


The following screenshots show the CVS commit process as done in TortoiseCVS.

The module **documentation** in the project **spellcheck** contains a changed file (similar.readme) in the local copy. This change has not been committed to the remote CVS repository, as indicated by the icon overlay. In this example, we have selected the module directory to be committed, not only the file that we know has changed locally. If other files in the module had been changed as well, they would be committed at the same time. We assume that an update process as described in the previous chapter has already been done for the entire module, and that the local revision numbers of all files in the module are the same as the latest revision numbers of the corresponding files in the remote CVS repository.

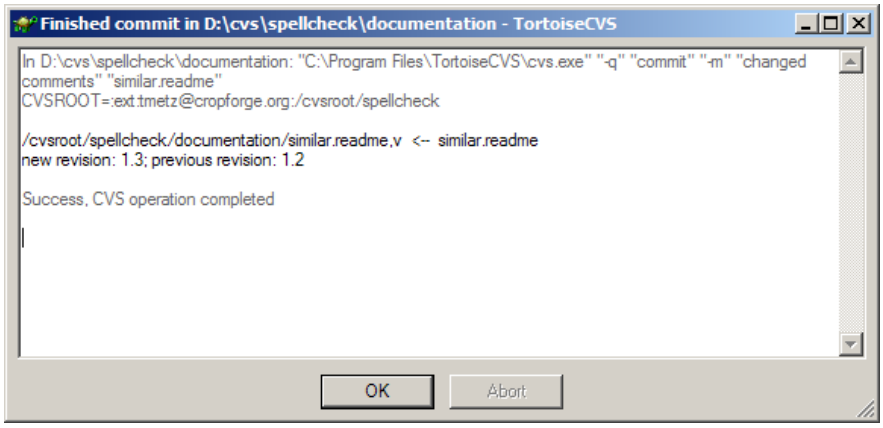


The commit process displays a window with all files that will be affected by the commit process, as well as their current local status. At this stage you can de-select files from the list, verify that the file formats are recognized properly (Text/ASCII, binary), and add a comment to the commit process. This comment will be attached to each new file revision generated by this commit process on the remote CVS repository.

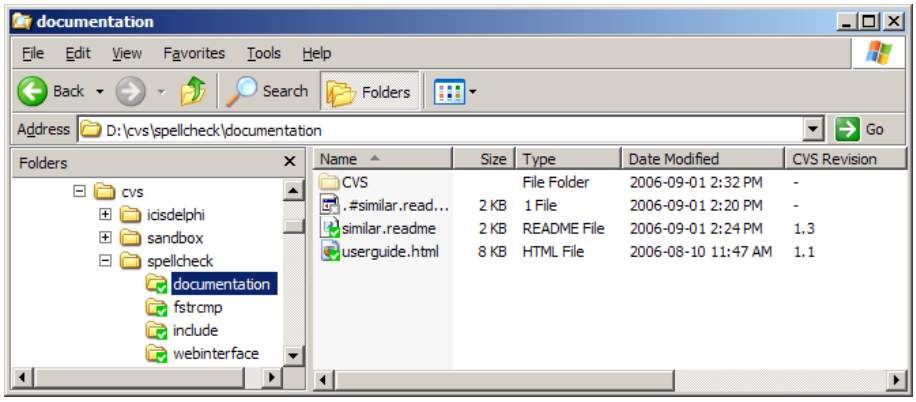


Subsequently, the commit process generates a message window that shows the details of the communication between the CVS client and CVS server, as well as the actions performed.

According to the message, the new revision 1.3 was generated in the remote CVS repository, as well as in the local copy.



The local copy now shows an overlay icon indicating that the local file **similar.readme** is in sync with the copy in the remote CVS repository. Note that the version number has changed to 1.3 after the successful commit process.

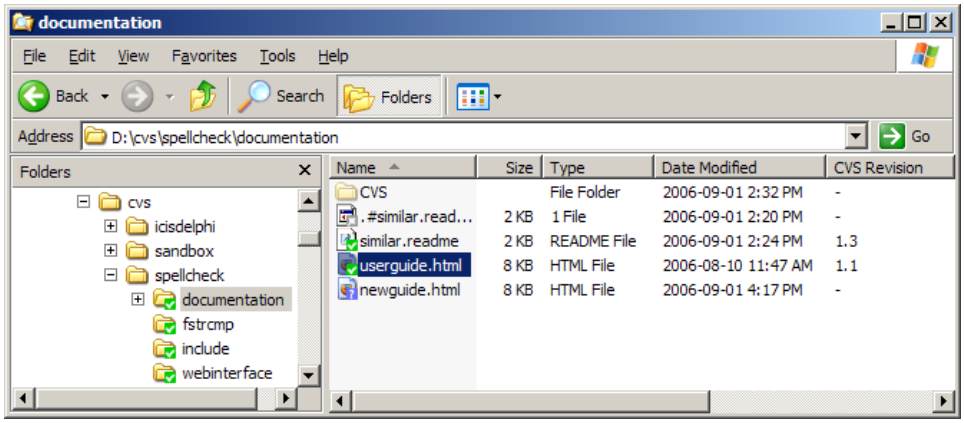


At this stage, we have incorporated our local changes into the remote CVS repository. These changes are now visible through the CVS web interface of the CropForge server, and they are accessible to other developers through an update process of their local copy of the source code.

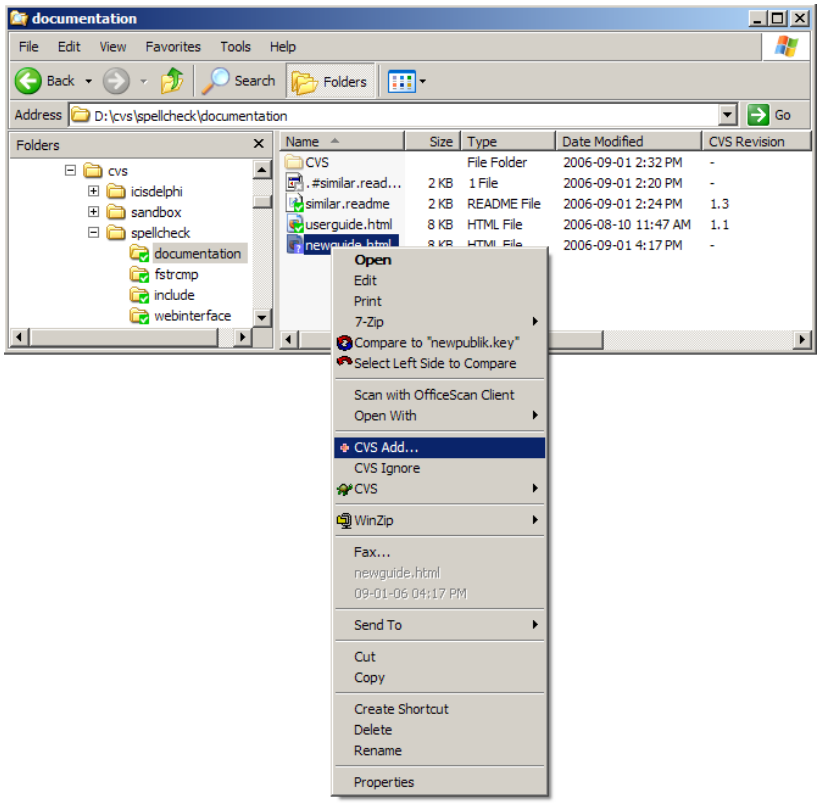
6. Adding a file or directory to the source code

