WinCvs -- Daily Use Guide

Or: How to use WinCvs while keeping your mental health.

Table of Complaints

Introduction

This document describes day to day usage of the WinCvs 1.0.x client. It is *not* an introduction to version control systems, *not* an introduction to CVS, and *not* an introduction to WinCvs. It is more like a place you may turn to when you know approximately what you want to do, but don't quite remember how to do it.

It is assumed that you already have installed WinCvs, and that you have configured it to talk to the CVS repository you want to use.

If your browser supports style sheets, you will find a couple of different typesetting and color variations throughout the document. *Things looking like this* refer to elements in the WinCvs user interface, such as buttons, tabs and input field labels. References to the original CVS documentation

Adding Files and Directories

When you create new files that you want to include in the repository, you must tell CVS to handle the files. If the directory containing the files is not under CVS control, you will have to add it before adding the files. To add files or directories, do this:

1. Select the directory, file or files that you want to add.
2. Click the right mouse button on the selection, and choose the *Add selection* or *Add selection binary* menu item. Use binary for non-text files, otherwise the files will be corrupted by CVS!
3. As the files are only *marked* for addition, you have to commit them to enter them in the repository. See Sending Your Changes to CVS if you don't know how to commit.

If you want to add entire directory hierarchies instead of a few files, the above technique becomes cumbersome as the *add* operation is not recursive. In this case you should rather use *import*, as in Starting a New Module:

1. Choose the *Cvs Admin -> Import module* menu.
2. In the file dialog that pops up, select the directory you want to add by making its folder icon open.
3. WinCvs will now try to identify any binary files that you may have in your directory tree. If a filter window shows up, make sure the files mentioned have the correct text/binary setting.
4. In the *Import settings* dialog, type the module name with the directory appended. If the module is called `MyProject`, and you are importing the subdirectory `source/utils`, you enter `MyProject/source/utils` in the *Select the module name...* input field.
5. Type your name, or your company name (no spaces) in the *Vendor tag* field.
6. Type `start` in the *Release tag* field.
7. Press the *OK* button.

WinCvs will now import your entire directory hierarchy to the CVS server, under the given module. Note that nothing will be changed in your local files

are given like this.

Terminology

The terminology used in the CVS documentation, and thus also in WinCvs, may differ from terminology used in other source repository systems. In an attempt to avoid confusion, we provide a short list of the most essential terms. Please make yourself familiar with these words before diving into the rest of this document.

Checkout
: Normally used to describe the first retrieval of an entire module from the repository.

Commit
: Sending your modifications to the repository.

Export
: Refers to extraction of an entire module from the repository, without any CVS administrative files: Exported modules will not be under CVS control.

Import
: Normally refers to the process of creating a new module in the repository by sending an entire directory structure.

Module
: A directory hierarchy. A software project normally exists as a single module in the repository.

Release
: The version of an entire product.

Revision
: The version of a single file.

Tag
: A symbolic name given to a set of files at a specific time of development.

Update
: Get other users' modifications from the repository. Updates the local copy only.

For the entire story of "revisions", "releases", "versions" and "tags", see chapter 4 of the CVS documentation.

Starting a New Module

Getting a project under version control is called *importing* in CVS terminology. As the name implies, you should have something for the CVS server to import *before* thinking version control.

during the import operation. This means that your local copy will not be under version control after the import. To have the newly imported hierarchy under version control, obtain a copy of it from the repository:

1. Move your original hierarchy away, eg. by renaming the directory you just imported to `*.old` using Windows' file explorer.
2. In WinCvs, select the directory *above* the one you just added. Right click on it, and choose *Update selection*.
3. Make sure *Create missing directories that exist in the repository* is checked.
4. Press the *OK* button.

WinCvs will now give you a fresh, version controlled copy of your hierarchy.

[CVS Doc: "Adding, removing, and renaming files and directories"]

Removing Files and Directories

To remove files, you first schedule the files for removal, and then commit the change:

1. Select the file or files that you want to remove.
2. Click the right mouse button on the selection, and choose the *Remove selection* menu item.
3. As the files are only *marked* for removal, you have to commit them to remove them from the repository. See Sending Your Changes to CVS if you don't know how to commit.

