



1.3 WinCVS

This program is distributed under the terms of the Gnu Public Licence

Users Guide

Copyright 2001 Don Harper

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified version of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Section 1 – Introduction	5
Section 2 – Download and Installation	6
Section 3 – CVS Repository Configuration.....	7
3.1 Local – Direct Access to Local Repository	7
3.2 Pserver, Rhosts, and Ssh – Remote Repository on Non-Windows Server	7
3.3 Ntserver – Remote Repository on Windows Server	7
Section 4 – Beginners Guide to WinCvs	8
4.1 Running WinCvs for the First Time	8
4.2 Setting WinCvs Preferences.....	10
4.2.1 General Preferences Panel	11
4.2.2 Globals Preferences Panel	13
4.2.3 WinCvs Preferences Panel.....	14
4.2.4 Command Dialogs Preferences Panel.....	16
4.2.5 Configuring the Shell.....	17
4.3 Customizing the ToolBar	18
4.4 Logging in to the server (pserver only).....	19
4.5 Creating the Repository	20
4.6 Creating Modules.....	22
4.6.1 Creating an Empty Top Level Module Using Import.....	22
4.6.2 Creating a Module from a Populated Directory Tree	25
4.6.3 Adding a Module to the <u>modules</u> File	28
4.7 Checking Out a Module – Creating a Working Directory.....	29
4.8 Changing the Browse Location.....	32
4.9 Updating a Working Directory.....	36
4.9.1 Running Query Update Prior to Update	36
4.9.2 Running the Update Command	38
4.10 Modifying Files.....	43
4.10.1 Using the Edit Command to Add Write Access	43
4.10.2 Viewing or Editing Files from WinCvs.....	45
4.10.3 Using the Unedit Command to Remove Write Access.....	47
4.11 Checking Diffs Prior to Commit.....	49
4.11.1 Generating a Text Diff.....	49
4.11.2 Generating a Graphical Diff	52
4.12 Committing Files and Folders.....	55
4.13 Adding Files or Folders to the Repository	60
4.13.1 Adding Files or Folders Using Add.....	60
4.13.2 Adding Files or Folders Using Import.....	68
4.14 Multiple Developer Coordination.....	73
4.14.1 Understanding Merging and the Unreserved Checkout Model.....	73
4.14.2 Example Commit with No-Conflicts Merge.....	73

4.14.3 Example Commit with Conflict Resolution.....	74
4.14.4 Understanding Locking and the Reserved Checkout Model.....	76
4.14.5 Using Log to Determine Lock Status	78
4.14.6 Unlocking Files.....	79
4.15 Revision History – Graph Command.....	81
4.16 Viewing a Previous Revision using Update	85
<i>Section 5 – Administrative Commands</i>	87
5.1 Maintaining CVS Administrative Files.....	87
5.1.1 Editing the Modules Administrative File.....	87
5.2 Recovering from Locked Repository	89
5.3 Release Management	89
5.3.1 Tagging a Product Release	90
5.3.2 Fixing Bugs after Product Release.....	93

Section 1 – Introduction

CVS stands for Concurrent Versions System. It is a version control system that has been developed in the public domain by many people beginning in 1986. Currently, CVS is maintained as an open source development project hosted by Collab.Net, Inc (see www.collab.net). Links to the CVS source code, binaries for various platforms and documentation can be found on the CVS project home page at www.cvshome.org.

The biggest limitation of CVS is that it uses a command line interface. Since most developers prefer a graphical user interface, several groups around the world have developed graphical front ends to the CVS core. The best one available for the Windows Operating System is WinCvs developed by a group of dedicated people from all over the world. Information about WinCvs is available on the web site at www.cvsgui.org.

Note that the versions of CVS and WinCvs documented here differ slightly from the standard releases. The only visible difference can be seen in the examples of the import command. The unofficial patch called Main Branch Import has been applied to both CVS and WinCvs to allow importing files directly to the trunk instead of a vendor branch. The patch can be found at www.cvshome.org/cyclic/cvs-html/dev-trunk-import.txt. The result of this patch is that revision numbers for imported files start with 1.1 instead of 1.1.1.1 which is the normal CVS behavior.

Anyone planning to use CVS and WinCvs seriously should begin by reading the available documentation. A number of very useful documents can be found on the CVS documentation page at www.cvshome.org/docs. In particular the “official” manual for CVS: Version Management with CVS by Per Cederqvist et al is required reading for any serious CVS user. This manual is available in various formats at www.cvshome.org/docs/manual. This manual is often referred to by the name of it’s author: the Cederqvist manual.

Another very useful document is the WinCvs – Daily Use Guide written by Sverre H. Huseby which can be found at www.computas.com/pub/wincvs-howto.

This document describes the installation procedure for WinCvs and explains some of the basics required to get started using CVS. Please address all questions or problems with this document to Don Harper donsjc@yahoo.com. Questions about the use of CVS or WinCvs should be directed to the info-cvs newsgroup at info-cvs@gnu.org. Information and links to archives for info-cvs can be found at mail.gnu.org/mailman/listinfo/info-cvs. A searchable archive of this newsgroup can be found at www.mail-archive.com/info-cvs@gnu.org. Answers to many questions about CVS and WinCvs can be found by browsing the info-cvs archives or in the CVS FAQ at ccvs.cvshome.org/fom/cache/1.html.

Section 2 – Download and Installation

The current release of WinCvs is 1.3 can be downloaded from www.sourceforge.net. The easiest way to find the correct file is to follow the links to the latest version from the WinCvs download page at www.cvsgui.org/download.html.

In addition to WinCvs itself, two applications are required to run WinCvs:

- WinCvs 1.3 requires installation of the Python programming language. Without Python, WinCvs will not run. Python can be downloaded from www.sourceforge.net. The easiest way to find the correct file is to follow the links to the latest released version from www.python.org/download.
- A very useful feature of WinCvs is the built in TCL shell. To enable this feature, the latest version of ActiveTcl or the latest version of Tcl/Tk must be installed. ActiveTcl can be found at aspn.activestate.com/ASPN/Downloads/ActiveTcl. Tcl/Tk can be found at www.scriptics.com/software/tcltk/downloadnow82.tml. Early beta versions of WinCvs 1.3 do not recognize ActiveTcl and require installation of a Tcl such as 8.2.

The following two optional applications may be installed if desired:

- WinCvs supports selection of the external editor to be run when editing or viewing a file. The default editor is Notepad. A really good shareware editor called EmEditor (\$30) can be found at www.emurasoft.com/emeditor3/index.htm.
- WinCvs supports selected of an external diff program used to graphically compare versions of a file. An excellent freeware file compare utility called ExamDiff is available on the web site at www.prestosoft.com/examdiff/examdiff.htm. A shareware version called ExamDiff Pro (\$25) has more features and is available at www.prestosoft.com/examdiff/examdiffpro.htm.

After all of the required and optional files are downloaded, unzip the WinCvs release and run the setup program (Setup.exe). WinCvs will add itself to the Start menu but a Desktop shortcut must be added manually if desired.

After installing WinCvs, install Python, ActiveTcl or Tcl/TK (optional), and any preferred external editor or graphical file compare utility. At this point WinCvs should be ready to run in the most basic configuration: a local repository.

Refer to Section 3 for further installation requirements depending on the protocol chosen for the CVS repository.

Section 3 – CVS Repository Configuration

Section 2 explained how to obtain and install the various software packages required to run WinCvs in the most basic configuration: a local repository. This section lists some local and remote repository configurations as well as authentication protocols used to communicate with a remote repository. If the reader is not familiar with CVS or the concept of a repository, now would be a good time to read the first two chapters of the Cederqvist manual.

When WinCvs is run for the first time or when Admin->Preferences is selected from the WinCvs command menu, the Preferences panel will be displayed. One of the fields on the General tab is the Authentication field. This field specifies one of the following protocols for communicating with the repository: **local**, **pserver**, **rhosts**, **ntserver**, and **ssh**. The following sections summarize the meaning of these settings as well as additional installation instructions if applicable.

3.1 Local – Direct Access to Local Repository

A local repository means that repository is accessible directly on same computer (Windows 95/98/Me/NT/2000) where WinCvs is running. This configuration is normally useful only for projects with just one developer since the files are only available on one computer. It is possible, of course, to set up a local repository on a mapped network drive thus contradicting the definition of a local repository. This is possible but should not be used as a way to share a repository among multiple developers since there is no way to synchronize access to the repository.

To configure a local repository, no additional software is required beyond what was described in Section 2.

3.2 Pserver, Rhosts, and Ssh – Remote Repository on Non-Windows Server

The most common configuration of WinCvs is remote access to the CVS repository running on a server system such as Unix(Solaris, SunOS, FreeBSD, etc.) or Linux. For this configuration, the appropriate version of CVS software must be obtained from www.cvshome.org/downloads.html. Alternatively, the appropriate binary can be built from the source distribution available at ccvs.cvshome.org/servlets/ProjectDownloadList.

Once the CVS source code or appropriate binary is downloaded, it must be built and/or installed on the server. The installation procedures for the binary distributions are system dependent. To build and install custom binaries from the source usually involves running the configure script, make, and make install.

To configure a remote repository using rhosts (rsh) or pserver, no additional software is required beyond what was described in Section 2.

To configure a repository using ssh (Secure Shell), additional work is required depending on what software is currently installed on the client (Windows) and server (Unix or Linux) computers. If SSH is not available, a version must be installed on both the client and server computers. Some of the web sites you may need to visit are listed here: www.cvs GUI.org/ssh.html, www.ssh.com, www.chiark.greenend.org.uk/~sgtatham/putty, and sources.redhat.com/cygwin.

3.3 Ntserver – Remote Repository on Windows Server

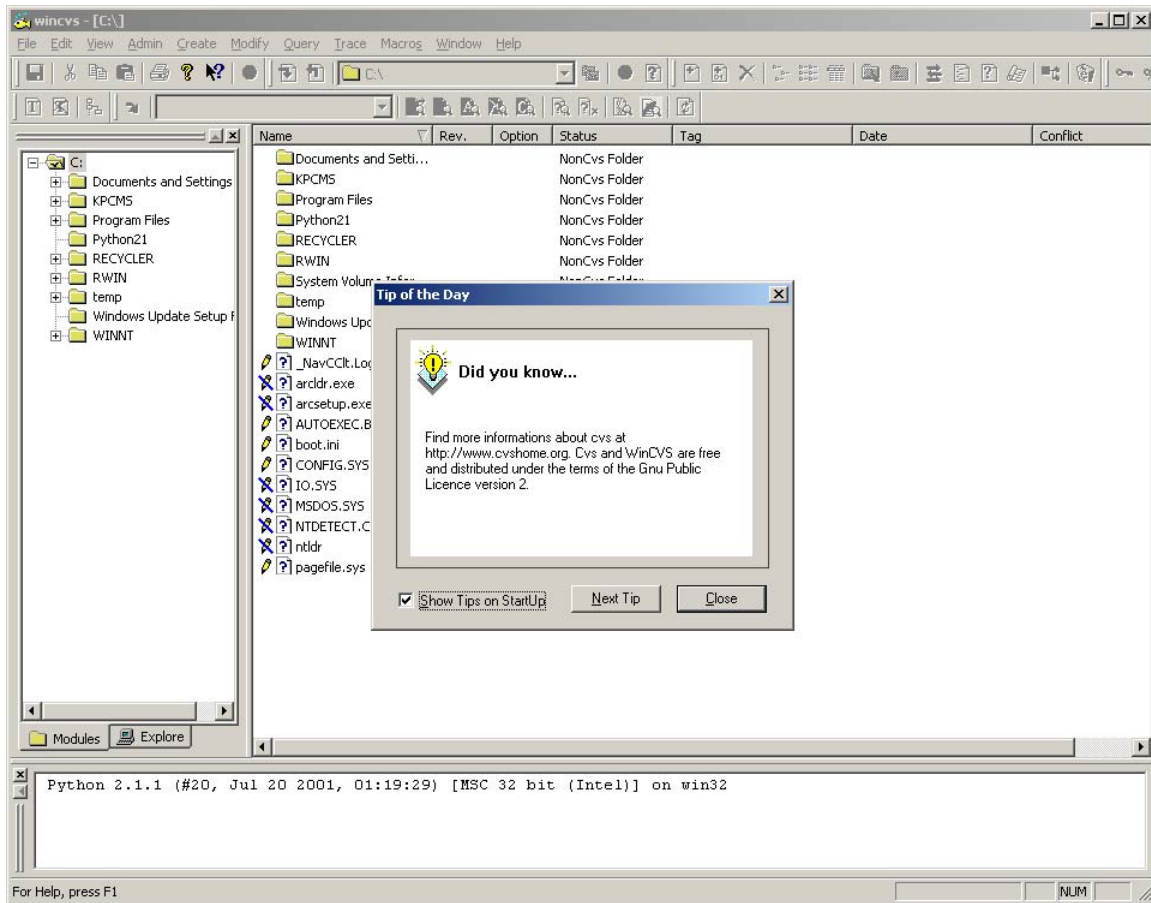
Another popular configuration of WinCvs is remote access to a CVS repository on a Windows server. For this configuration, CVSNT must be obtained from www.cvsnt.org. Detailed installation instructions can be found at www.cvsnt.org/readme.nt.

Section 4 – Beginners Guide to WinCvs

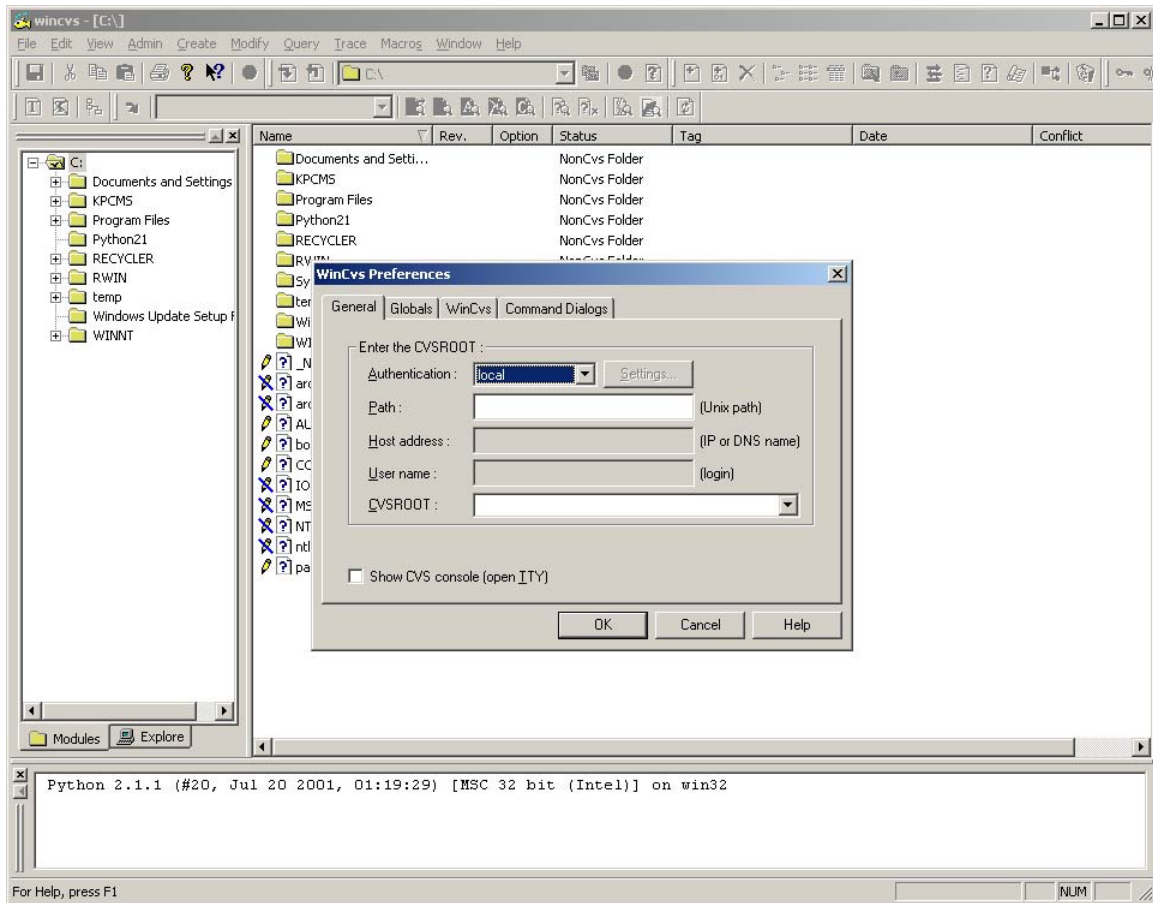
Before proceeding with this section, it is assumed that a protocol for communicating with the repository has been selected as described in Section 3. All additional software packages (such as ssh or cvsnt) must have been installed and configured as described in the appropriate documentation. For remote repository configurations such as local, pserver, and rhosts, the instructions in Chapter 2 of Cederqvist must have been followed prior to running WinCvs.