Adding a file or a directory to the remote CVS repository is a three-step process. The first step is to create or copy a file or directory into the local module directory that is under CVS control. Then the file or directory has to be added to the local CVS system maintained by TortoiseCVS. Finally, the file or directory has to be committed to the remote CVS repository.

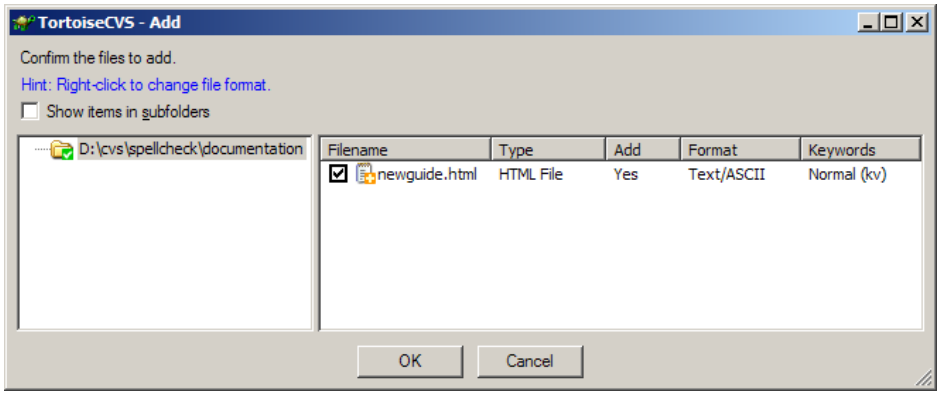
In the following example, a file (newguide.html) has been created in the local **document** module of the **spellcheck** project. The overlay icon indicates that the file has not yet been added to the local copy of the source code under CVS.



We now perform a CVS Add for the file **newguide.html**.

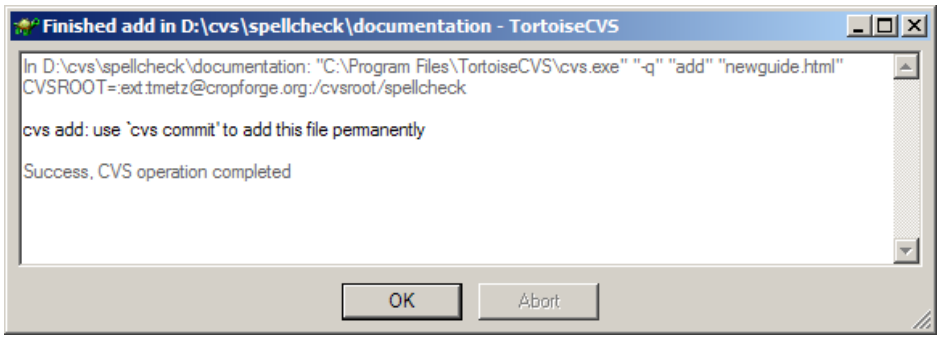


The **add** process displays a window with all files that will be affected by the add process, as well as their current local status. At this stage, you can de-select files from the list and verify that the file formats are recognized properly (Text/ASCII, binary). The file format can be changed by a right-click on the format value of a specific file.

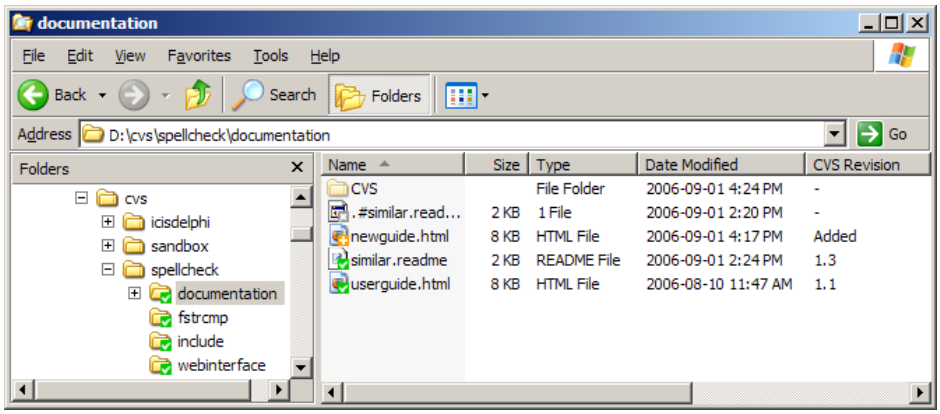


Subsequently, the **add** process generates a message window that shows the details of the communication between the CVS client and CVS server, as well as the actions performed.