The files will now be removed from the repository. Note that files are not physically removed, but rather marked as "dead". By this way it will still be possible to retrieve the files if you choose to check out an old version of the module.

Removing directories is another story. CVS will optionally remove empty directories when you *update* one of its parent directories. If you want to get rid of an empty directory, do the following:

1. Select the parent directory of the empty directory you want to remove.
2. Click the right mouse button on the selection, and choose the *Update selection* menu item.
3. Select the *Globals* tab.
4. Make sure *Prune (remove) empty directories* is checked (it is by default).
5. Press the *OK* button.

CVS lacks good support for restructuring a project file hierarchy, so you save yourself a lot of trouble if you spend some time planning before importing a new module.

The basis for the import operation is a "clean" directory structure. By clean we mean that files which need no versioning (compiler generated files, backups etc.) are removed. This is particularly important if your project has been going on for a while. You may have several files in your hierarchy that you do not want to put under version control, but that you want to keep where they are anyway. In that case you have to move them out before importing, and move them back afterwards.

Be aware that CVS treats empty directories as non-existent. If you want to add a directory in which you have neither files nor subdirectories, you will have to create a dummy file in it. We suggest that you create a file named `README.txt`, which contains a short description of what this directory is for.

Once you have a directory structure that contains only files you want to have in the repository, do the following to import the module using WinCvs:

1. Choose the *Cvs Admin -> Import module* menu.
2. In the file dialog that pops up, select the top level directory of the project you want to import by making its folder icon open.
3. WinCvs will now try to identify any binary files that you may have in your directory tree. If a filter window shows up, make sure the files mentioned have the correct text/binary setting.
4. In the *Import settings* dialog, type a module name. This will be the name of the top level directory for the module when people check it out.
5. Type your name, or your company name (no spaces) in the *Vendor tag* field.
6. Type `start` in the *Release tag* field.
7. Press the *OK* button.

WinCvs will now import your entire directory hierarchy to the CVS server. Note that nothing will be changed in your local files during the import operation. This means that your local copy will not be under version control after the import. Before starting to work on your version controlled sources, you have to do the following:

1. Move your original hierarchy away, eg. by renaming the top level directory of the project to `*.old` using Windows' file explorer.
2. Check out a copy of the new module by following the instructions in the section Getting a New Module from CVS.

The directory will be removed if all its files were previously removed from your local copy and from the repository.

[CVS Doc: "Adding, removing, and renaming files and directories"]

Moving or Renaming Files and Directories

Operation of moving or renaming files or directories is not available in CVS. This is one of CVS's shortcomings. To simulate moving or renaming, you have to combine remove and add operations. See Adding Files and Directories and Removing Files and Directories.

[CVS Doc: "Moving and renaming files"]

Branching

One of the features of version control systems, is the ability to isolate changes onto a separate line of development. This line is known as a *branch*. (See What branches are good for in the CVS documentation.)

To create a branch, do the following:

1. Select the directory or files that you want to branch.
2. Click the right mouse button on the selection, and choose the *Tag selection -> Create a branch* menu item.
3. In the *New branch name* input field, enter a tag name that you want to use on your branch. See Tagging Files with a Label to learn what characters are valid in a tag.
4. Turn on the *Check that the files are unmodified before branching* checkbox.
5. Press the *OK* button.

Now a new branch with the given name is created in the repository. **Note:** The branch is created *in the repository only*. To start working on the newly created branch, you have to do what is described in Selecting a Branch to Work on.

[CVS Doc: "Branching and merging"]

Selecting a Branch to Work on

To start working on a branch instead of the default development line, you have to bind your local copy to the branch. This is needed to make sure that actions such as updates, commits etc. work on the branch rather than on the main line of development.

3. Optionally copy back the "unimportant" files that you moved out.

You should now be able to start working on your version controlled project.