4.1 Running WinCvs for the First Time

When WinCvs is run for the first time, the WinCvs main window will appear showing the contents of the root folder of the system drive (C:\). The Tip of the Day dialog is displayed as shown here:

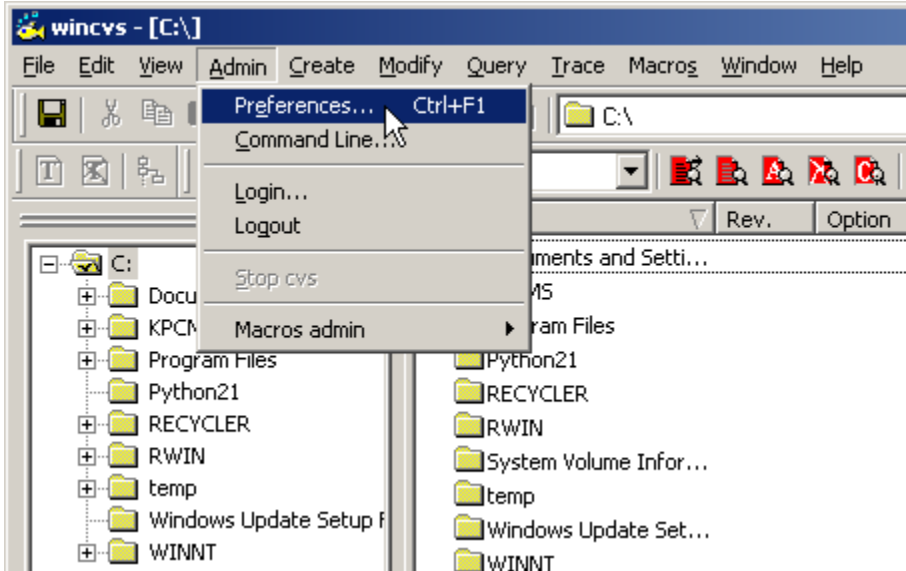


After closing the Tip of the Day dialog, the WinCvs Preferences dialog will be displayed as shown here:

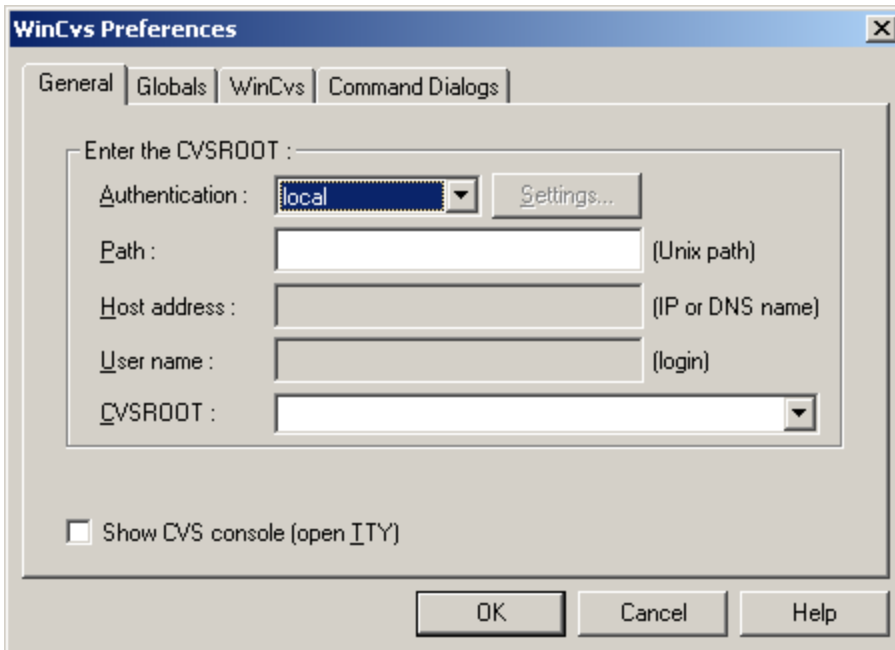


4.2 Setting WinCvs Preferences

As described in Section 4.1, the WinCvs Preferences dialog will be displayed automatically when WinCvs is run for the first time. This dialog can be reopened at any time using the Preferences command from the WinCvs Aadmin menu as shown here:



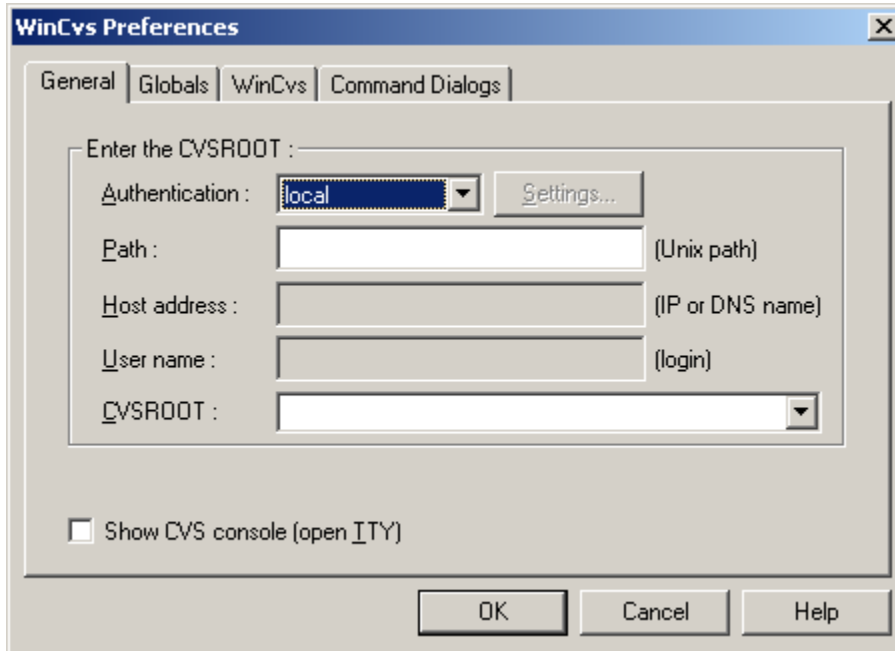
The default contents of the WinCvs Preferences dialog are shown here:



The General Preferences Panel must be filled in before WinCvs can be used for anything meaningful. See Section 4.2.1 for help with the General Preferences Panel.

4.2.1 General Preferences Panel

To display the General Preferences panel, open the WinCvs Preferences dialog as described in Section 4.2. There are five fields on the General Preferences panel as shown here:



The **Authentication** field specifies the protocol used to communicate with the CVS repository. Select the appropriate entry from the pull-down list based on the repository configuration. See Section 3 of this manual for further information on repository configurations.

The **Path** field specifies the location in the file system where the repository will be located. If a local repository will be used (local authentication), specify a local path such as `C:\cvs\myrepository`. If a remote repository will be used (pserver, rhosts, ntserver, or ssh authentication), specify the path on the remote server such as `/export/cvs/myrepository`.

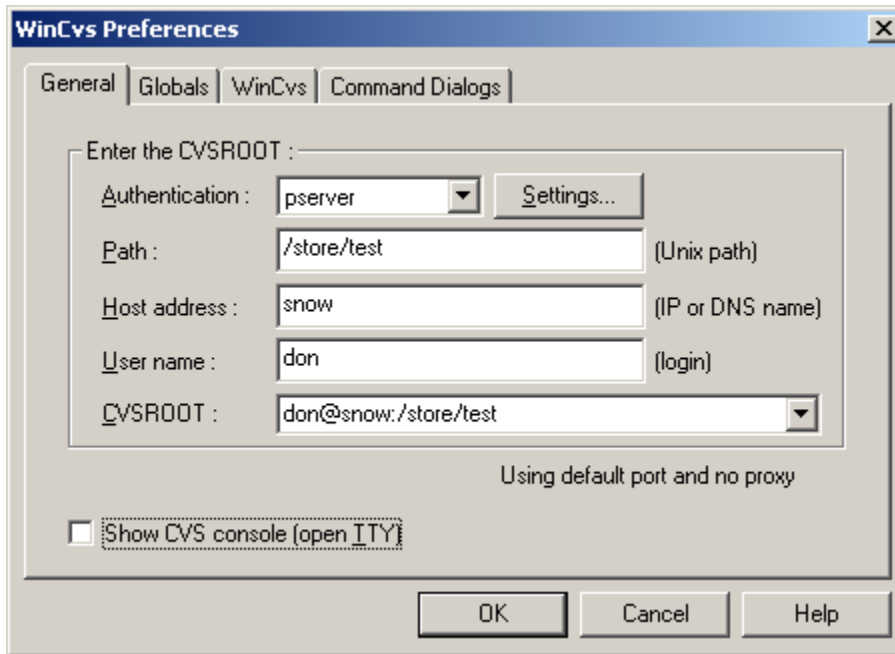
The **Host address** field is only required for remote repositories. This field specifies the IP address or host name of the server where the CVS repository will reside. To specify a host name, the IP address of the host must be available from a domain name server (DNS) or listed in the hosts file on the local computer. For Windows NT and Windows 2000 computers, this file is `C:\WINNT\system32\drivers\etc\hosts`.

The **User name** field is only required for remote repositories. This field must specify a valid username on the server where the CVS repository will reside. CVS will require a successful login with this user name when the pserver authentication is selected. Other authentication protocols such as rhosts, ntserver, and ssh do not require a login.

The **CVSROOT** field is a composite of the User name, Host, and Path and need not be edited directly. The CVSROOT is what is actually used to tell CVS where the repository is located. Refer to Section 2.1 of the Cederqvist manual for further explanation of CVSROOT.

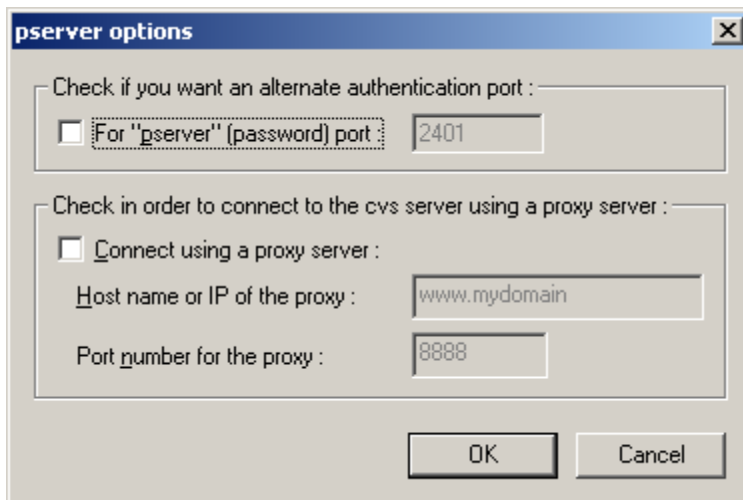
When WinCvs invokes a CVS command, it spawns a background CVS process. The **Show CVS console (open TTY)** option causes the CVS process to be invoked in the foreground. This is sometimes useful when using ssh authentication (see the ssh documentation).

An example of a General Preferences panel for a remote repository using pserver authentication is shown below:



When WinCvs is run for the first time, all required fields on the General Preferences panel should be filled in and the **OK** button clicked prior to displaying any other tabs in the WinCvs Preferences. If another tab is displayed, it is possible that the data on the General tab will be saved incorrectly. If another tab is displayed prior to clicking the **OK** button, go back to the General Preferences tab and ensure the data is correct prior to closing the dialog.

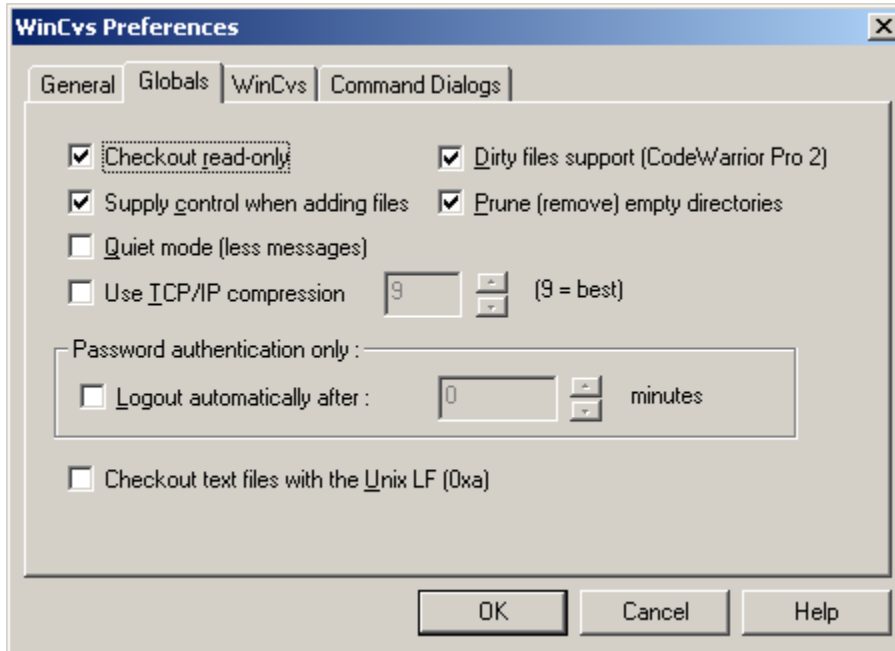
Some less frequently used options are available on the pserver options panel accessible by clicking on the **Settings...** button to the right of the Authentication field:



Use this panel to specify a pserver authentication port other than the default 2401 or to configure a proxy server.

4.2.2 Globals Preferences Panel

To display the Globals Preferences panel, open the WinCvs Preferences dialog as described in Section 4.2 and click on the Globals tab. Beginning users of WinCvs should leave the default settings as shown here:



If the **Checkout read-only** box is checked, all files will be set to read-only when checked out of the repository using the Checkout module... command from the WinCvs Create menu. Read-only files can only be modified by first adding write access using the CVS edit command.

If the **Supply control when adding files** box is checked, WinCvs will attempt to give advice when it thinks a file being adding is the wrong type. For example, adding as a binary file when it appears to be a text file will result in a warning dialog.

If the **Quiet mode (less messages)** box is checked, CVS commands will produce fewer messages in the WinCvs Output Pane. Important messages will still be printed but informational messages such as listing the name of each directory during recursion will be omitted.

If the **Use TCP/IP compression** box is checked, the data transferred between WinCvs and the CVS server will be compressed. This option should only be used when communication is done through a slow modem since it adds significant processing on both the client and server computers. The number stands for the level of compression from 0 (minimum compression) to 9 (maximum compression).

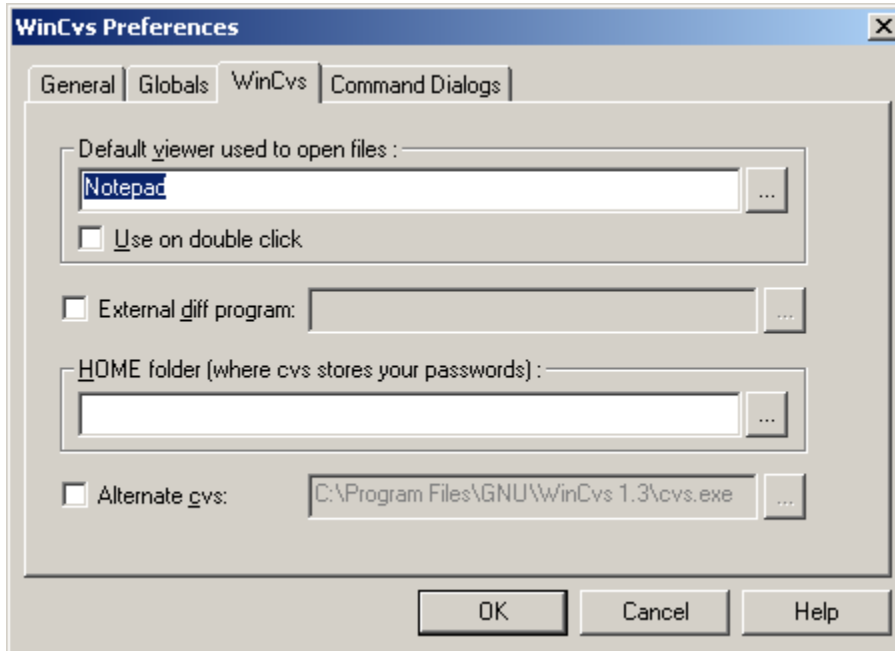
If the **Dirty files support (CodeWarrior Pro 2)** box is checked, files with extension .mcp are considered dirty files and are ignored by WinCvs unless specifically named in a command. When committing a folder, for example, modified files with .mcp extensions will be ignored (not committed). If changes are made to a .mcp file that need to be committed, the file can be committed by individually selecting the file for commit.

If the **Prune (remove) empty directories** box is checked, empty folders in the working directory will be deleted if they become empty as a result of an update operation. Empty folders will not be created on a checkout operation.

If the **Checkout text files with the Unix LF (0xa)** option is checked, the work area will be maintained in Unix file format. This is useful for environments such as Cygwin. Note that you cannot use WinCvs in both modes.

4.2.3 WinCvs Preferences Panel

There are four items on the WinCvs Preferences panel. Beginning users of WinCvs need to at least specify the HOME folder as described below. The default contents of the WinCvs Preferences panel are shown here:



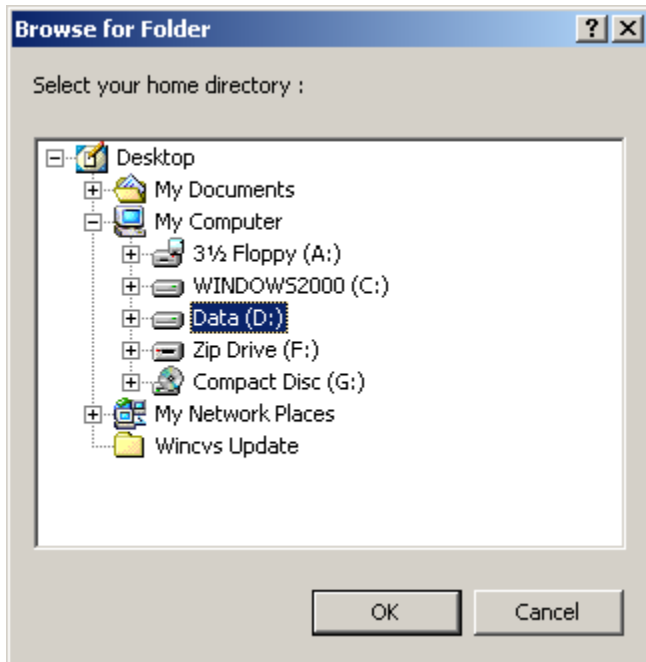
The **Default viewer used to open files:** field allows the user to specify an editor other than the default Notepad to be used when editing or viewing files from within WinCvs. There are two ways to open a file for edit/view from WinCvs: use one of the **view** commands from the Query menu or double-click on the file. The standard **view** command will open the file with application associated with the file type in Windows. The Query menu also has an explicit entry to open the file with the **Default viewer** specified in the WinCvs Preferences. If the **Use on double click** box is checked, the file will be opened with the **Default viewer** instead of the Windows associated application.