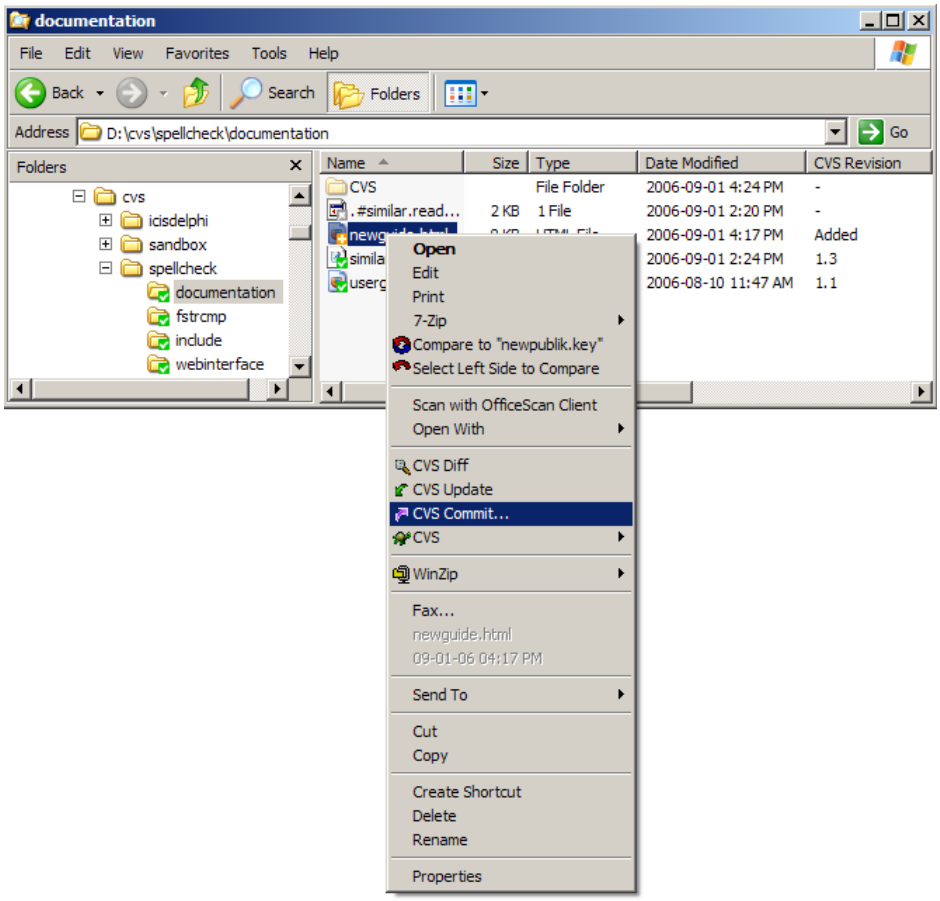
Note the message that alerts the user to perform a **commit** in order to add the file permanently to the remote CVS repository.



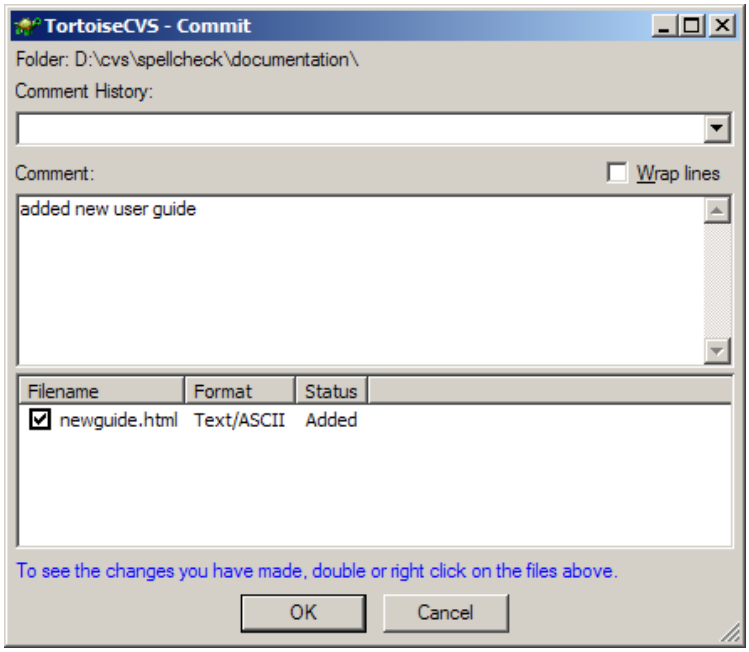
Now the overlay icon indicates that the file has been added to the local copy, but it does not have a revision number yet.



We now perform a **commit** for the file (newguide.html) that was added to the local copy of the **documentation** module.

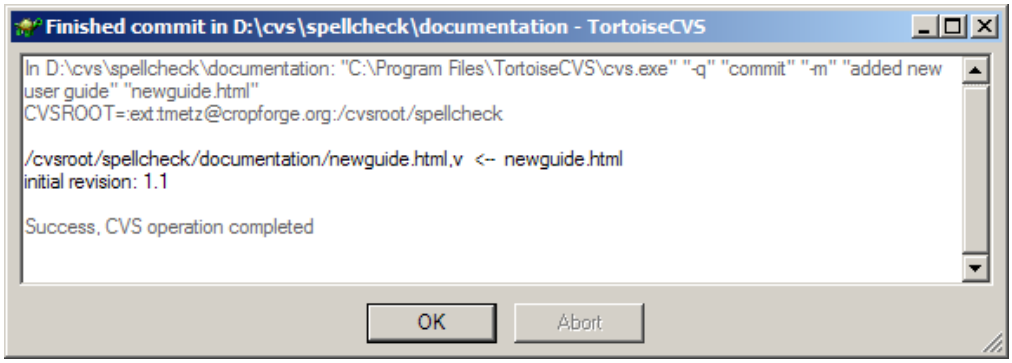


The **commit** process displays a window with all files that will be affected by the commit process, as well as their current local status. At this stage, you can de-select files from the list, verify that the file formats are recognized properly (Text/ASCII, binary), and add a comment to the commit process. This comment will be attached to the initial file revision on the remote CVS repository.

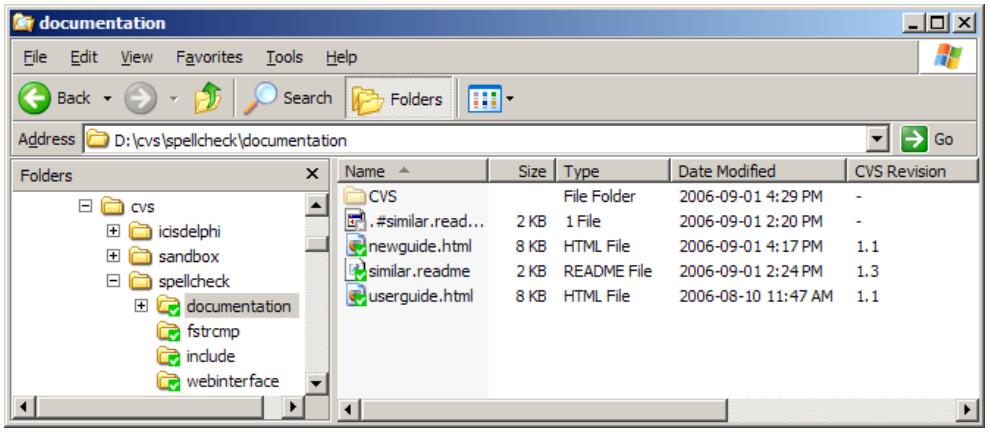


Subsequently, the **commit** process generates a message window that shows the details of the communication between the CVS client and CVS server, as well as the actions performed.