[CVS Doc: "Starting a project with CVS"]

Getting a New Module from CVS

To obtain a module from the CVS server for the first time, is known as a *checkout*. Checking a module out from the repository gives you a local copy of the directory hierarchy that makes up the module. To perform a checkout, do the following:

1. Choose the *Cvs Admin -> Checkout module* menu.
2. In the file dialog that pops up, choose the directory in which you want the module to be located. A directory named after the module will be created in the directory you supply, so you will normally want to provide the directory in which you keep all your projects.
3. A checkout settings dialog appears. In the *Enter the module name...* field, enter the module name. Note that the module name is case sensitive.
4. Select the *Globals* tab.
5. Set the *Checkout read-only* according to what the project leader tells you (see below).
6. Press the *OK* button.

If you state that *Checkout read-only* should be in use, you will have to tell WinCvs which files you intend to edit before editing them. This may be cumbersome, but it enables other developers to track who is currently editing given files. See the section Who's Editing that File? for more information.

[CVS Doc: "checkout -- Check out sources for editing"]

Getting Other People's Changes from CVS

Occasionally you may want changes done by others to get incorporated in your local working copy. The process of getting changes from the server to your local copy is known as *updating*. Updating may be done on single files, a set of selected files, or recursively on entire directory hierarchies. To update, do the following:

1. Select the directory, file or files that you want to update.
2. Click the right mouse button on the selection, and choose the *Update selection* menu item.

To move your local copy to another branch, do the following:

1. Select the top level directory of the project. (If you know exactly what directories and files are part of the branch, you may select these instead.)
2. Click the right mouse button on the selection, and choose the *Update selection* menu item.
3. Make sure *Create missing directories that exist in the repository* is checked.
4. Select the *Sticky options* tab.
5. Click the *Retrieve rev./tag/branch* checkbox.
6. In the *Retrieve rev./tag/branch* input field, enter the tag name of the branch you want to switch to.
7. Press the *OK* button.

WinCvs will now do the necessary updates to your working copy and move it to the desired branch. The updating may include adding or removing files.

CVS puts what is known as *sticky tags* on the files that are affected by the branch. You may view these sticky tags by issuing a *status* command on the files, as described in Viewing Changes. To remove the sticky tags and thus go back to the main development line, follow the description in Going Back to the Main Line of Development.

[CVS Doc: "Accessing branches"]

Merging from a Branch

When you are satisfied with the changes you have done on a branch, you may want those changes to be available on the main line of development. Incorporating changes from one branch to another, is known as *merging*. To merge from a branch, do the following:

1. Move your local copy to the branch you want to merge the changes into. See Selecting a Branch to Work on or Going Back to the Main Line of Development.
2. Select the top level directory of the project. (If you know exactly what directories and files are part of the branch, you may select these instead.)
3. Click the right mouse button on the selection, and choose the *Update selection* menu item.
4. Make sure *Create missing directories that exist in the repository* is checked.
5. Select the *Merge options* tab.
6. Click the *Only this rev./tag* radio button.

3. Make sure *Create missing directories that exist in the repository* is checked.
4. Press the *OK* button.

Changes done by others will be merged into your files, keeping any changes you may have done to the same files. The repository is *not* affected by an updating.

If you receive reports of conflicts during the update, please read the next section.

[CVS Doc: "Bringing a file up to date"]
[CVS Doc: "update -- Bring work tree in sync with repository"]

Resolving Conflicts

Once in a while, the CVS server will report a *conflict* when you update your files from the repository. A conflict occurs when two or more developers have changed the same few lines of a file. As CVS knows nothing of your project, it leaves resolving the conflicts to the developers. Whenever a conflict is reported, you should open the file in question, and search for lines starting with the string <<<<<<<. The conflicting area is marked like this:

```
<<<<<<< filename
    your changes
=======
    code merged from repository
>>>>>>> revision
```

You should decide what the code should look like, do the necessary changes, remove the CVS markup, and commit your modifications to the repository.

[CVS Doc: "Conflicts example"]

Sending Your Changes to CVS

Making local modifications available in the repository, is known as *committing* the changes.