If the box to the left of the **External diff program:** field is checked, the specified program will be used to compare files when an external diff is requested. External diff is used any time the diff command is invoked from the graph display or when the Use the external diff box is checked on the Diff settings panel.

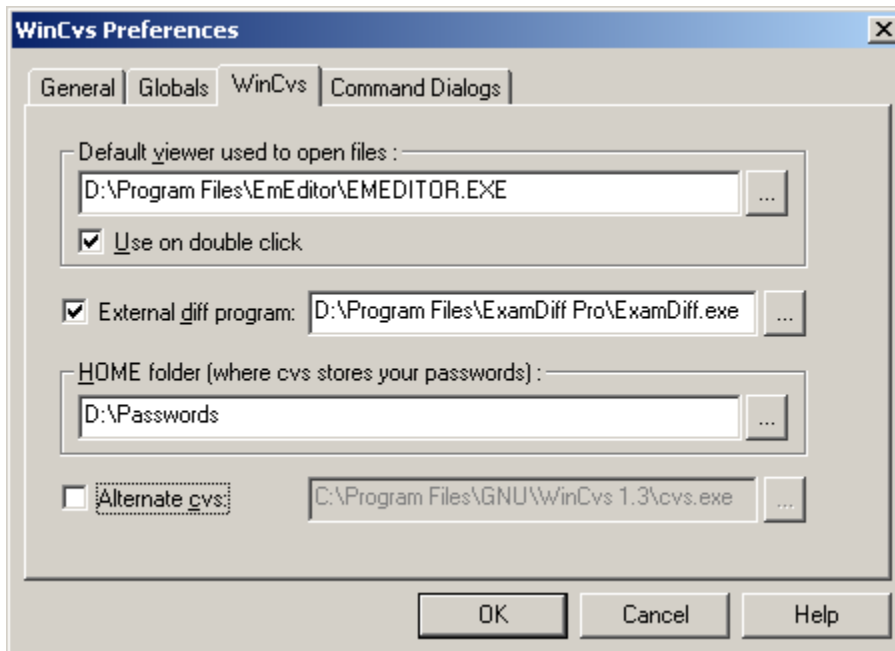
As the comment states, the **HOME folder** is used by CVS to store passwords. CVS will create a file in this folder called .cvspass which will store a password for each repository that has been accessed from WinCvs. Since this folder is used for all repositories accessed from WinCvs, the folder should not be part of any CVS working directory or repository. This folder is also the home for the global .cvsignore and .cvsrc files.

The **Alternate cvs** field should be ignored. Checking this box allows specification of a folder that contains a different version of the internal CVS client built into WinCvs. This feature is for debugging WinCvs only. If this option is used, it must be noted that not all versions of cvs.exe (such as Cygwin cvs) can work with WinCvs.

Note that if the HOME folder field is left blank, WinCvs will ask for a HOME folder the first time an attempt is made to execute a CVS command such as **login**, **checkout**, etc. as shown here:

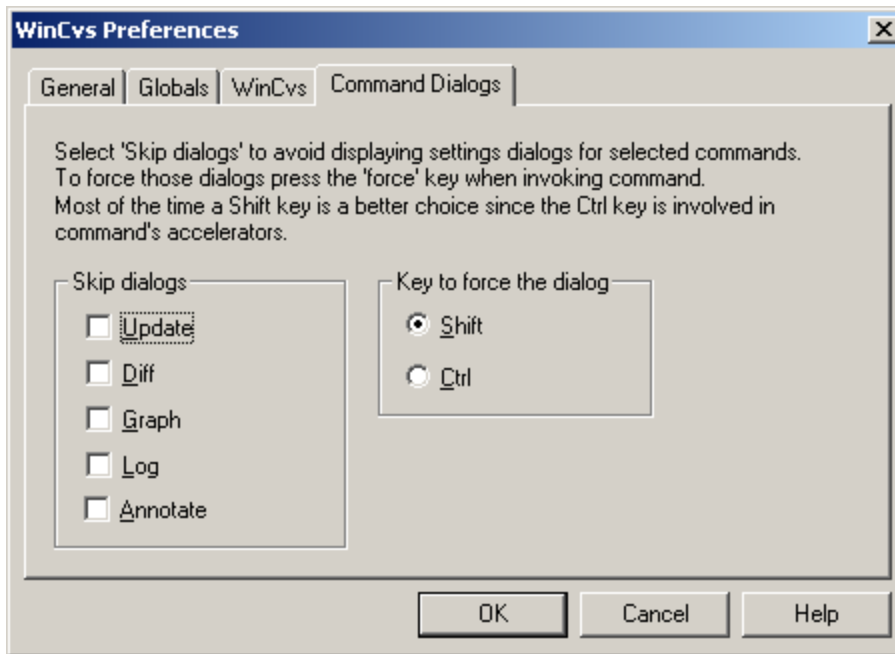


The following example shows a typical WinCvs Preferences Panel:



4.2.4 Command Dialogs Preferences Panel

When using WinCvs with the default preferences, there are many commands that pop up a settings dialog prior to executing the command. Since the default settings are most often used, it may be useful to disable the settings dialog for some or all of the common commands: **update**, **diff**, **graph**, **log**, and **annotate**. The settings panel can then be forced by holding down the SHIFT key (or CTRL if preferred) when invoking the command. Refer to the comment on the Command Dialogs preferences panel for more information:

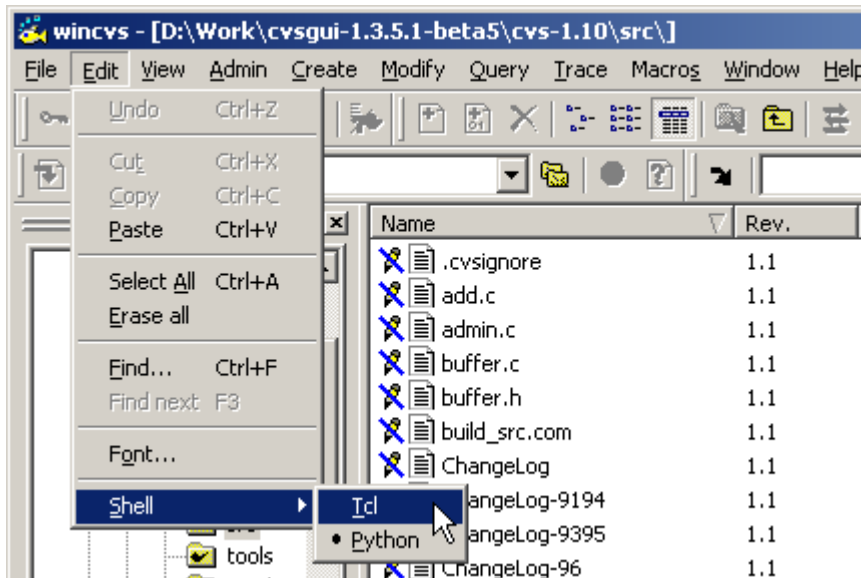


4.2.5 Configuring the Shell

The Output pane, which by default is in the lower portion of the WinCvs display, can be used to manually enter CVS commands as described in several examples in this manual. When WinCvs starts up, the Output pane will normally display messages such as this:

```
Python 2.1.1 (#20, Jul 20 2001, 01:19:29) [MSC 32 bit (Intel)] on win32
CVSROOT: don@snow:/Store/Cvs-Admin (password authentication)
TCL is available, shell is enabled : help (select and press enter)
```

There are two shells available for entering commands in the Output pane: Python and Tcl. By default, the Python shell is enabled when WinCvs starts up. To switch to the Tcl shell, use the WinCvs **Edit** menu as shown here:



When switching from Python to Tcl as in the above example, the following message will be displayed in the Output pane:

```
Switching the shell to TCL...
```

The syntax for CVS commands using the Python shell and Tcl shell are very different. For example, the CVS **login** command would be entered like this using the Python shell:

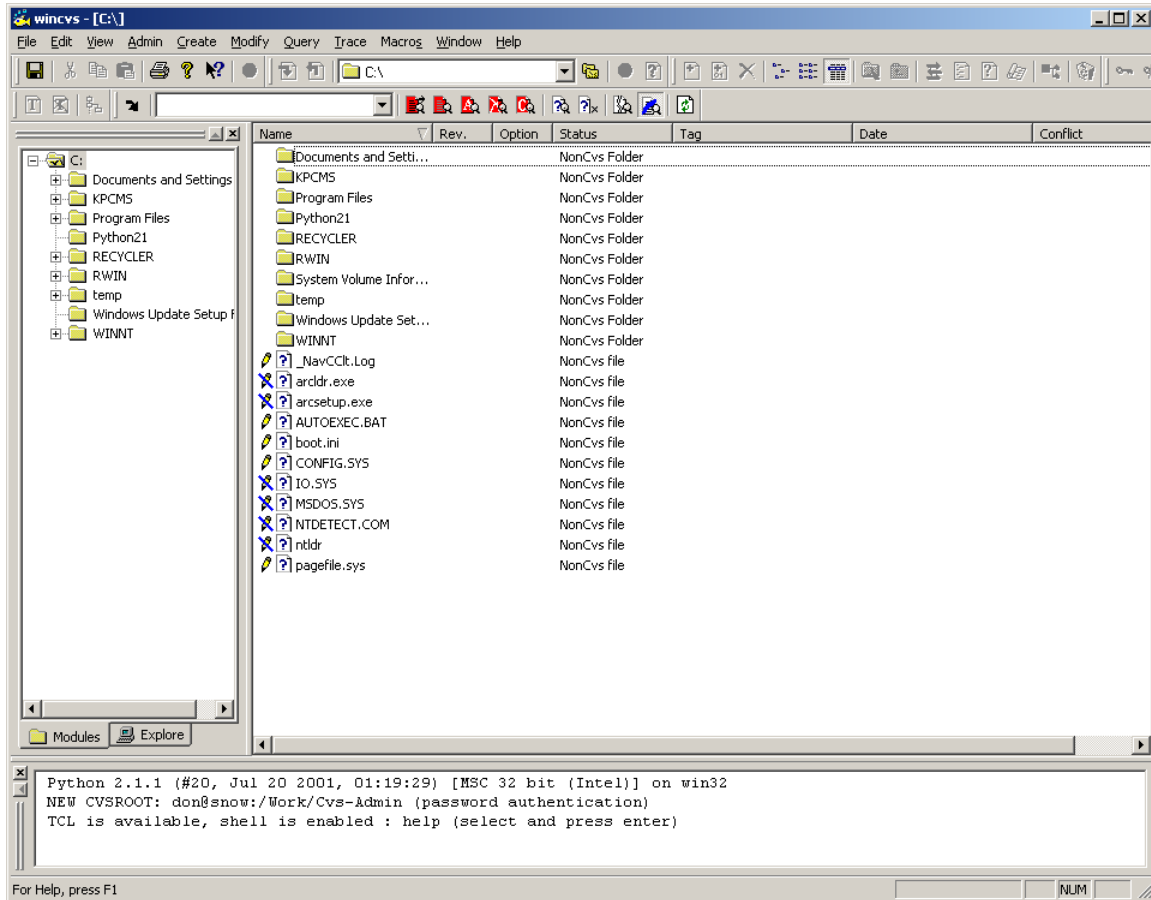
```
import cvsgui; cvsgui.Cvs.Cvs().Run("login")
```

The same command would be entered like this using the Tcl shell:

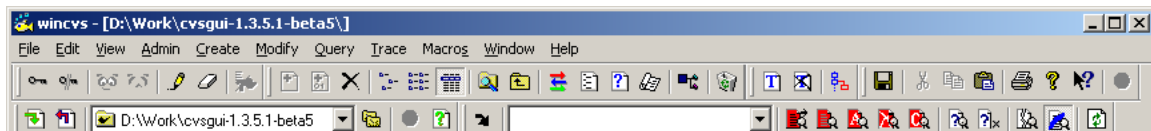
```
cvs login
```

4.3 Customizing the ToolBar

After starting up WinCvs and entering the WinCvs Preferences as explained in Section 4.2, the WinCvs main window is displayed. By default, WinCvs sets the displayed location to C:\ so the folders and files of C:\ are displayed as shown here:



Note that several of the icons on toolbar are outside the window border and therefore not visible. There are actually six individual toolbars that are available in WinCvs: Standard, Browse, Files, Multi-Users, Tags, and Filter. The gripper on each of the toolbars can be used to float, dock, or rearrange the toolbars as desired. For the remainder of this document, the toolbar layout will be configured as shown here:

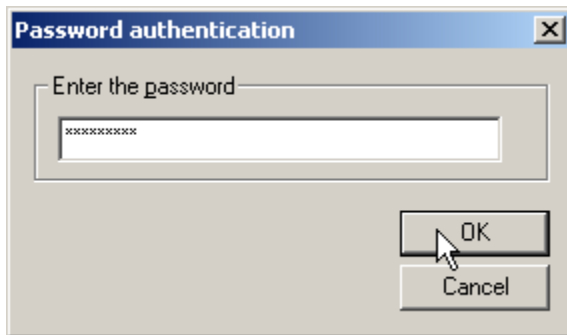


The function of the icons on the toolbar will be explained in subsequent chapters.

4.4 Logging in to the server (pserver only)

For remote repository configurations with pserver authentication, successful login is required prior to issuing any CVS commands. For local, rhosts, ntserver, or ssh authentication, this section is not applicable.

Log into CVS by selecting the **Login...** command from the WinCvs **Admin** menu. The **Password authentication** dialog will be displayed as shown here:



Enter the password for the user name specified in the WinCvs General Preferences (see Section 4.2.1) and click the **OK** button. Review the CVS messages in the Output Pane (lower panel of the WinCvs display window) to determine the status of the login.

A successful login will display messages such as:

```
cvcs -q login
(Logging in to don@snow)
*****CVS exited normally with code 0*****
```

An unsuccessful login will display messages such as:

```
cvcs -q login
(Logging in to don@snow)
cvcs [login aborted]: authorization failed: server snow rejected access
*****CVS exited normally with code 1*****
```

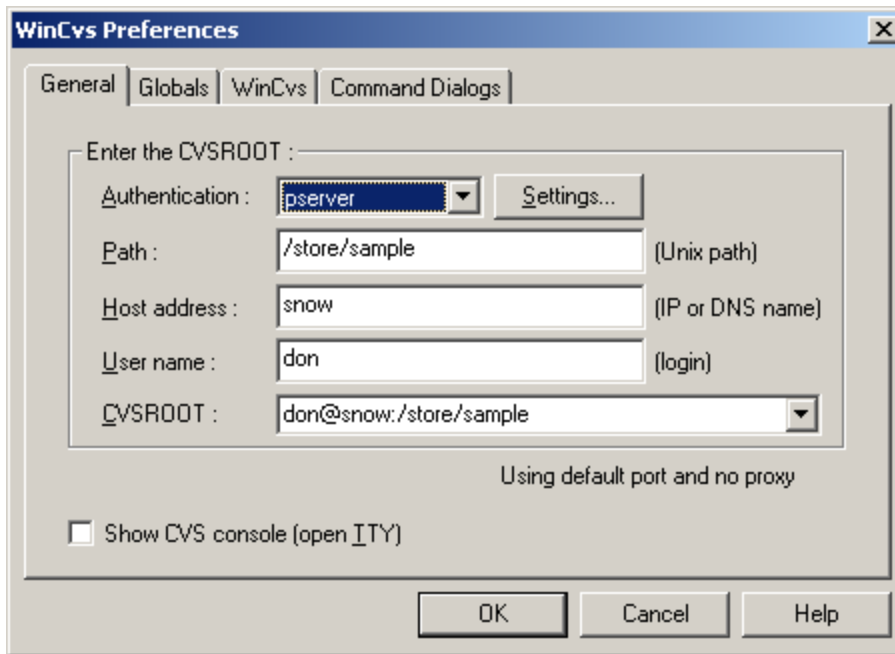
Attempts to use CVS commands prior to logging in will result in messages similar to the following attempt to run the checkout command:

```
cvcs checkout -P module1 (in directory D:\Work)
cvcs.exe checkout: authorization failed: server snow rejected access to
/store/test for user don
cvcs.exe checkout: used empty password; try "cvcs login" with a real password
*****CVS exited normally with code 1*****
```

4.5 Creating the Repository

Before WinCvs or CVS can be used to manage any files, a repository must be created. If the repository has already been created, skip this section. Otherwise, follow the example in this section to create a repository.