According to the message, the initial revision 1.1 was generated in the remote CVS repository, as well as in the local copy.



This is now reflected in the local copy, which shows that the file **newguide.html** has the revision 1.1 and according to the overlay icon is in sync with the remote CVS repository.



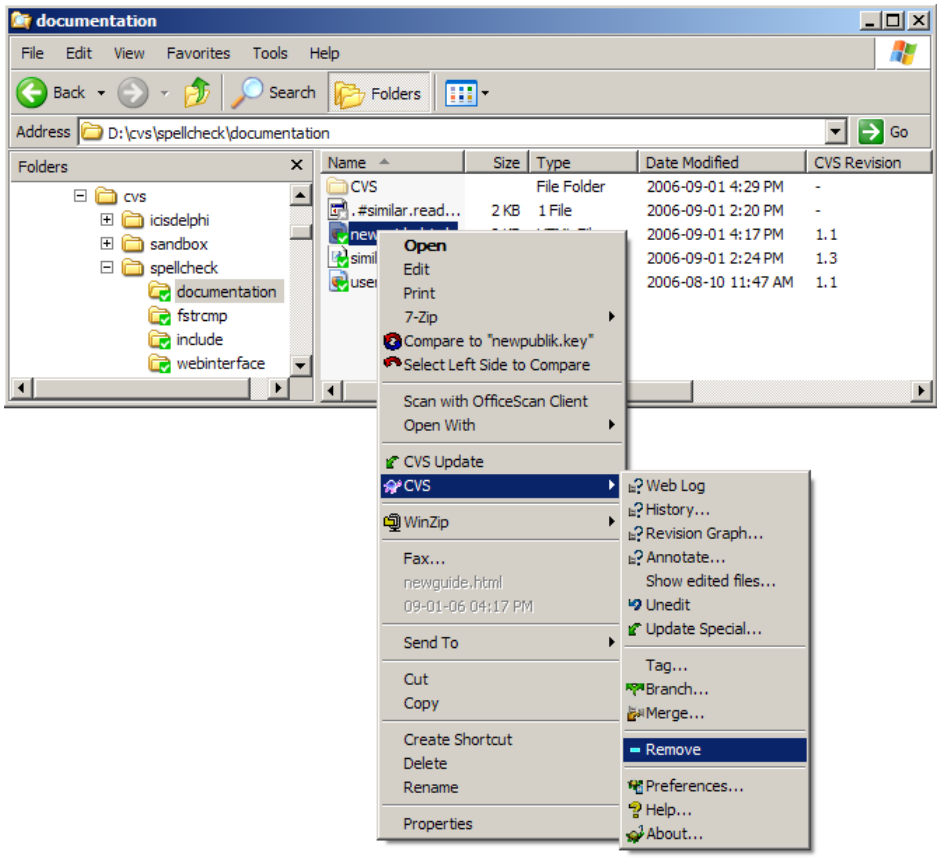
At this stage, we have incorporated a new file into our local copy as well as into the remote CVS repository. These changes are now visible through the CVS web interface of the CropForge server, and they are accessible to other developers through an update process of their local copy of the source code. When another developer performs an update to his/her local copy of the **documentation** module of the **spellcheck** project, the file **newguide.html**, revision 1.1, will be added to the local copy.

The process as shown above for a single file works equally for several files or entire directories and their respective content.

7. Deleting a file or directory from the source code

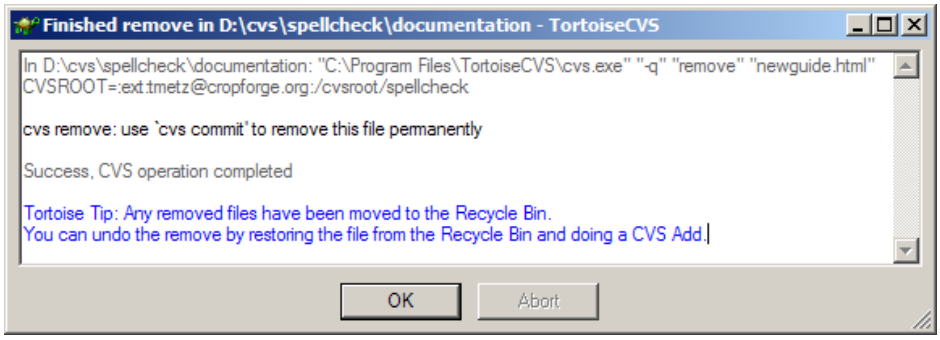
Deleting a file or a directory from the remote CVS repository is a two-step process. The first step is to remove the copy of a file or directory from the local module directory and at the same time from the local CVS control. Then, the deletion has to be committed to the remote CVS repository. Note: Adding a file is a three-step process, deleting a file is a two-step process.

In the following example, a file (newguide.html) is removed from the local **document** module of the **spellcheck** project. Initially, the overlay icon indicates that the file is in sync with the remote CVS repository.

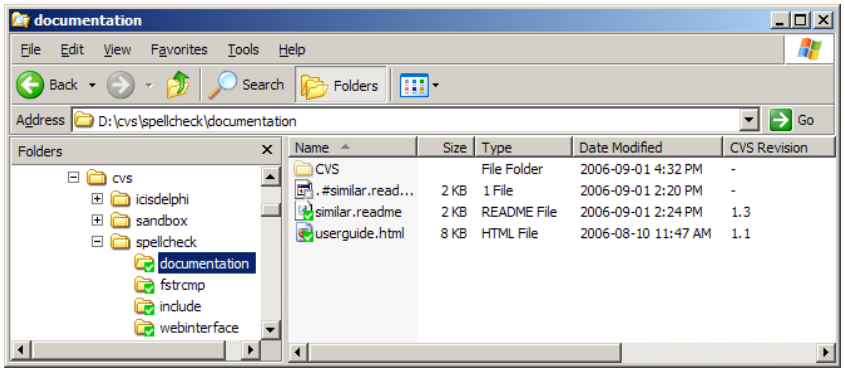


The **Remove** option in the **CVS submenu** of the Explorer removes the file from the local directory and moves it into the Recycle Bin, and at the same time records the removal in the local CVS administration information.