1. Before committing, you should do an update to make sure there are no conflicts. See Getting Other People's Changes from CVS.
2. Select the directory, file or files that you want to commit.
3. Click the right mouse button on the selection, and choose the *Commit selection* menu item.
4. In the *Commit settings* dialog, enter a log message. This step is
7. In the *Only this rev./tag* input field, enter the tag name of the branch you want to merge from.
8. Press the *OK* button.

Any changes on the branch will now be merged into your local copy. You will probably also want to commit the merged files back to the repository, as described in Sending Your Changes to CVS.

Important note: The merge given above will try to merge changes from the start of the branch. If you do the operation a second time (to merge changes done to the branch after the last merge), merging from the start of the branch is not what you want, and it will most likely get you into trouble. To get around this problem, you should give the branch a new tag after every merge, and use the new tag when naming the branch for subsequent merges.

[CVS Doc: "Merging an entire branch"]

Going Back to the Main Line of Development

If you want to stop working on a branch and move your local copy back to the main line of development, you have to make WinCvs remove all sticky tags. To remove the sticky tags, and thus update your local copy to the main development line, do the following:

1. Select the top level directory of the project. (If you know exactly what directories and files are part of the branch, you may select these instead.)
2. Click the right mouse button on the selection, and choose the *Update selection* menu item.
3. Make sure *Create missing directories that exist in the repository* is checked.
4. Click the *Reset any sticky date/tag/'-k' options* checkbox.
5. Press the *OK* button.

WinCvs will now update your local copy so it matches the current main line of development. The branch you were on still exists in the repository, and you may return to it as described in Selecting a Branch to Work on whenever you want to.

[CVS Doc: "Sticky tags"]

Who's Editing that File?

It is possible to ask CVS who is currently editing a file, but it only works if the developers announce to CVS when they intend to edit a file. This model

optional, but it is highly recommended that you take some time to shortly describe what was changed.
5. Press the *OK* button.

Please note that committing changes will not automatically add new files that you have created to the repository. See Adding Files and Directories for a description of doing that.

[CVS Doc: "commit -- Check files into the repository"]

Viewing Changes

WinCvs may be used for viewing status, logs, diffs etc, of files and directories.

1. Select the directory, file or files that you want to know more about.
2. Click the right mouse button on the selection, and choose one of *Diff selection*, *Log selection*, *Status selection* or *Graph selection*.
3. If a dialog box pops up, enter any necessary information, and press the *OK* button.

Here is a short explanation on what output you might expect from the various status commands.

Diff
: Lets you compare your local copy of a file with any revision of the same file in the repository. It also lets you compare different revisions within the repository. The output consists of lines starting with < or >, symbolizing lines that should be removed or added respectively, to go from your revision to the one you compare against.

Log
: Shows log messages, dates, tags, authors, etc. for all revisions of the given files (unless you limit what revisions to display).

Status
: Displays the modification status of the given files, i.e. whether the files are modified locally or in the repository. It also shows both the local and the repository revision numbers, and tags, if any.

Graph
: A rather cool feature that shows the revision graph for a single file. Particularly useful when you have one or more branches in the file's revision history. The local revision is marked by a document icon. The graph lets you select two revisions (using the shift button for some reason) and do a diff between them.

[CVS Doc: "diff -- Show differences between revisions"]

is known as the *IR CVS model* (well, that's actually an internal joke at Computas), and it should be decided at project startup if this model is to be used.

To use the IR CVS model, developers should check out the module read only. It can be done by checking *Checkout read-only* in the *Globals* tab of every WinCvs dialog.

To edit a file, do the following:

1. Select the file you want to edit.
2. Click the right mouse button on the selection, and choose the *Monitors selection -> Edit selection* menu item. (Alternatively, use the shortcut in the toolbar.) The read-only flag will be removed, and the CVS server registers you as an editor of the given file.
3. Edit the file.
4. Commit the file as described in Sending Your Changes to CVS.