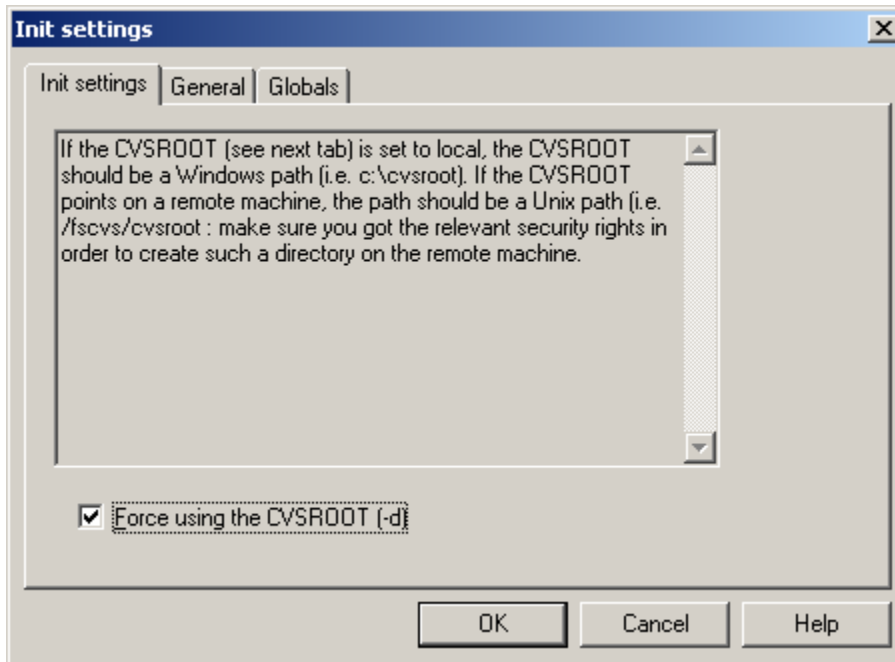
In this example, the repository resides on a Sun Solaris server. The name of the repository is **sample** and is located in the directory `/store/sample`. Communication with the server (snow) is using pserver authentication. The WinCvs General Preference panel in this example looks like this:



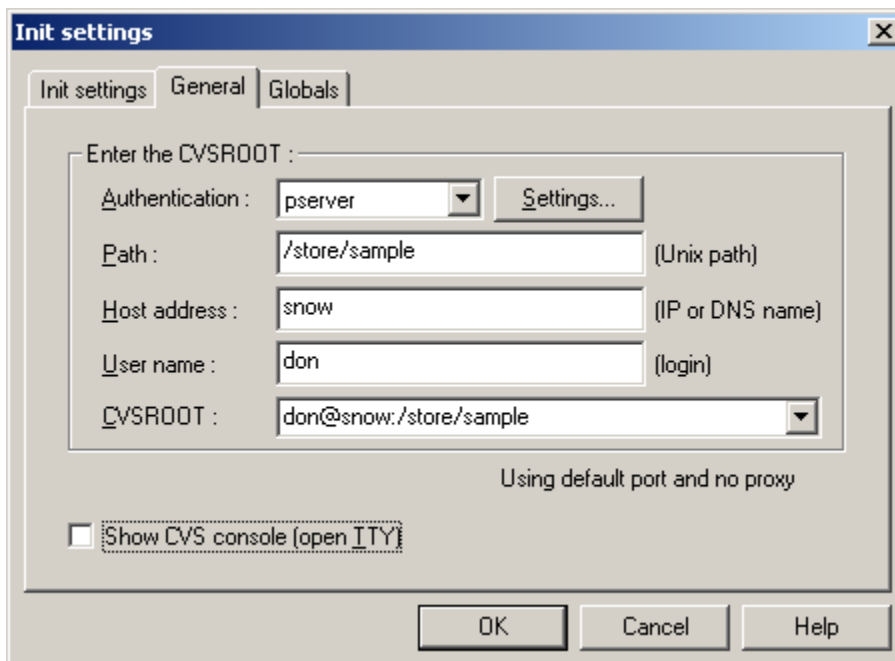
Note that prior to creating the repository, the user must be logged in as described in Section 4.4. A previous login to a different repository will not be recognized.

In addition to logging in, all parent directories (folders) of the new repository must be created. CVS will automatically create the directory (folder) for the new repository itself but will fail if the parent directory does not exist.

Select the **Create new repository...** command from the WinCvs **Create** command menu. The **Init settings** dialog will be displayed as shown here:



From the **Init settings** dialog, it is also possible to confirm the WinCvs General Preferences by clicking on the **General** tab as shown here:



After verifying all parameters, click **OK** to create the repository. The Output Pane will look like this:

```
cvs -d :pserver:don@snow:/store/sample init
*****CVS exited normally with code 0*****
```

4.6 Creating Modules

CVS organizes repositories into modules. A module is a hierarchy of folders and files beginning at any folder in the hierarchy of the repository. Every repository has a top level module called CVSROOT that stores administrative files. It is probably good practice to have a repository administrator maintain the CVSROOT module. It also may be good practice to allow only the administrator permission to add new modules at the top level of the repository.

Files and directories (folders) can only be added to an existing module in the repository. Before any files or folders can be added to the repository, it is therefore necessary to create at least one module.

There are two ways to create modules in a repository: using the CVS import command, or migrating repositories from other version control systems (RCS, SCCS, PVCS, etc). See Section 3.1.2 of the Cederqvist manual for tips on migrating from other version control systems.

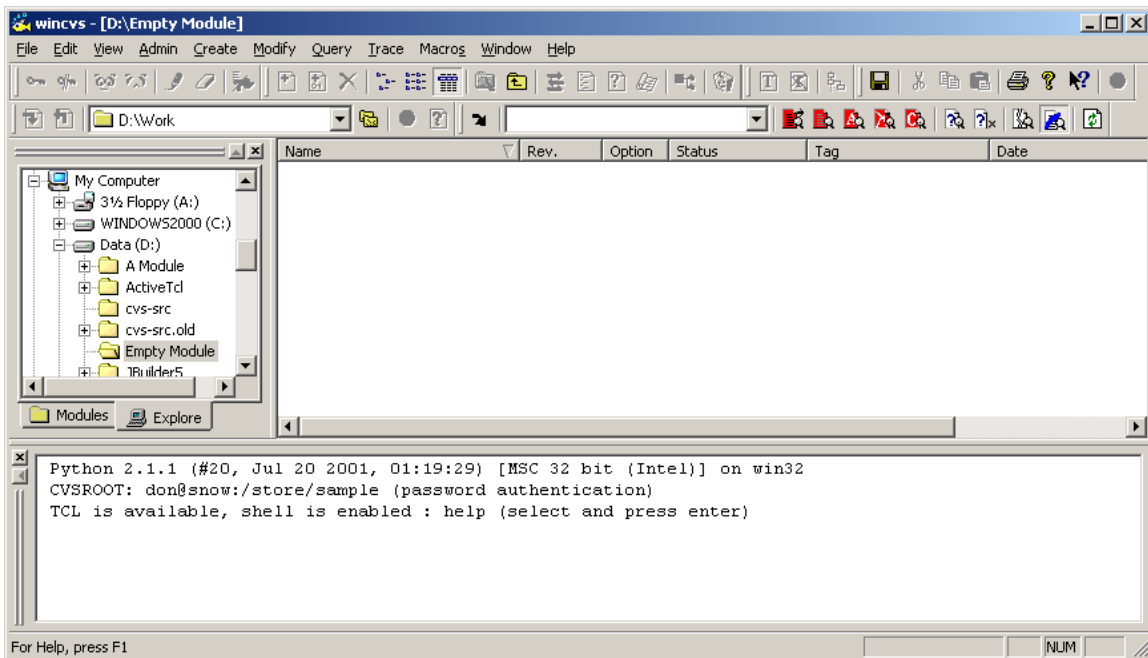
The CVS import command can be used to import an entire tree of source files or just import an empty top level directory for the module. The following two sections give examples these two types of imports.

4.6.1 Creating an Empty Top Level Module Using Import

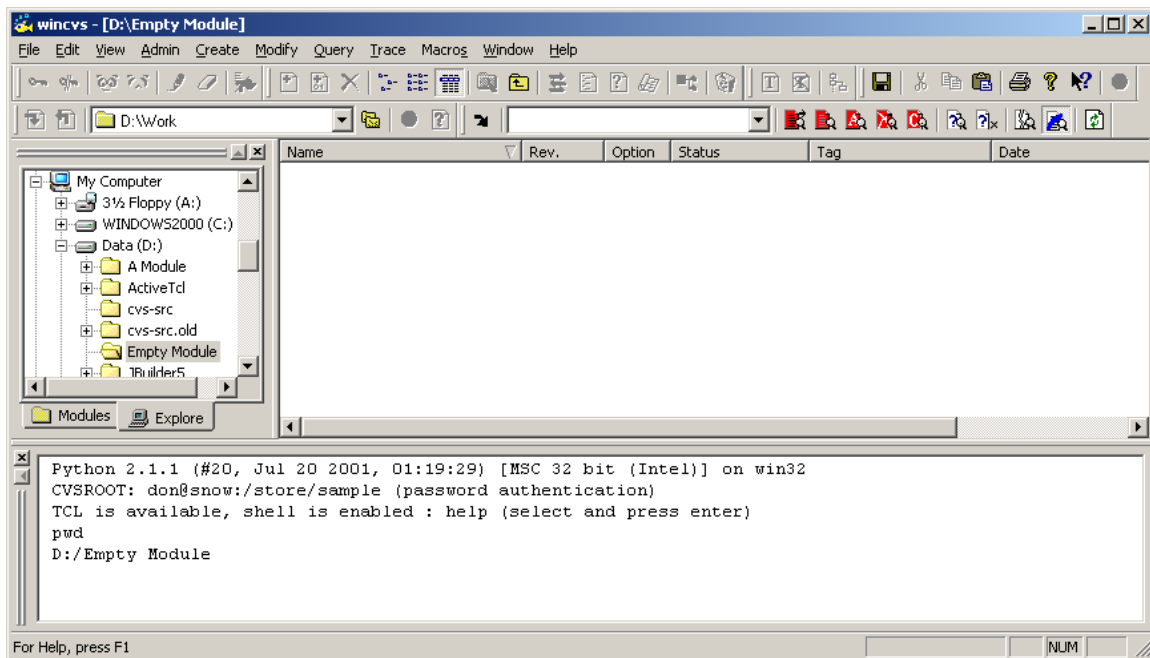
When creating a new project from scratch, files and folders will be added to CVS as they are created. To get started, however, a module must be created. The easiest way to do this with WinCvs is to import an empty folder.

Start by creating an empty folder anywhere in your Windows file system. In this example the folder will be D:\Empty Module. This folder will be used to create a new module called module1. Next, run WinCvs and make sure there is a current login for the repository where the module will be created.

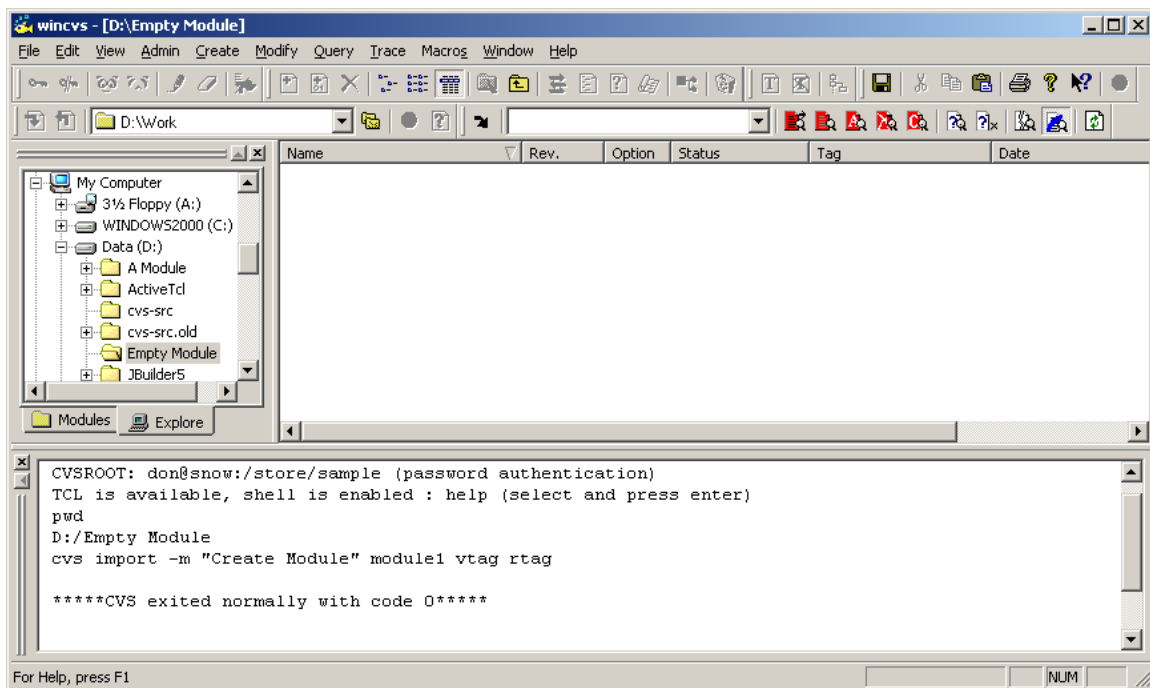
Display the Explore panel of the Workspace panel (left side of WinCvs window) by clicking on the Explore tab at the bottom of the panel. Select the empty folder to import as shown here:



Verify that the current folder is correct by entering the **pwd** command in the WinCvs Output Pane as shown here:



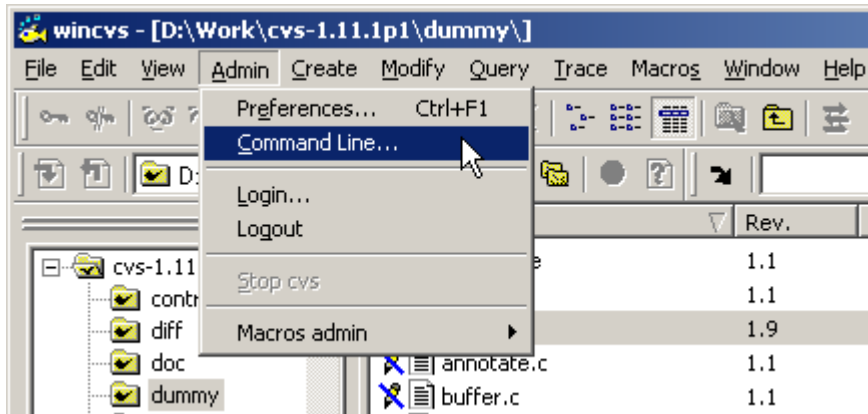
Finally, enter the **import** command in the WinCvs Output Pane as shown here:



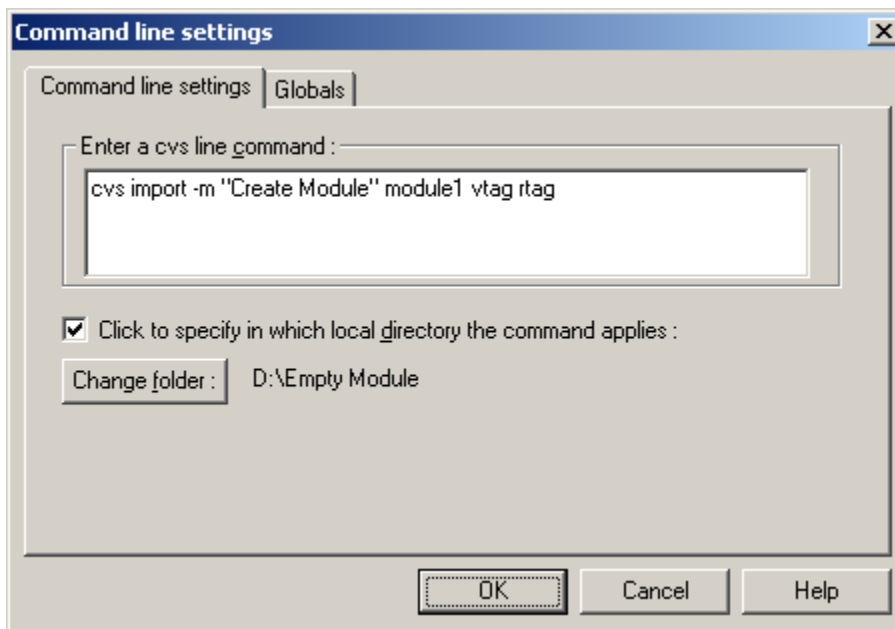
In the above example, an empty module called **module1** was added to the repository. The additional parameters to the import command (-m, vtag, and rtag) are explained in the Cedervqist manual.

Note that the reason the import command must be invoked from the command line in this example is that WinCvs does not support importing empty folders directly. Issuing the command manually from the command line gets around this limitation.