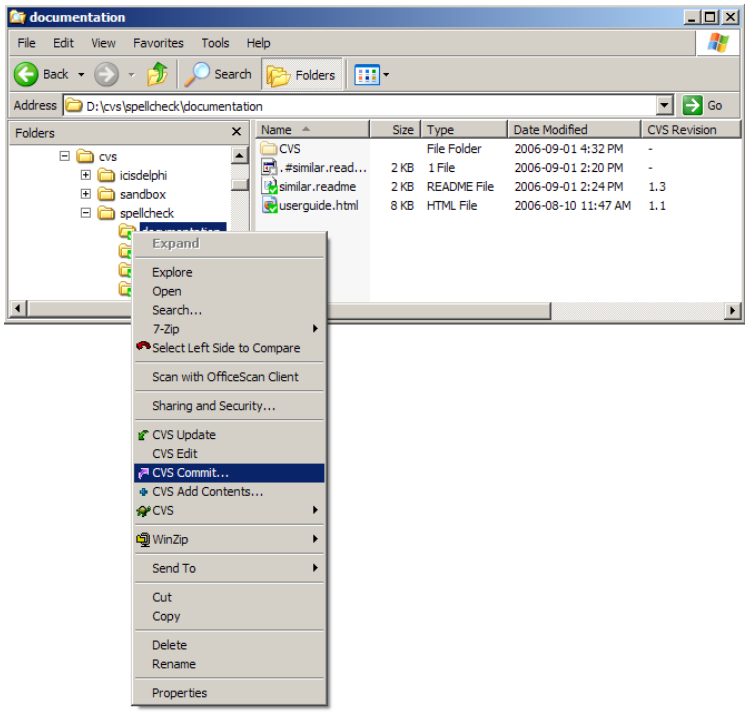
Note: A simple file deletion from the Explorer is not the same, as it will change the local CVS administration information.



After the removal, the deleted file (newguide.html) no longer appears in the Explorer file display.

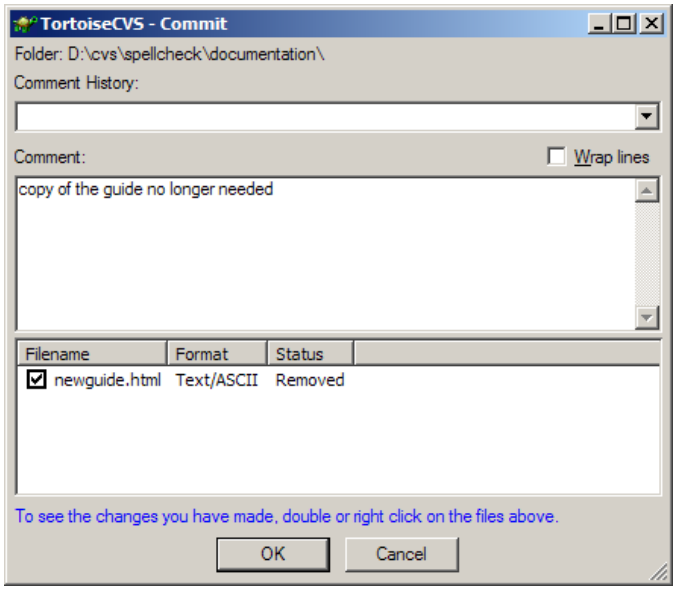


In order to commit the file deletion, we need to go to the higher level directory (documentation), and perform a commit on the entire directory.



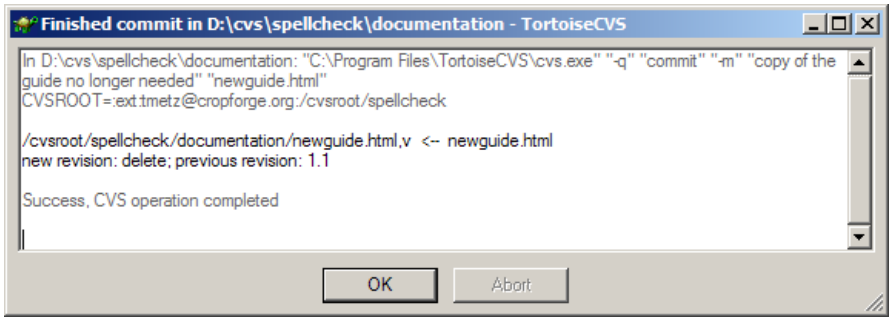
The **commit** process displays a window with all files that will be affected by the commit process, as well as their current local status. At this stage, you can

de-select files from the list and add a comment to the commit process. This comment will be attached to the initial file revision on the remote CVS repository.



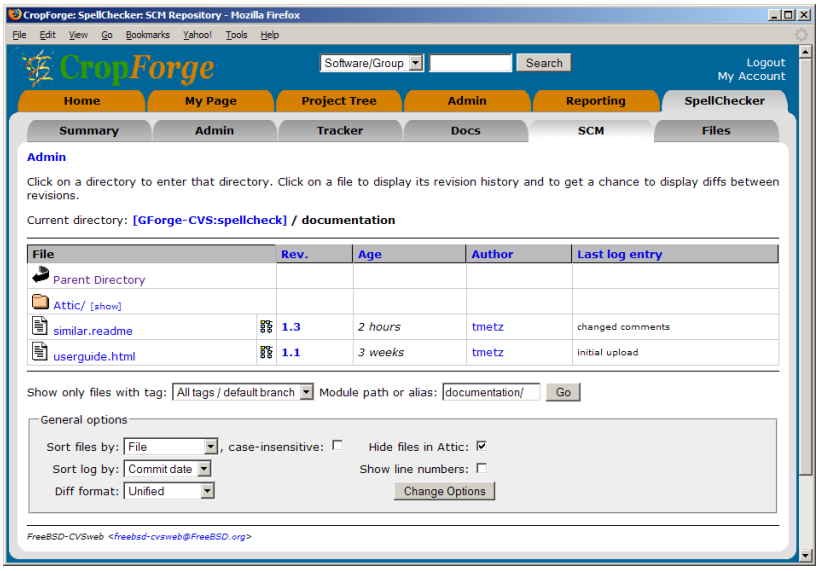
Subsequently, the **commit** process generates a message window that shows the details of the communication between the CVS client and CVS server, as well as the actions performed.

According to the message, a new revision was generated, and the revision 1.1 was deleted in the remote CVS repository.