When you commit the file, the CVS server assumes you are no longer editing it, and WinCvs will make the file read only again. If you did not make any changes, a commit will not tell the CVS server that your have finished editing it. In that case you should rather do the following:

1. Select the file you marked for editing.
2. Click the right mouse button on the selection, and choose the *Monitors selection -> Unedit selection* menu item. (Alternatively, use the shortcut in the toolbar.)

And now, back to the original question: "Who's editing that file?" To list the editor(s) if any of a file, do this:

1. Select the file you are curious about.
2. Click the right mouse button on the selection, and choose the *Monitors selection -> Editors of selection* menu item.

WinCvs will now list the known editors of the file.

[CVS Doc: "Multiple developers"]

Appendix A: FAQ

1. *What does the mystical letters shown during update mean?*

   See update output in the CVS manual.
2. *The clock on the server is wrong! Id tags and logs show the wrong*

[CVS Doc: "log -- Print out log information for files"]

Tagging Files with a Label

At a given stage of development, giving one or more files a common label to refers to their revisions, is known as *tagging* those files. Tagging is typically used on entire modules, so that the current state of the module can be reconstructed in the future. This kind of tagging should always be done on project deliverables, and before starting major changes.

To tag one or more files or directories with a label, do the following:

1. Select the directory, file or files that you want to tag.
2. Click the right mouse button on the selection, and choose the *Tag selection -> Create a tag* menu item.
3. Enter the label in the *New tag name* input field. (See below for restrictions on tag names.)
4. Press *OK* button.

Note that CVS is quite restrictive when it comes to what characters a tag may contain: A tag must start with a letter, and may contain letters, digits, "`-`" (dash) and "`_`" (underscore) only. In particular, this means no dots, and no spaces. If you want to include version numbers in a tag, replace the dots with dashes. Two tag names are reserved, as they have special meaning in CVS: "`HEAD`" refers to the most recent version available in the repository, while "`BASE`" is the revision you last checked out into the local directory.

[CVS Doc: "Tags -- Symbolic revisions"]

time!

This is a "feature". CVS will always use UTC for logging.

Appendix B: How CVS Differs from Microsoft Visual Source Safe

CVS and WinCvs differ from Visual Source Safe (VSS) in many ways. The most apparent difference may be that CVS does not require users to lock the files they are working on, as VSS does by default. In fact, the CVS documentation even *encourages* users not to use file locking. In the rare occasion where several people have changed the same file at the same time, CVS will normally be able to merge their changes. If two or more developers have changed the same few lines, CVS will report a *conflict*, insert directives in the file, and leave it to the developers to decide what to do. Such conflicts are very rare, as they normally occur as a result of lacking communication between the developers (eg. two people trying to fix the same problem).

Another important difference is that VSS gives you a *server view*, while WinCvs shows a *client view*. In practice this means that, unlike VSS, WinCvs will not tell you about changes in the repository until you do an update, or explicitly query the status of selected files. Changes reported in the file view of WinCvs reflect modifications done by you after the last checkout, update or commit.

CVS is not as "visual" as VSS. When using WinCvs, it becomes quite clear that CVS was a command line driven program for many years before someone decided to encapsulate it in a GUI. Some of us prefer the simplistic approach of WinCvs, which may give a feeling of control for experienced users. Others will hate it. If you're one of the haters, feel free to improve the program and contribute your changes back to the community. :-)

2000-04-28 Sverre -- <shh@thathost.com>
The original CVS doc 1.10.6 that we reference is written by Per Cederquist et al. You may find its source on the official CVS site.

Brought to you by Computas AS.
Update 2003-06-13: I no longer work for Computas, and Computas no longer has this guide publicly available. The official site is now http://www.thathost.com/wincvs-howto/. Computas should still be credited, as they payed me to write this guide, and agreed to make it public.
Download `wincvs-howto.zip` if you want local access.

This guide is no longer maintained by the author. You'll find various updated offsprings with more or less proper credit of me and of Computas at the following locations: [Egil at Ikon], [G. Lawrence at AIMTec], [sourceforge], [Getos Ltd], [TortoiseCVS], [Kati Mikkolainen], [blog.empas.com]