Another way to issue the import command from the command line without typing directly in the Output Pane is to use the **Command Line...** command from the WinCvs **Admin** menu as shown here:



The **Command line settings** dialog will be displayed. Enter the **import** command as shown in the following example. Note that **D:\Empty Module** was set as the local directory by checking the **local directory** box and browsing to the folder with the **Change folder** button:

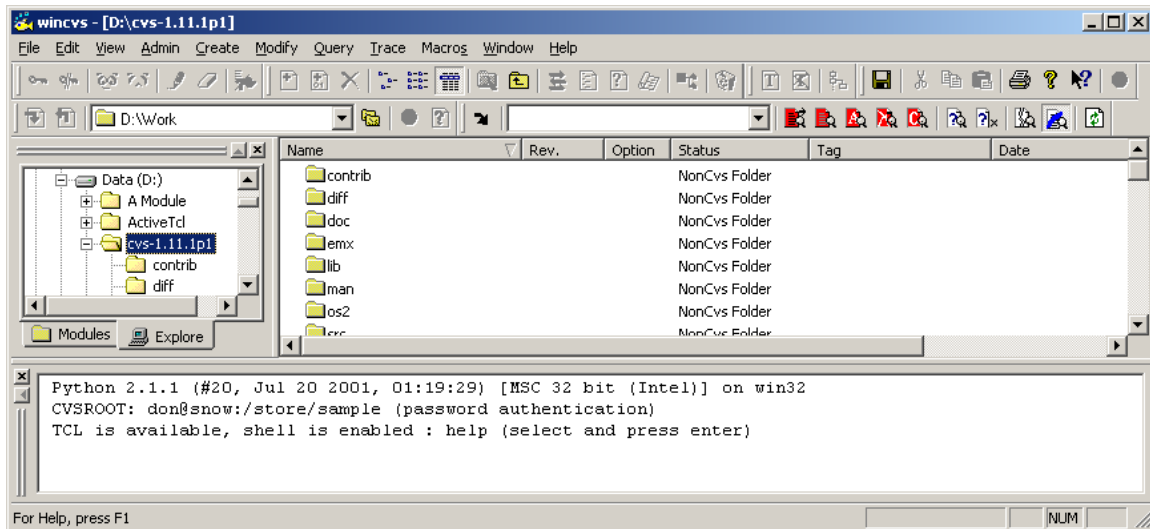


4.6.2 Creating a Module from a Populated Directory Tree

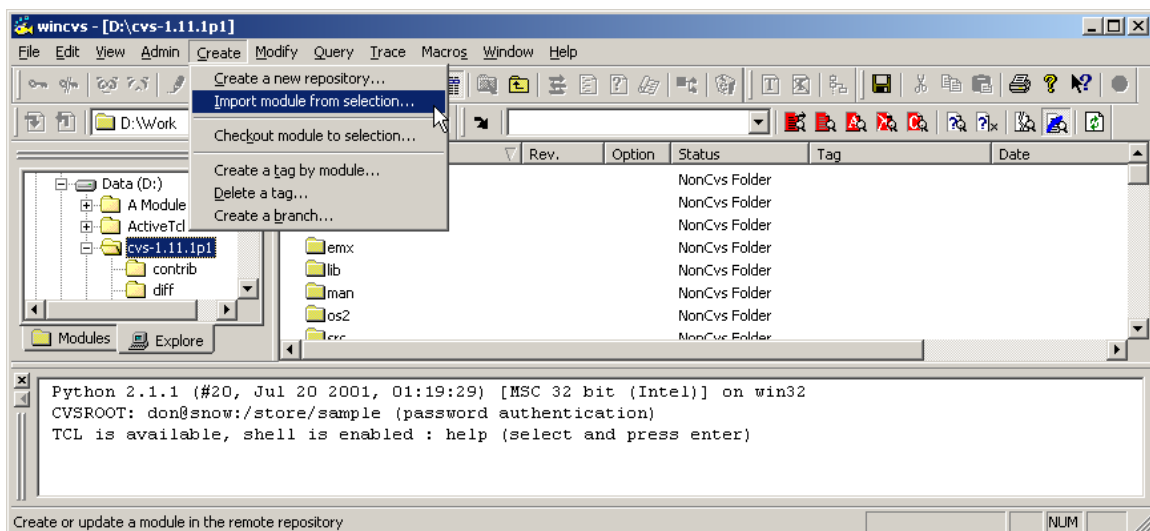
A very common way to create a module in CVS is to import a complete tree of files and folders.

In this example the folder `D:\cvs-1.11.1p1` will be imported. This folder contains the complete source code for CVS version 1.11.1p1. Start by running WinCvs and make sure there is a current login for the repository where the module will be created.

Display the Explore panel of the Workspace panel (left side of WinCvs window) by clicking on the Explore tab at the bottom of the panel. Select the folder to import as shown here:



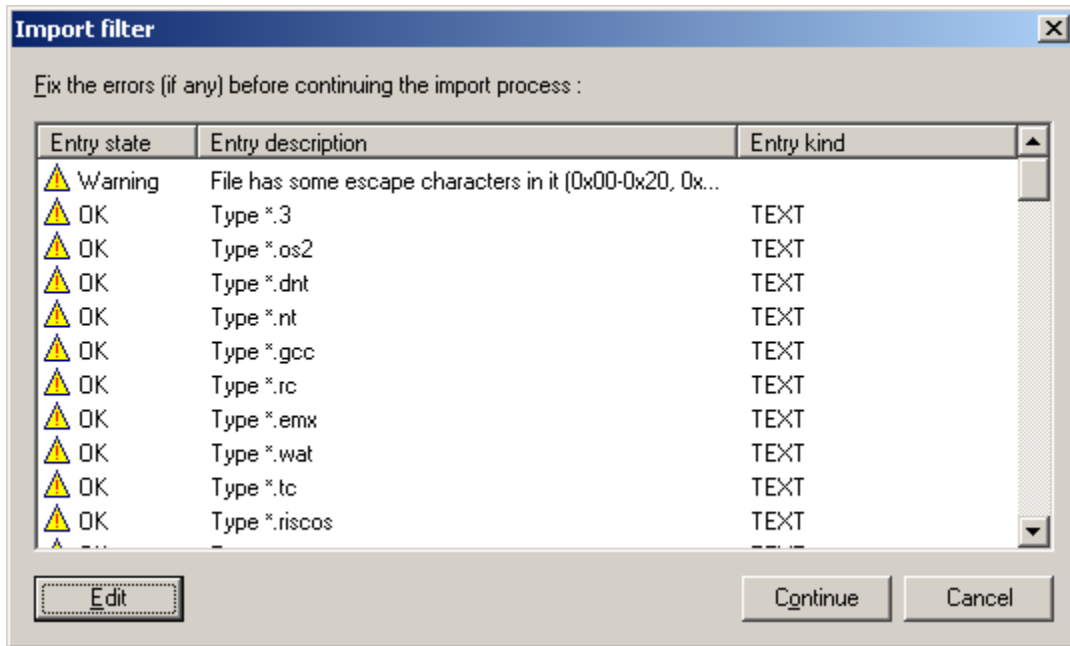
Next, select the **Import module from selection...** command from the WinCvs **Create** menu as shown here:



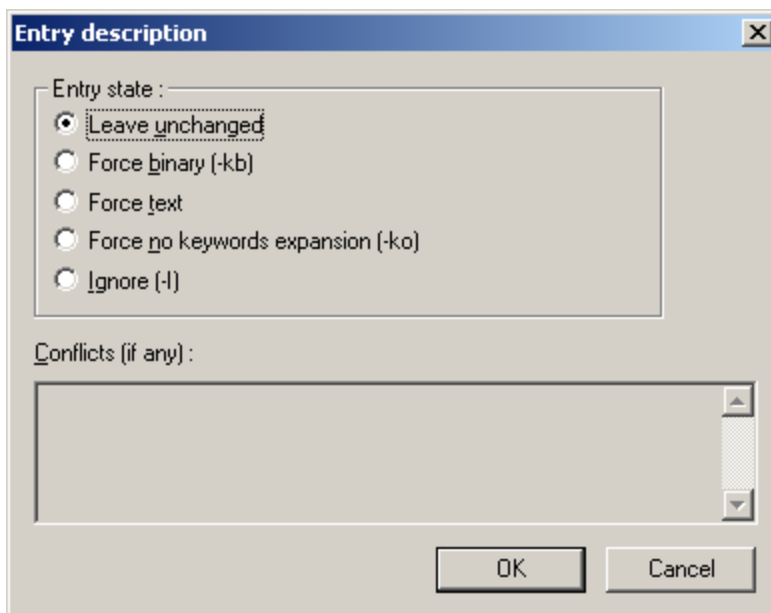
After invoking the **import** command, there may be a delay while WinCvs examines the files to be imported. During this time, messages such as the following will be displayed in the WinCvs Output Pane:

```
Filtering 'D:\cvs-1.11.1p1'...
Filtering 'D:\cvs-1.11.1p1\contrib'...
Filtering 'D:\cvs-1.11.1p1\diff'...
Filtering 'D:\cvs-1.11.1p1\doc'...
```

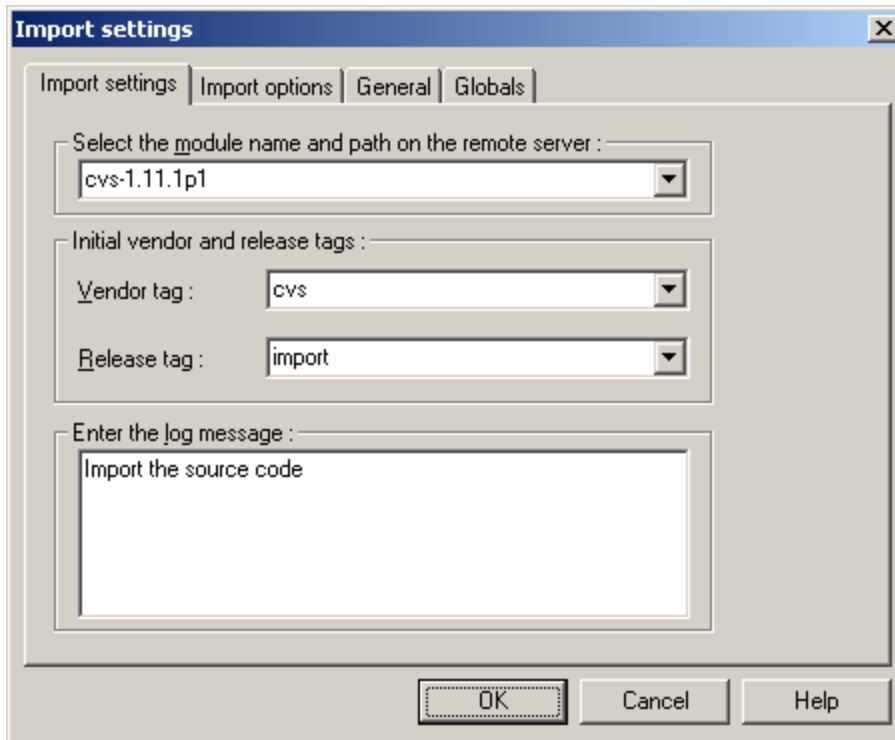
When the filtering process is complete, WinCvs will display a summary of the file types found. For each file, the type of file (text or binary) is listed as shown here:



To override the file types recommended by WinCvs, select the file type to be modified and click the **Edit** button. The **Entry description** dialog will be displayed as shown here:



After making any desired edits to the entry types listed on the **Import filter** dialog, click the **Continue** button to proceed with the import process. At this point, the **Import settings** dialog will be displayed as shown here:



By default, the name of the module to create will be the name of the folder being imported. This name can be changed to any name. It may be desirable to change the name to remove any embedded spaces since it can be inconvenient to have module names containing spaces.

The Vendor tag, Release tag, and Log message are for informational purposes only and may be left as the default values or changed to any appropriate values. When module name, tags, and optional log message have been entered, click the **OK** button to proceed with the import.

The import command takes some time to complete its processing depending on the size of the tree being imported. A successful import will have messages similar to the following displayed in the WinCvs Output Pane:

```
cvs import -I ! -I CVS -I Makefile -m "Import the source code" cvs-1.11.1p1 cvs
import (in directory D:\cvs-1.11.1p1)
I cvs-1.11.1p1 source/windows-NT/SCC/CVS
.
.
.
cvs server: Importing /store/sample/cvs-1.11.1p1 source/zlib/os2
N cvs-1.11.1p1 source/zlib/os2/Makefile.os2
N cvs-1.11.1p1 source/zlib/os2/zlib.def
```

No conflicts created by this import

```
*****CVS exited normally with code 0*****
```

4.6.3 Adding a Module to the modules File

After a module has been created using the import command (Sections 4.6.1 and 4.6.2), the module should be defined in the CVS modules file. Although this is not required (modules can be checked out whether or not they are listed in the modules file), it can be useful. As explained in Section 4.7, module names must be known before they can be checked out from the repository. Modules that are defined in the modules file will appear in the output of the Get the cvs modules command in the Admin->Macros admin menu.

To define a module in the modules file, the modules file must first be checked out from the repository. The modules file belongs to the CVSROOT module which exists in every repository. The steps required to edit the modules file are:

- Check out a working copy of the CVSROOT module to a working directory
- Point the WinCvs browser to the working directory
- Edit the modules file (add write permission)
- Open the modules file in an editor and add the new module
- Save the modified file
- Commit the modified file to the repository

To accomplish this requires skills that are explained in Sections 4.7 through 4.13. Refer to those sections prior to trying to edit the modules file. Further information on the modules file can be found in appendix C.1 of the Cederqvist manual.

Sample contents of a modules file are shown here:

```
# Three different line formats are valid:
#   key   -a   aliases...
#   key [options] directory
#   key [options] directory files...
#
# Where "options" are composed of:
#   -i prog           Run "prog" on "cvs commit" from top-level of module.
#   -o prog           Run "prog" on "cvs checkout" of module.
#   -e prog           Run "prog" on "cvs export" of module.
#   -t prog           Run "prog" on "cvs rtag" of module.
#   -u prog           Run "prog" on "cvs update" of module.
#   -d dir            Place module in directory "dir" instead of module name.
#   -l               Top-level directory only -- do not recurse.
#
# NOTE:  If you change any of the "Run" options above, you'll have to
# release and re-checkout any working directories of these modules.
#
# And "directory" is a path to a directory relative to $CVSROOT.
#
# The "-a" option specifies an alias.  An alias is interpreted as if
# everything on the right of the "-a" had been typed on the command line.
#
# You can encode a module within a module by using the special '&'
# character to interpose another module into the current module.  This
# can be useful for creating a module that consists of many directories
# spread out over the entire source repository.
#
CVSROOT -a CVSROOT
cvs-1.11.1p1 -a cvs_1.11.1p1
cvs -a cvs-1.11.1p1
```

4.7 Checking Out a Module – Creating a Working Directory

In order to add files or folders to a CVS repository or to edit or view existing files, the first step is to check out a working copy of one or more modules in the repository. The working copy is referred to as the working directory.

To checkout a module from the repository, the name of the module must be known. If the name of the module is not known, select **Get the cvs modules** from the WinCvs **Admin->Macros admin** menu. This command will show the list of modules in the WinCvs Output Pane as shown here:

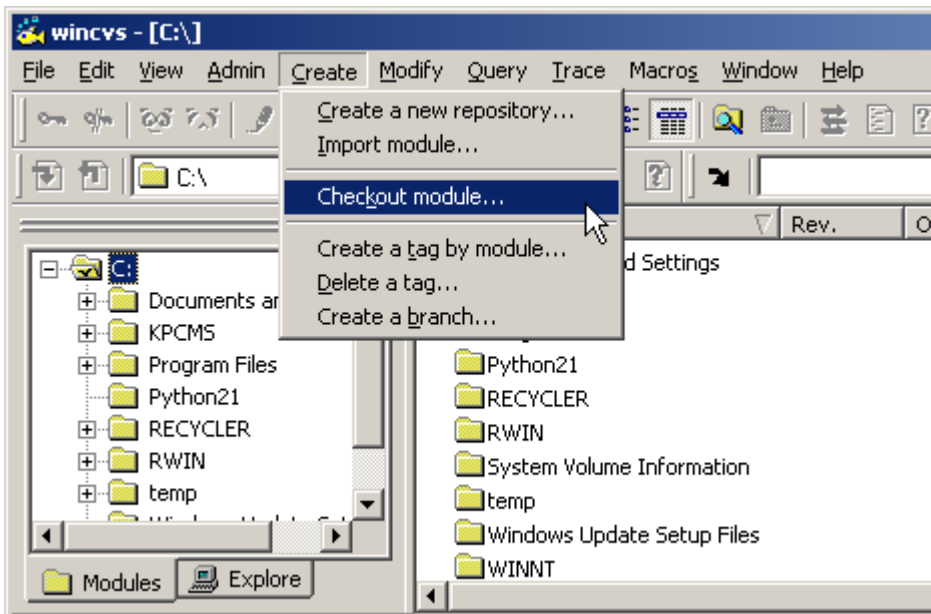
```
*****CVS exited normally with code 0*****
```

```
CVSROOT
  -a CVSROOT
cvs
  -a cvs_1.11.1p1
cvs-1.11.1p1
  -a cvs_1.11.1p1
```

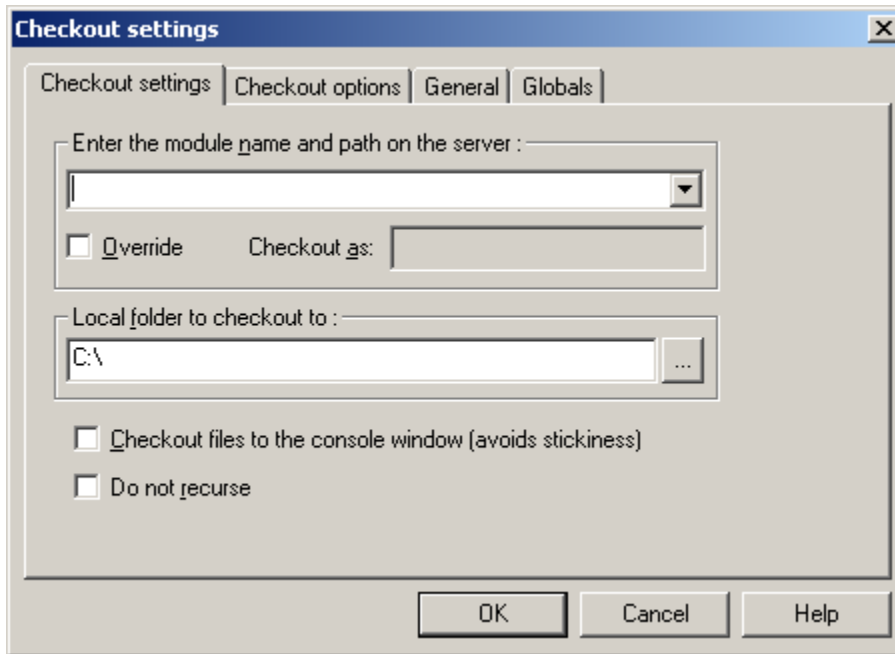
The field on the first line of each entry is the name or alias that can be used to checkout a module from the repository. The second line of each entry specifies the name of the root folder for the module relative to the repository root or an alias to another module.

It is very important to note at this time that the list of modules displayed by the **Get the cvs modules** command is obtained from the CVS **modules** file. Modules not defined in the **modules** file will not be listed in response to the **Get the cvs modules** command. See Section 4.6.3 for information on adding entries to the modules file. Modules not defined in the modules file can still be checked out however, as long as the name is known. One way of finding the list of available modules is to display the contents of the root directory of the repository. The name of any directory in the repository root directory can be used as a module name.