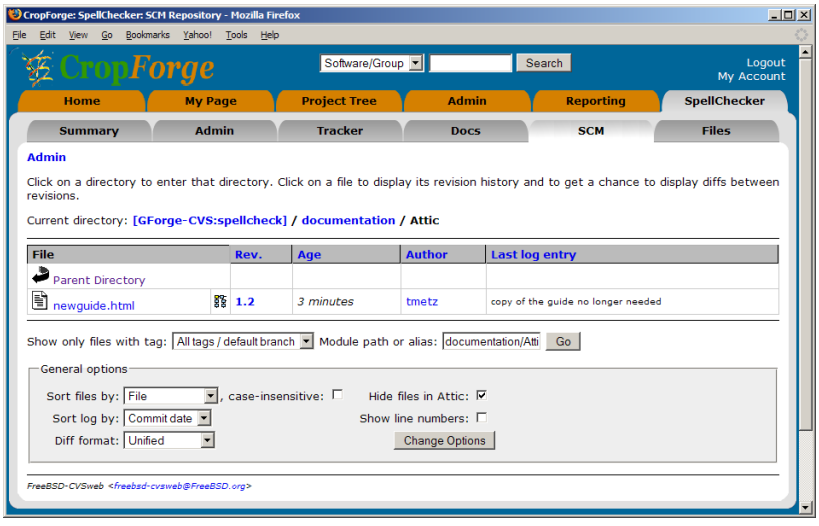


When another user now performs an update of his/her local documentation module against the remote CVS repository, the file **newguide.html** will be removed from his/her local copy of the source code.

Using the CVS web interface on CropForge, we can verify that the file newguide.html has been removed from the documentation module on the remote CVS repository.



However, the file **newguide.html** has not disappeared completely. A special directory has been created on the remote CVS server called **Attic**, that holds the deleted file. As we can see, the delete process has resulted in a new revision (1.2) of **newguide.html**, and a move of that file into the Attic subdirectory of the documentation module directory. As we will see in a later section, CVS needs to be able to restore previous versions of the complete source code, and for that reason deleted files cannot be removed permanently.



At this stage, we have removed a file from our local copy as well as from the remote CVS repository. These changes are now visible through the CVS web interface of the CropForge server, and they are accessible to other developers through an update process of their local copy of the source code. When another developer performs an update to his/her local copy of the documentation module of the **spellcheck** project, the file **newguide.html** will be deleted from his/her local copy.

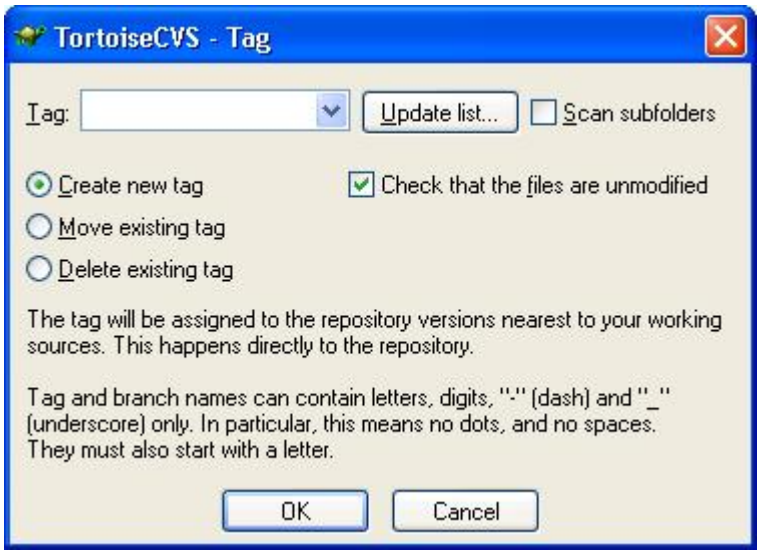
Deleted files and their complete revision history are still retained in the remote CVS repository, so that they can be restored if necessary.

The process as shown above for a single file works equally for several files or entire directories and their respective content.

8. Tagging source code

Giving a common label to one or more files (typically on entire modules) is referred to as "tagging," and this is used to mark a version of these files for easier retrieval. Basically, the assigned tag refers to the revision(s) implemented on these files. It is recommended that tagging be performed on project deliverables, and before making major changes to files.

To create a tag, select the file or directory for tagging, click the right mouse button, then choose **CVS -> Tag**. A TortoiseCVS window (see example below) will be displayed.

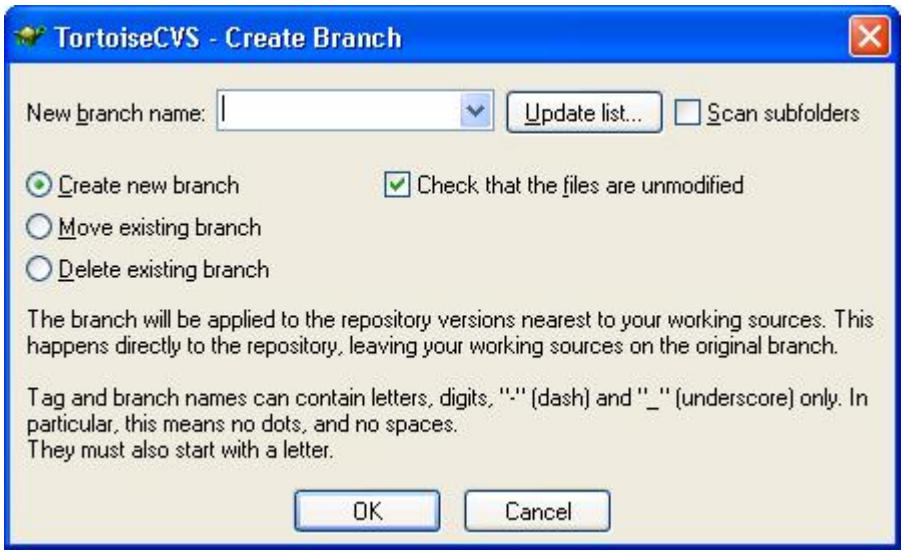


Proceed by selecting **Create new tag** radio button and enabling **Check that the files are unmodified**, then type in a label name in the **Tag** input field. Note that there are restrictions in the characters that a tag may contain (see Section **14: Naming conventions**). When you are finished, click the **OK** button.

9. Branching source code

"Branching" in CVS refers to the process of isolating changes into a separate line or branch of development. This allows changes to files within a branch without affecting corresponding files in the main line of development or in other branches.