Once the name of the module is known, it can be checked out to the working directory selecting **Checkout module...** from the WinCvs **Create** menu as shown here:



After selecting **Checkout module...**, the **Checkout settings** dialog will be displayed as shown here:



Enter the name of the module to be checked out or select from the pull-down list. Note that the pull-down list is NOT a list of the available modules in the repository. It is just a history of modules that have been checked out previously. Complete modules can be checked out by specifying the top level name of the module such as `cvsg-1.11.1p1`. Partial modules can be checked out by specifying a path relative to the repository root such as `cvsg-1.11.1p1/src`.

By default, a module will be checked out to a newly created folder with the same name as the module. If a folder name other than the default is desired, check the **Override** box and specify the folder name in the **Checkout as** field.

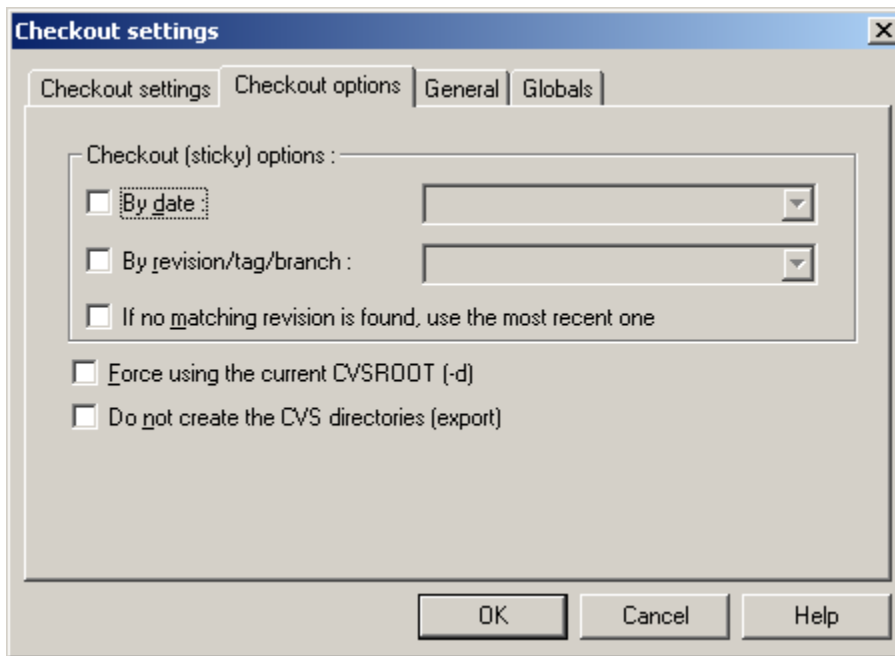
Specify the **Local folder to checkout to** by entering the folder name or using the browse button. This field specifies the **parent** folder in which the folder for the module checkout will be created.

The **Checkout files to the console window** option can be used when a small number of relatively small files needs to be viewed but not written to the local disk. If this option is checked, the file headers and contents will be displayed in the WinCvs Output Pane as shown here:

```
=====
Checking out CVSROOT/modules
RCS:  /store/sample/CVSROOT/modules,v
VERS: 1.8
*****
# Three different line formats are valid:
#   key  -a   aliases...
#   key [options] directory
#   key [options] directory files...
#
```

The **Do not recurse** option is useful if only one level of the repository is needed. By default, all levels of the repository below the selected module will be checked out to the working directory hierarchy.

The Checkout settings dialog has three panels in addition to the Checkout settings panel. The General and Globals panels are documented in Sections 4.2.1 and 4.2.2 of this manual. The Checkout options panel however, is unique to the Checkout settings dialog and is shown here:



The Checkout (sticky) options section lists three options that can be used to checkout revisions of files other than the most recent revision in the repository. Refer to Section 4.9 of the Cederqvist manual for more information on sticky tags.

Use the By date parameter to checkout a revision of each file in the module with a date no later than the specified date. Dates are in ISO8601 format such as 1972-09-04, 2001-10-25 20:05, 24 Sep 1980, 03/04/01, etc.

Use the By revision/tag/branch field to checkout a specific revision of each file in the module based on the specified revision number, tag, or branch. Note that revision numbers are always dot-separated numbers such as 1.1 or 1.2.2.4 whereas tags and branches always begin with a letter.

Checking the last box in the sticky options section tells CVS to checkout the most recent revision of any file that does not match the specified date or revision/tag/branch criteria. If this option is not checked, no revision will be checked out for files that do not match the specified criteria.

The Force using the current CVSROOT option does not seem to have any effect.

When a checkout is not intended for use as a working directory, CVS refers to the operation an export rather than a checkout. An export differs from a checkout mainly in that the CVS administrative folders are not created on a checkout. To create an exported hierarchy rather than the default checkout, check the box labeled Do not create the CVS directories (export).

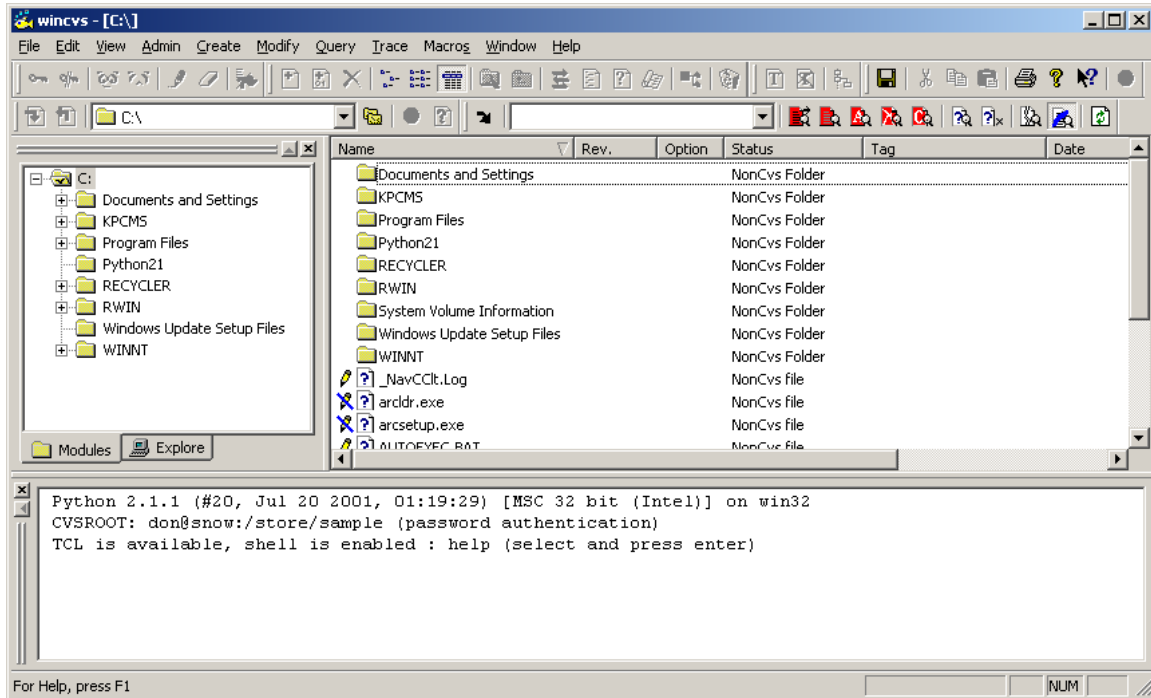
When the settings on all panels of the Checkout settings are complete, click the **OK** button to begin the checkout operation. Messages similar to the following will be displayed in the CVS Output Pane:

```
cvs checkout -P cvs-1.11.1p1 (in directory D:\cvs)
cvs server: Updating cvs-1.11.1p1
U cvs-1.11.1p1/.cvsignore
```

```
*****CVS exited normally with code 0*****
```

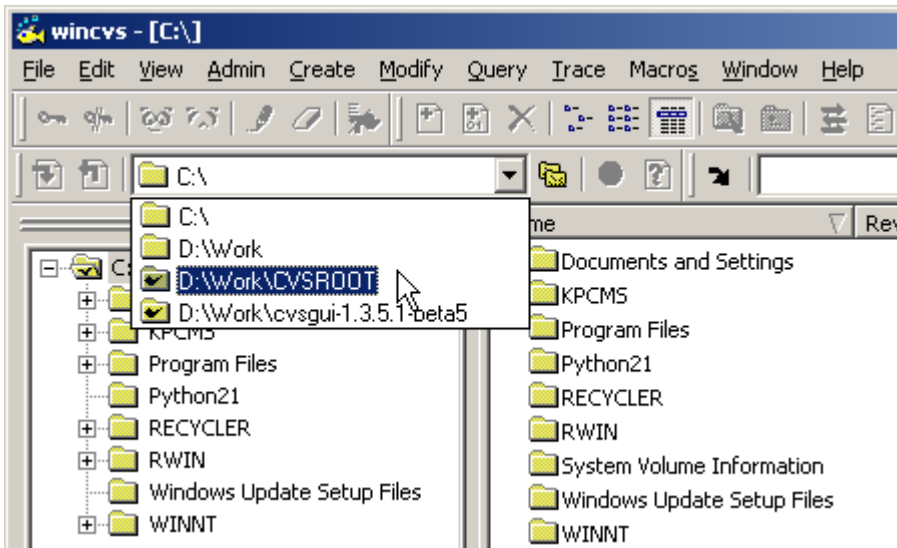
4.8 Changing the Browse Location

As noted in Section 4.1, the default browse location for the WinCvs Workspace and Files (left and right) panes is the root folder of the system drive (C:\) as shown again here:

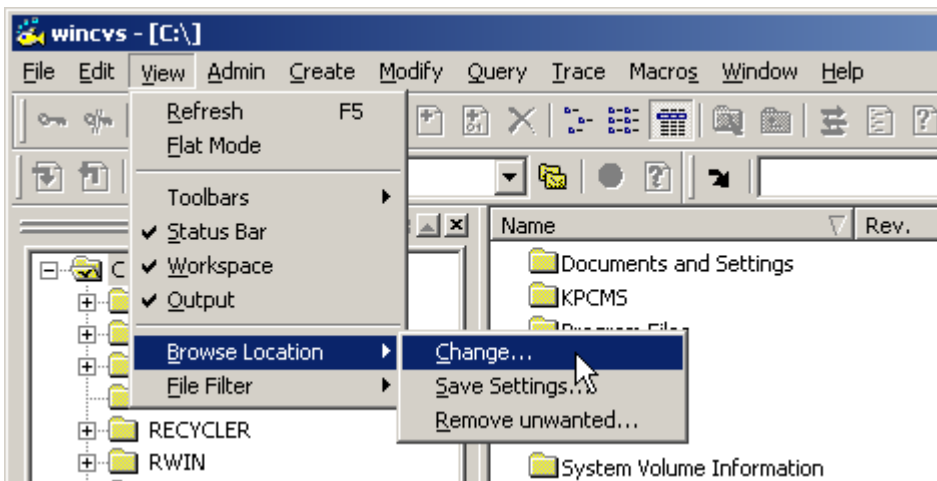


One useful way to make use of the browse location feature is to create a folder such as C:\Project-A to be the parent folder of all working directories for modules to be checked out from repository A. This folder can then be set as the browse location, and navigation to the various modules within repository A can be done from there.

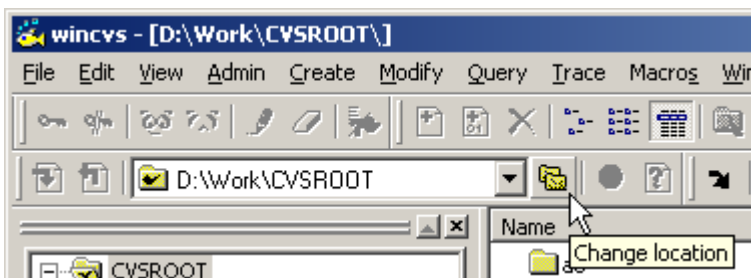
The browse location can be changed from the Browse toolbar by selecting from the pull-down history as shown here:



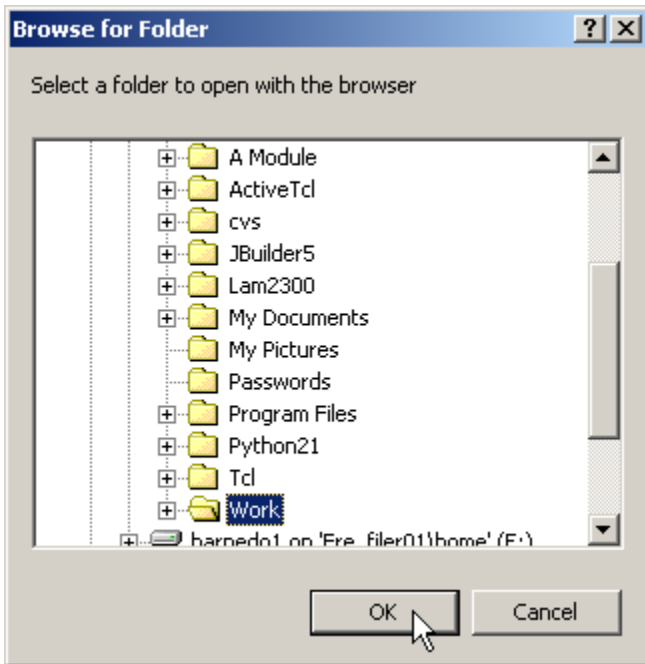
Alternatively, the browse location can be changed by selecting the **View->Browse Location->Change...** command from the menu as shown here:



Finally, the browse location can be changed from the Browse toolbar by selecting the double folder icon as shown here:

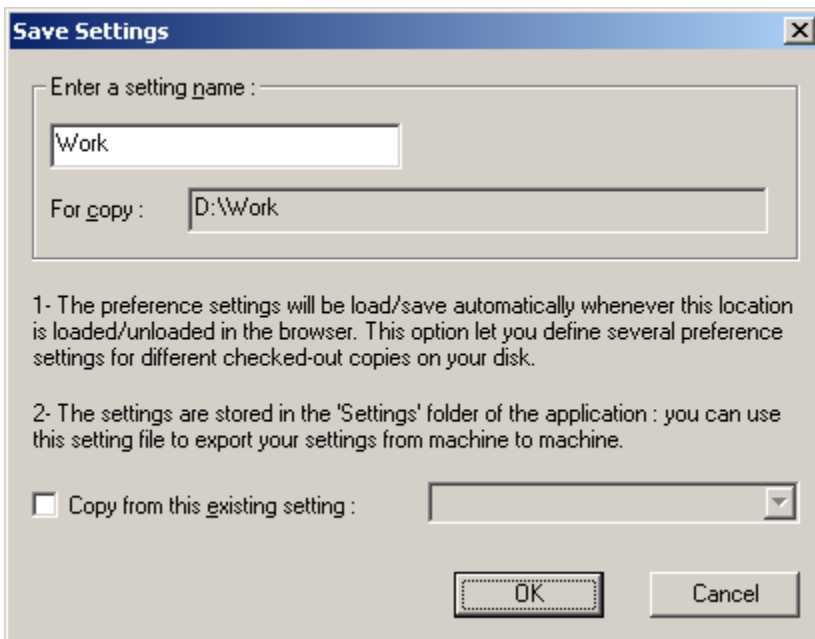


Selecting the Change location icon from the toolbar or selecting the Change location command from the menu opens the Browse for Folder dialog as shown here:



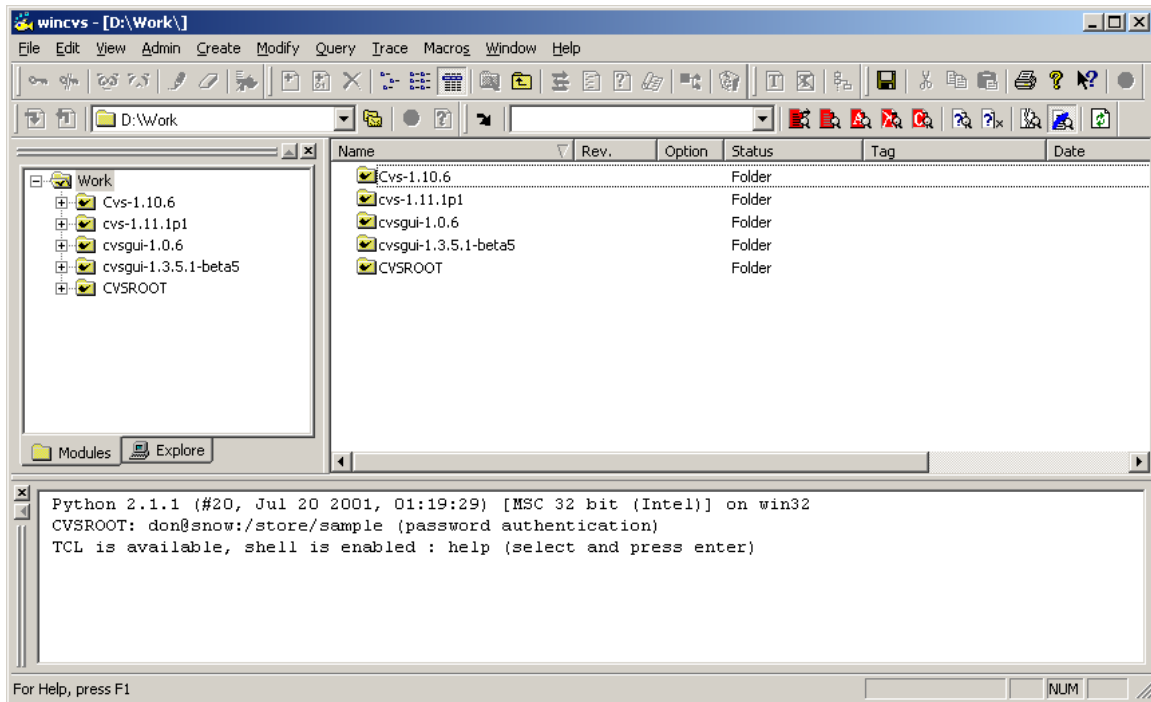
Browse to the desired folder and click the **OK** button to select the folder.

The first time a folder is selected as the browse location, the Save settings panel will be displayed as shown here:



Click **OK** save the current WinCvs Preferences for use when accessing this folder. An advanced option exists to copy from another previously saved setting if desired.

After changing the browse location, the WinCvs Workspace and File display panes will display the selected folder as shown in this example:



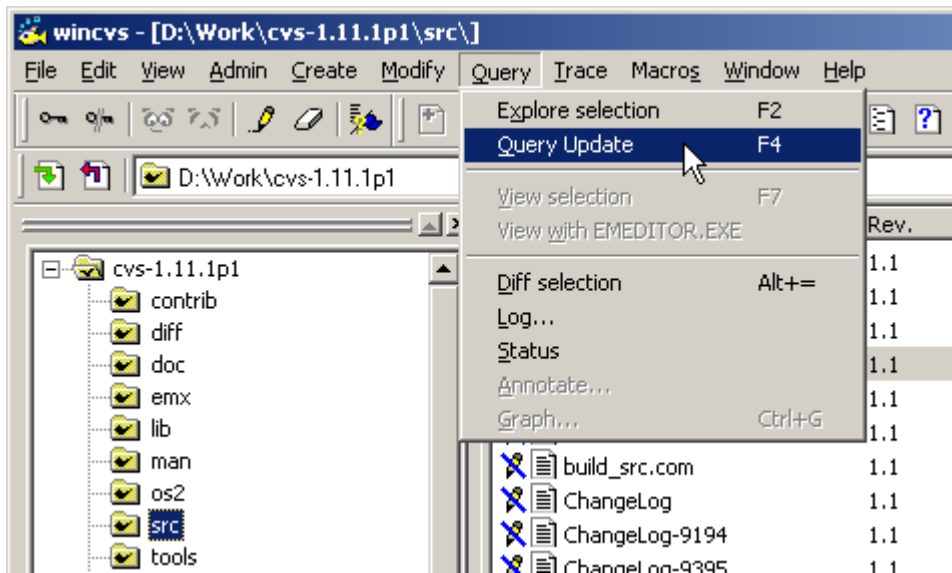
4.9 Updating a Working Directory

Once a working directory has been created, files may become out of date as other developers check in modifications from their own working directories. The CVS **update** command is provided to allow a working directory to be updated to the latest checked in sources. Another common use of the **update** command is to switch between branches or tagged versions or to view a previous revision of a file (see Section 5.3).

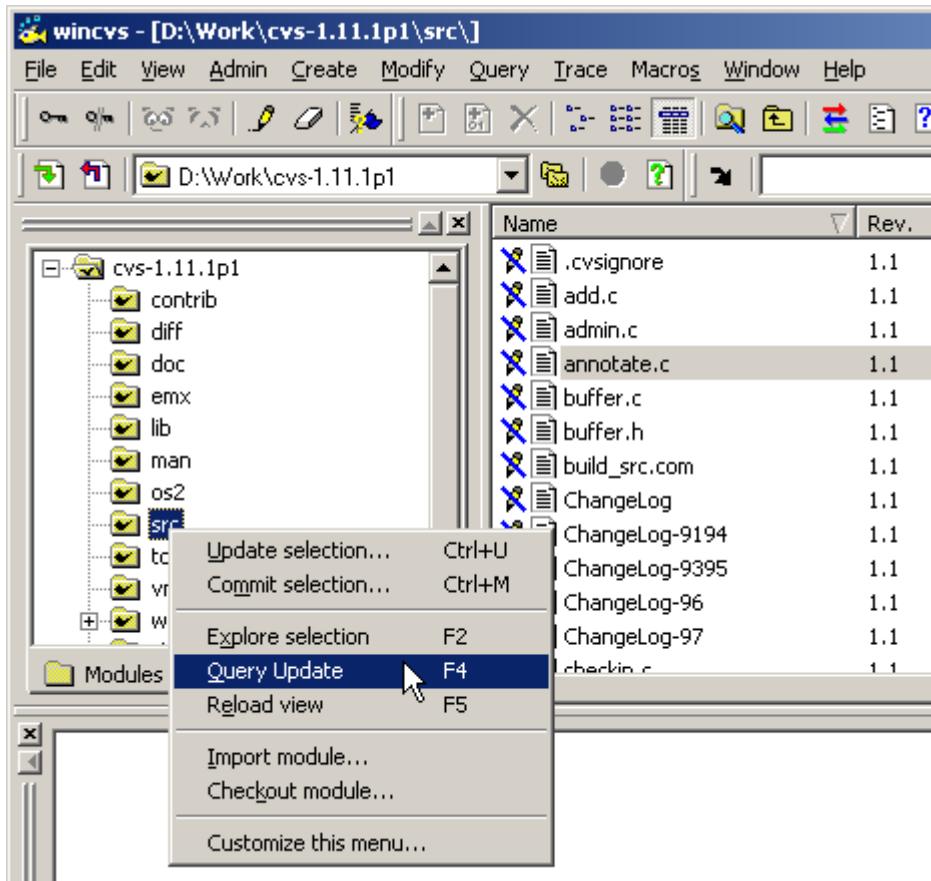
4.9.1 Running Query Update Prior to Update

Prior to running the **update** command, it may be useful to know which files are out of date. To find out what files are out of date, use the **query update** command. **Query update** lists the actions that will occur if the **update** command is run on the selected file or folder.

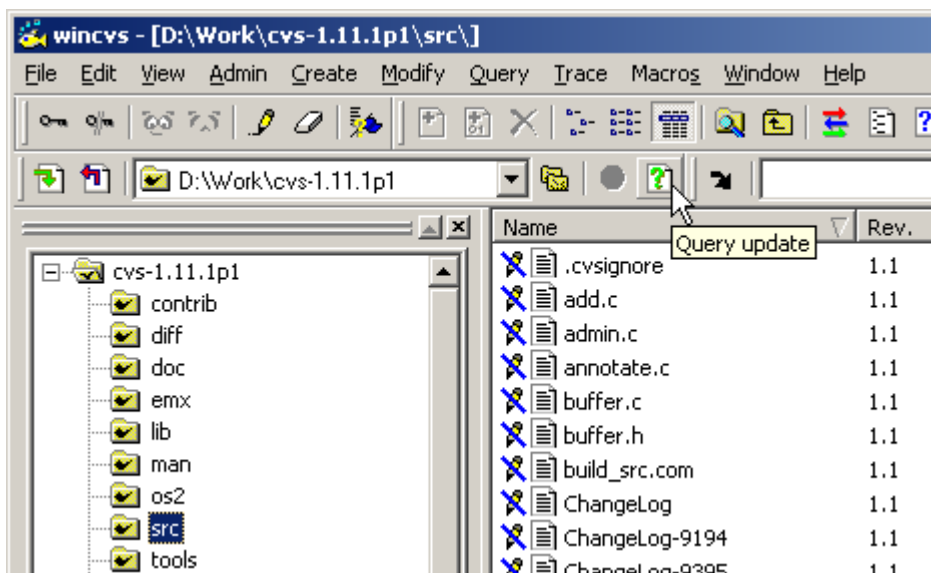
To run **query update** on a file or folder, select the file or folder and select **Query Update** from the **Query** menu as shown here:



The **query update** command is also available from the context menu by using right-mouse click on the desired file or folder as shown here:



The **query update** command is also available from using the icon on the toolbar as shown here:



When **query update** is run using any of the above methods, the result will be text output in the Output Pane as shown here:

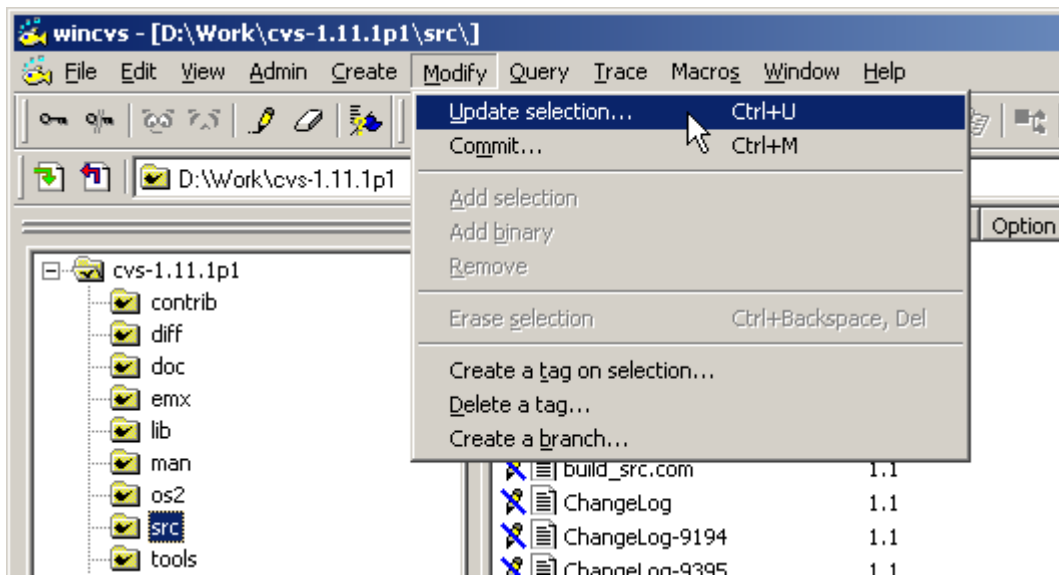
```
cvs -n update -P (in directory D:\Work\cvs-1.11.1p1\src\)  
cvs server: Updating .  
U ChangeLog  
U buffer.c  
U buffer.h  
U client.c  
U import.c  
  
*****CVS exited normally with code 0*****
```

The 'U' characters in the above example indicate that a new revision of the 5 listed files will be checked out from the repository if the **update** command is run on the selected folder. Refer to Appendix A.16.2 of the Cederqvist manual for further information.

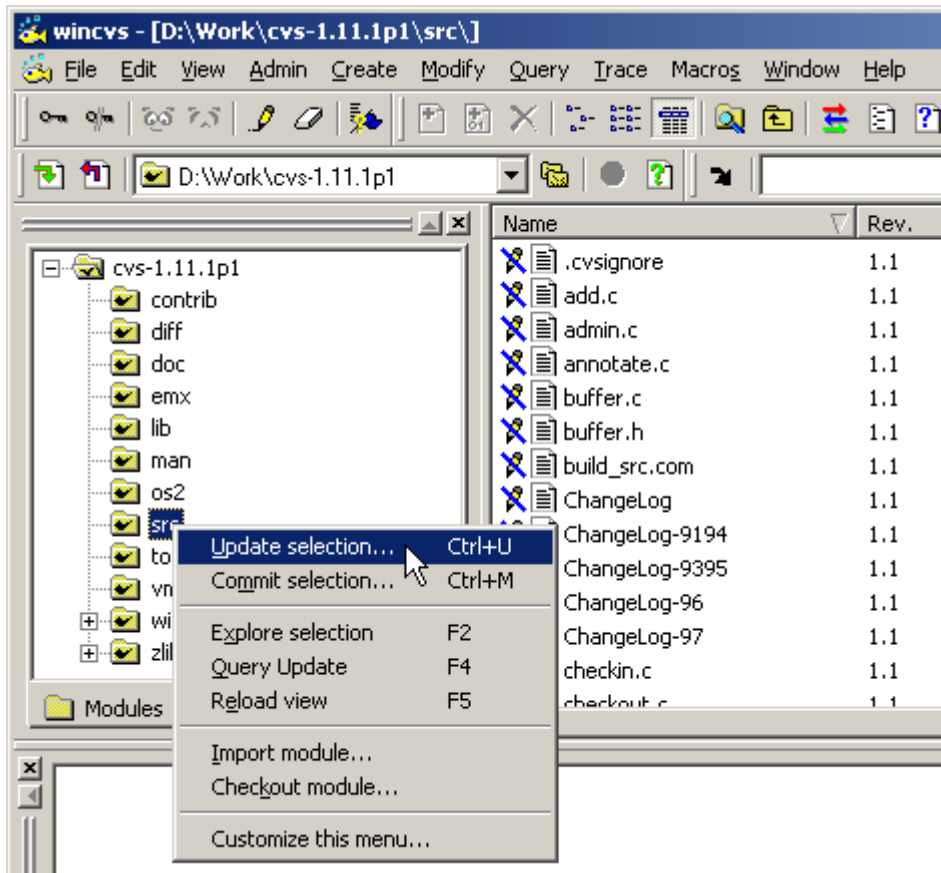
4.9.2 Running the Update Command

The **update** command is used to bring the working directory in sync with the repository. This can mean the latest revision of all files in the repository or specific revisions based on a revision number, date, or tag.

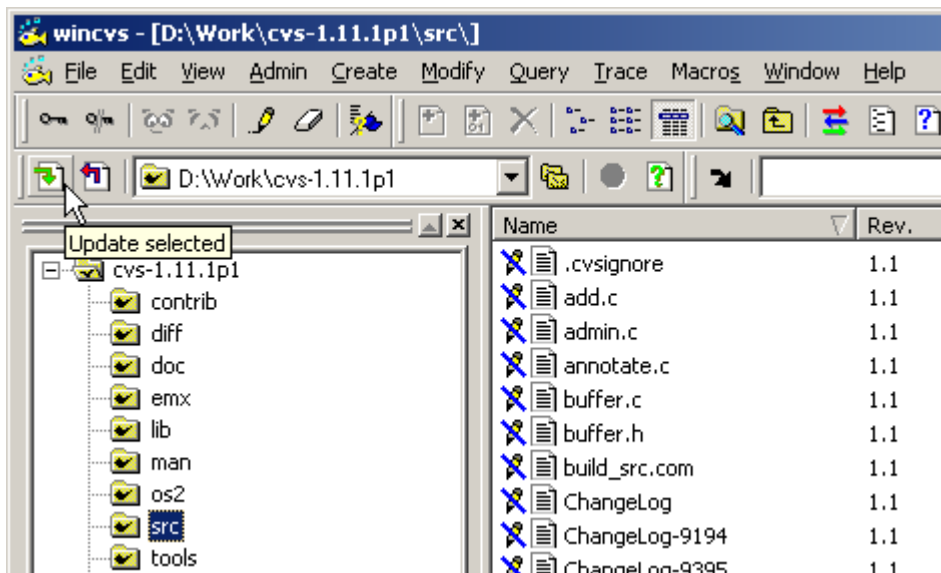
To run **update** on a folder or file, select the file or folder and select **Update selection...** from the **Modify** menu as shown here:



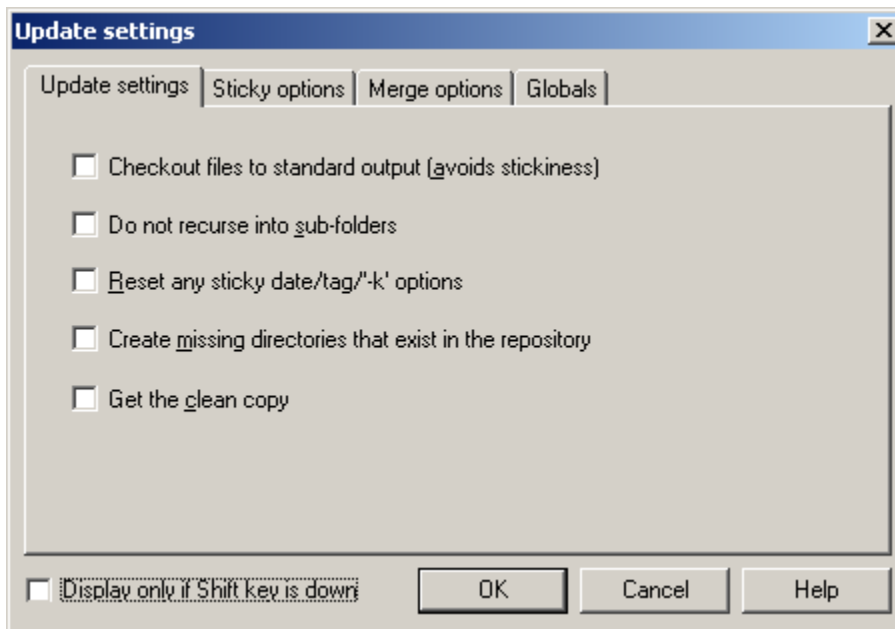
The **update** command is also available from the context menu by using right-mouse click on the desired file or folder as shown here:



The **update** command is also available as an icon on the toolbar as shown here:



When **update** is run using any of the above methods, Update settings dialog will be displayed unless configured to skip (see Section 4.2.4). If configured to skip, hold down the force key (SHIFT or CTRL) to force the dialog to be displayed as shown here:



The **Checkout files to standard output** option causes all files that are part of the **update** command to be printed in the WinCvs Output Pane instead of saving any files to disk. This option is rarely useful.

The **Do not recurse into sub-folders** option may be useful to update only a selected folder. The default in CVS is to update the selected folder and all sub-folders below it. This option has no effect when a single file is selected for update since recursion is not meaningful on a file.