To create a branch, select the files or directory to create a branch from, click the right mouse button, then choose **CVS -> Branch**. A TortoiseCVS window, as shown in the example below, will be displayed.

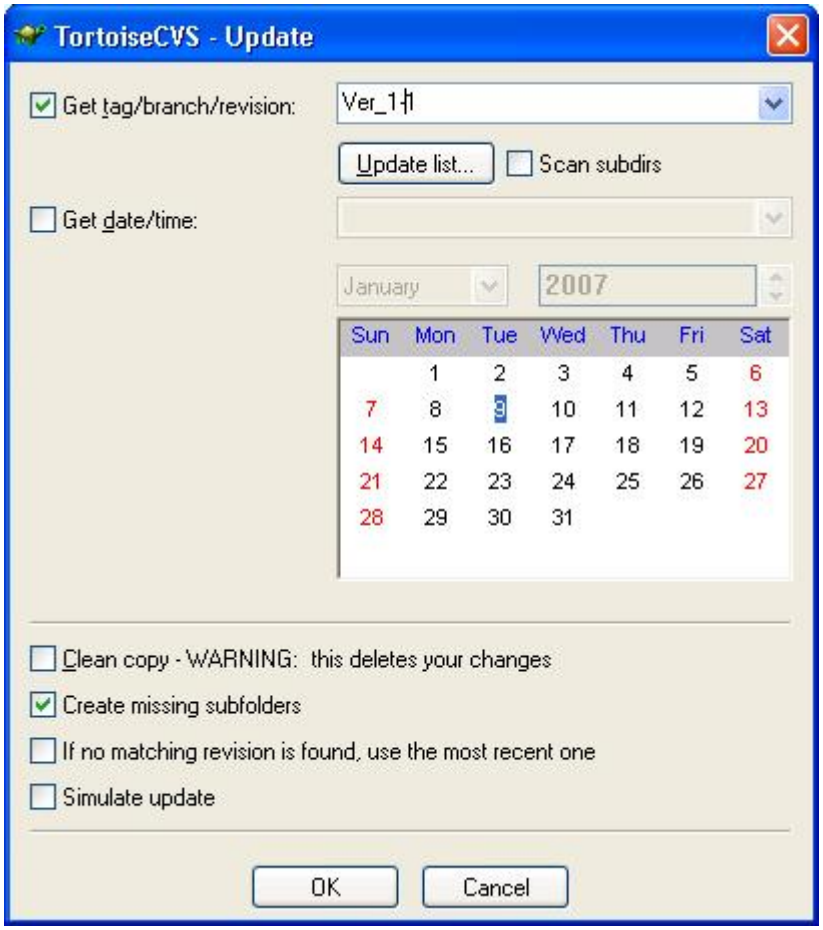


Proceed by selecting **Create new branch** radio button, and enabling **Check that the files are unmodified**, then type in a label name in the **New branch name** input field. Take note that there are restrictions in the characters that a branch name may contain (see **Section 14: Naming conventions**). When you are finished, click the **OK** button.

Note: The branch is created in the remote CVS repository only. Your local copy must be tied to the CVS repository branch you wish to work on. For information on how to select and access a branch, see **Section 10: Switching between branches**.

10. Switching between source code branches

To access and edit files on a branch (instead of the main development line), your local copy (working folder) must be bound to the branch. To bind your local copy to a branch, select a directory in your project (the top-level directory is recommended), click the right mouse button, then choose **CVS -> Update special**. A TortoiseCVS window, as shown in the example below, will be displayed.

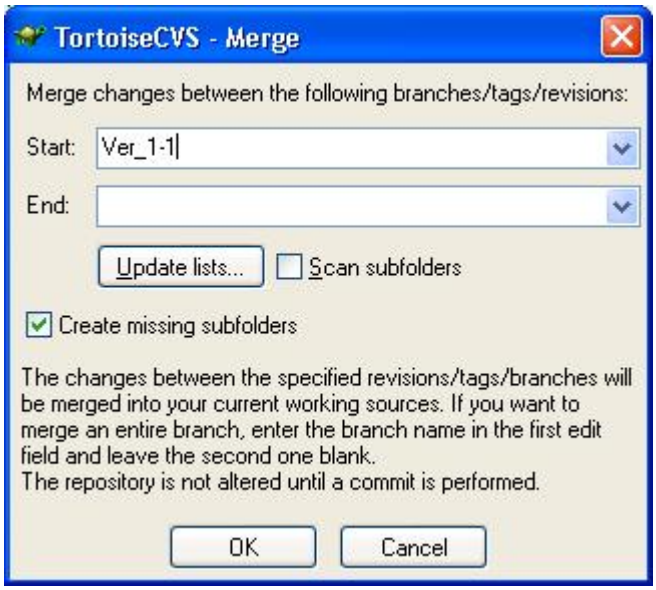


Proceed by checking **Get tag/branch/revision**, then type in the branch name. If you wish to stop working on a branch and move your working folder back to the main line of development, specify "HEAD" as the branch/revision name. When you are finished, click the **OK** button.

11. Merging changes from a branch into main development

Changes done on a branch may be made available on another branch or on the main line of development through a process known as "merging".

To perform a merge, move/bind your local copy to the branch where you want to merge your changes (see **Section 10: Switching between branches**), then select a directory in your project (the top-level directory is recommended). After making a selection, click the right mouse button and choose **CVS -> Merge**. A TortoiseCVS window, as shown in the example below, will be displayed.



Proceed by typing in the tag/branch name of the branch to merge from. Enter a name in the **Start** input field and leave the **End** input field blank to merge an entire branch. When you are finished, click the **OK** button.

Note: The steps specified above will merge changes from the start of the branch, so it is highly recommended to assign a new tag to a branch after every merge. The new tag would then be used to name a branch for subsequent merges. Keep in mind that a conflict may result from a merge operation. If that happens, the conflict must be resolved before committing the new revision (see **Section 13: Recognizing and resolving edit conflicts**). Also note that the merging process will only take effect on the CVS repository after performing a **commit**.

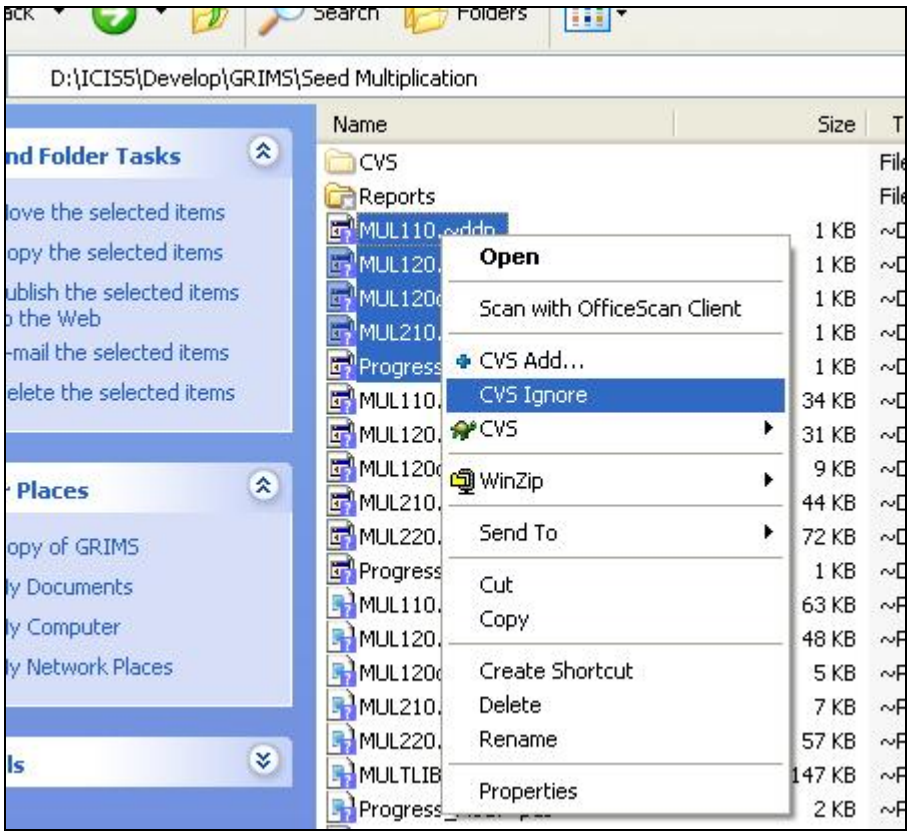
12. Determining which files to put under version control

As the rule-of-thumb, certain file types (usually binary) should not be placed under version control. In other words, we want the CVS server to ignore these file types whenever operations such as **Add** and **Commit** are performed on the CVS repository.

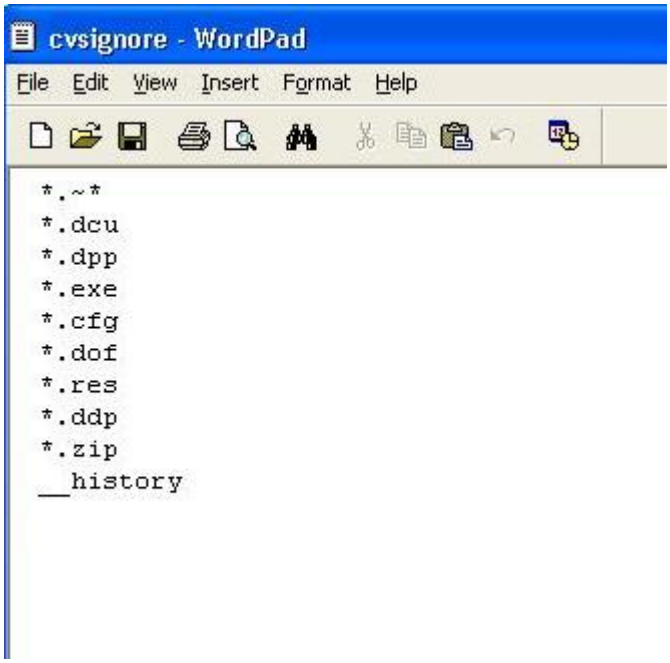
TortoiseCVS tries to automatically detect whether a file is binary or text when you perform an **Add** operation on the CVS repository. Files with extensions such as .doc and .exe are assumed to be binary, whereas other extensions such as .cpp and .txt are assumed to be text.

The CVS server can be told which file types to ignore through a **.cvsignore** file, which is basically a text file containing a list of files that you do not want to include in CVS operations. The **.cvsignore** file contains one pattern per line, which can be interpreted as file and/or directory names.

To create a **.cvsignore** file, select the files and/or directories that you want to ignore, then click the right mouse button. In the pop-up menu, select **CVS Ignore.** The **.cvsignore** file is automatically added to the current directory, and the file now contains the selected filenames that will be ignored.



It is possible to ignore certain filenames, or even a group of files with a common file extension. For example, you want to ignore files with the following file extensions: `*.~*`, `*.dcu`, `*.dpp`, `*.exe`, `*.cfg`, `*.dof`, `*.res`, `*.ddp`, and `*.zip`. Also, entire directory names can also be ignored. Just edit the **.cvsignore** file, then it would look something like this:



The **.cvsignore** file will be effective once it has been added and committed to every directory under CVS control. Once the **.cvsignore** file has been added to the CVS repository, it can be shared with other developers. The **.cvsignore** file works under WinCVS as well as TortoiseCVS.

13. Recognizing and resolving edit conflicts

A conflict occurs when two or more developers have changed the same few lines of a file. The CVS server will report any conflicts, which have to be resolved before performing further CVS operations. The CVS server has the ability to identify conflicts but not to automatically resolve them; therefore any conflict has to be resolved manually by the developer by editing of affected files. Once a file has been opened, locate the area with the conflict. This area begins and ends with the "**<<<<<<<**" and "**>>>>>>>**" markers, respectively. The area with a conflict looks like this:

```
<<<<<<< filename
    code with changes done in your local copy/working folder
=======
    code merged from CVS repository
>>>>>>> revision/version number
```

The next step is to remove the conflict "markers" (along with everything on the same line as the markers) and the line(s) of code you want to discard. You would have to decide whether to retain the code from your local copy or the code merged from the CVS repository. When a conflict occurs, a developer should consult with the other developer(s) involved (if applicable) and work out the correct solution together, rather than blindly overwriting one or the other revision. After saving the file in your local copy, perform a **commit** to save changes in the corresponding file in the CVS repository.

15. Resources

- CropForge server: http://cropforge.org
- TortoiseCVS software: http://www.tortoisecvs.org
- TortoiseCVS help: Start -> Programs -> TortoiseCVS -> Help
- PuTTY software: http://www.chiark.greenend.org.uk/~sgtatham/putty/
- CVS book: http://cvsbook.red-bean.com
- CVS manual: http://ximbiot.com/cvs/manual/