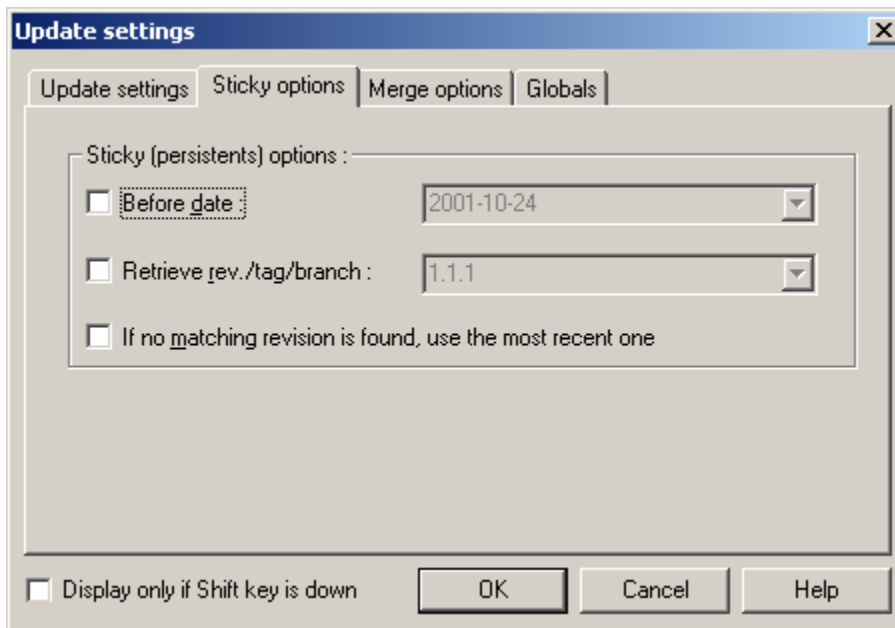
The **Reset any sticky date/tag/-k options** option is used to update back to the head trunk revision of all files. During development, files may be updated with sticky tags by revision number, date, or tag (including branch tags). Checking this box will restore the selected file or folder hierarchy to the latest trunk revision.

The **Create missing directories that exist in the repository** is useful to pick up new directories that have been added to the repository since the module was originally checked out. Without this option, only folders that exist in the current working directory will be updated.

Normally CVS will not overwrite files that have been modified in the working directory and not committed. If the **Get the clean copy** option box is checked, modified files will be replaced automatically while saving the modified file as *.#file.revision*.

The **Display only if Shift key is down** box will configure the Update settings dialog not to display unless the force key (see Section 4.2.4) is held down.

The second tab of the Update settings dialog is the Sticky options panel. This panel has three options as shown here:



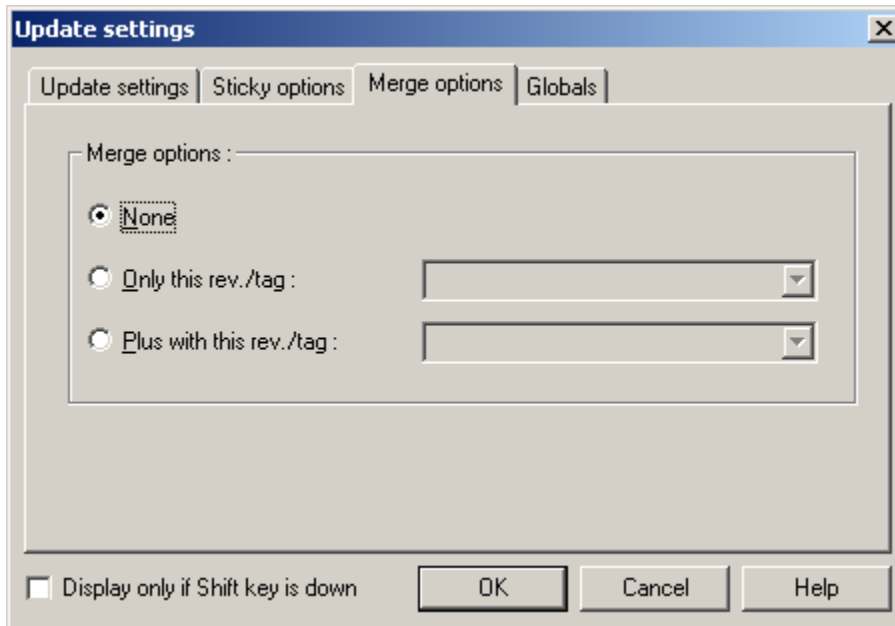
If the **Before date** box is checked, the newest revision of each file dated on or before the specified date will be retrieved from the repository.

If the **Retrieve rev./tag/branch** box is checked, only revisions matching the specified revision number, tag, or branch will be retrieved from the repository.

If the **If no matching revision is found, use the most recent one** box is checked, the head version of the repository trunk will be checked out in cases where the date and/or rev/tag/branch requirements do not match any files. If this box is not checked, files that have no matching revisions will not have any revision retrieved from the repository.

Refer to Section 4.9 of the Cederqvist manual for more information on sticky tags.









There are some advanced options related to merging available on the Merge options tab shown here:



This panel allows specification of 0, 1, or two merge options. These options are necessary when merging changes from different revisions or branches of a file or module. Refer to Section 5 of the Cederqvist manual for more information on merging.

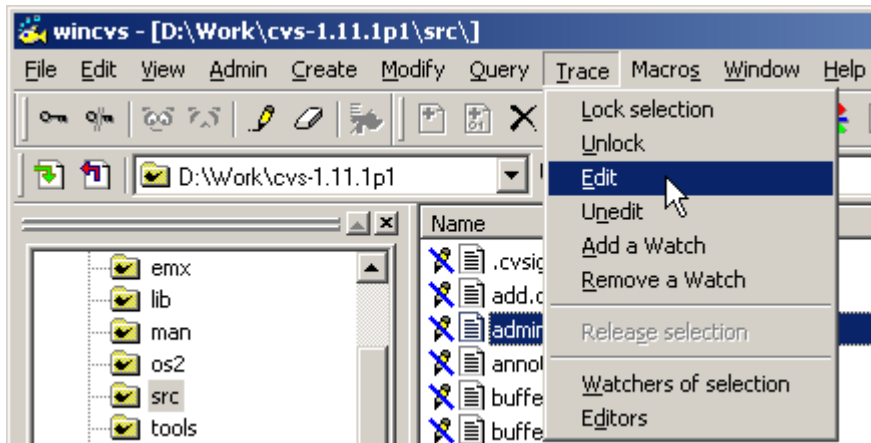
4.10 Modifying Files

By default, files are set to read-only when checking out a module or updating a working directory (although the default can be changed as explained in Section 4.2.2). In WinCvs, an icon to the left of the filename in the File browser window indicates the read/write status of a file. In the following example, the modules file has write access. All other files are read only:

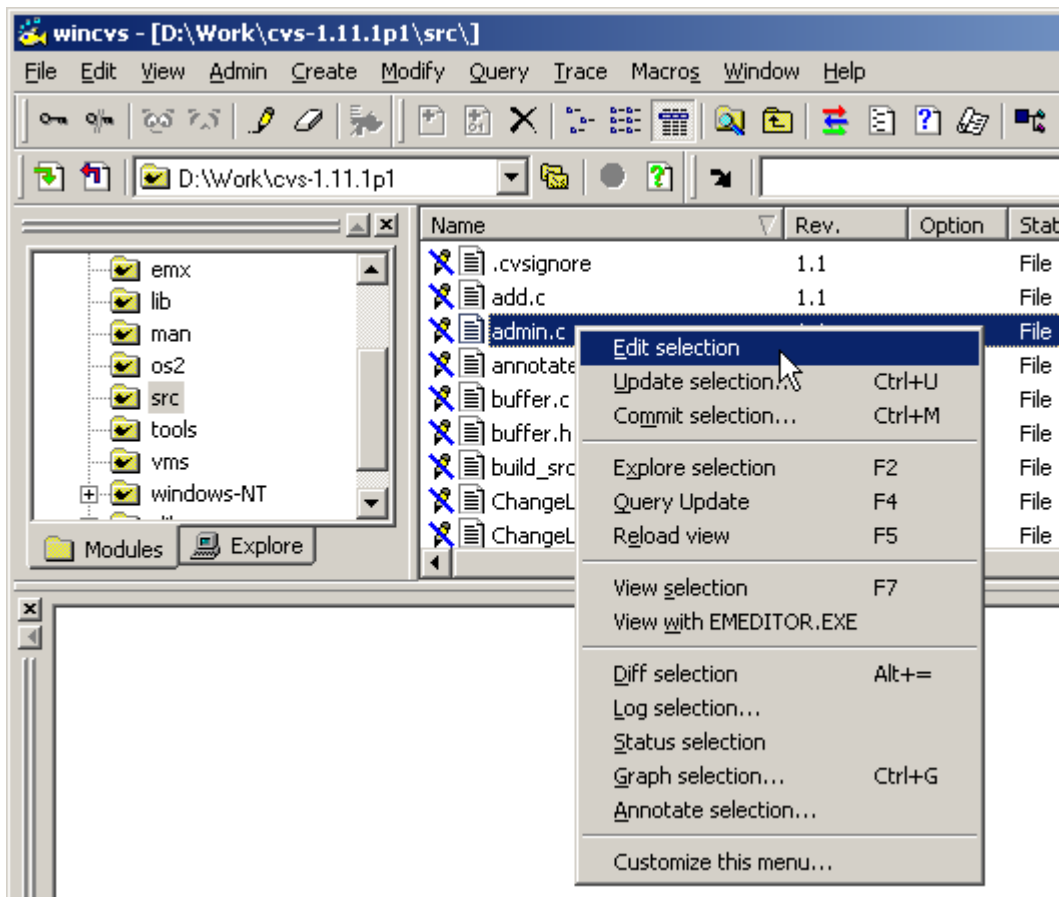
Name	Rev.	Option	Status	Tag	Date
 cvs wrappers	1.1		File		Tue Oct 23
 editinfo	1.1		File		Tue Oct 23
 loginfo	1.1		File		Tue Oct 23
 modules	1.8		File		Fri Oct 26 1
 notify	1.1		File		Tue Oct 23
 rcsinfo	1.1		File		Tue Oct 23
 taginfo	1.1		File		Tue Oct 23
 verifymsg	1.1		File		Tue Oct 23

4.10.1 Using the Edit Command to Add Write Access

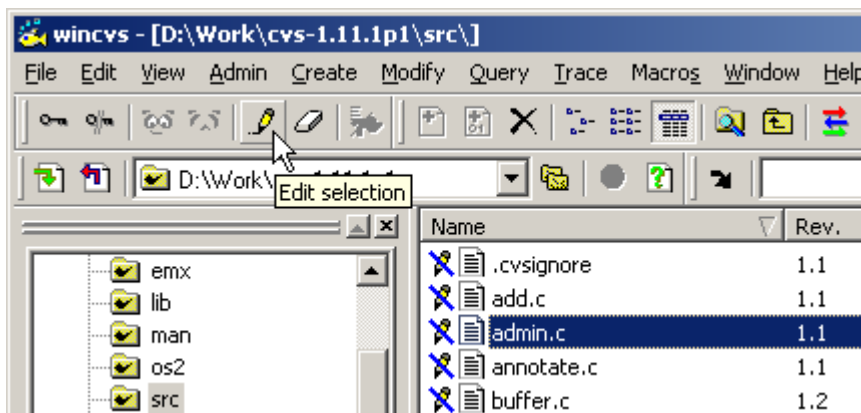
To modify a file, the **edit** command must first be used to add write access to the file. The **edit** command operates on a selected file or folder (recursively). The **edit** command can be invoked from the WinCvs menu by selecting **Edit** from the **Trace** menu as shown here:



The **edit** command can also be invoked from the context menu by using right-mouse click on the desired file or folder as shown below. Note that by default, the **Edit selection** entry does not appear in the context menu. To add it, select the **Customize this menu...** entry at the bottom of the context menu.



The **edit** command is also available as an icon on the toolbar as shown here:



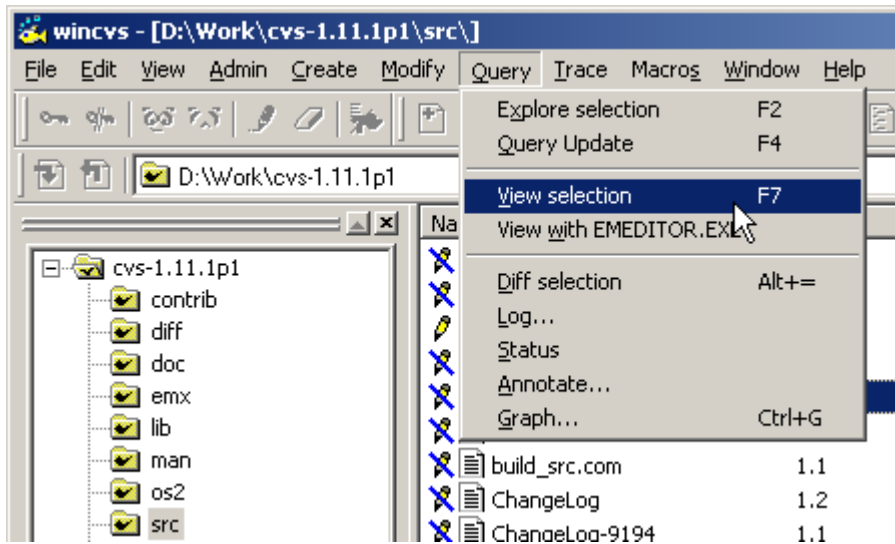
After write access is added to a file, it can be modified using any available editor. Double click on the file or use the **view** command to open the editor. Viewing files is described in the following section.

4.10.2 Viewing or Editing Files from WinCvs

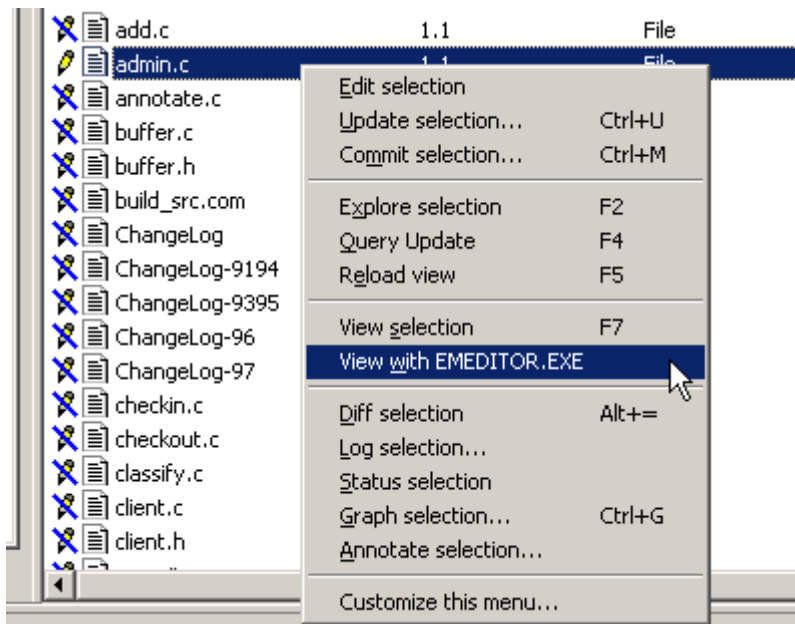
To open a file for modification or viewing in WinCvs, an application external to WinCvs is required. To specify which application to use in opening a file, refer to the WinCvs Preferences documentation in Section 4.2.3.

The easiest way to open a file is to double click on the filename in the WinCvs File pane. The Windows-associated editor or the user specified editor will be used to open the file based on the WinCvs Preferences.

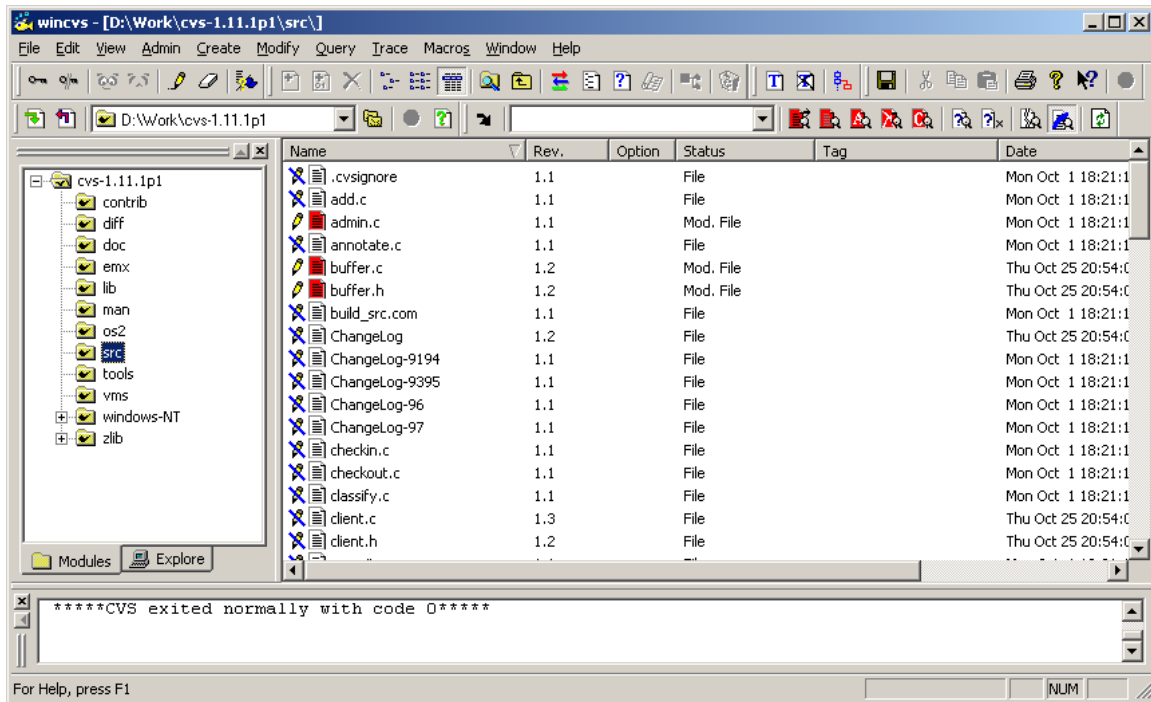
WinCvs also provides two versions of the **view** command in the **Query** menu as shown below. The **View selection** entry is equivalent to a double-click and will open the file with the Windows-associated editor or the user specified editor based on the WinCvs Preferences. The **View with ...** entry forces the file to be opened with the user specified editor listed on the menu:



The **view** commands are available by using right-mouse click from the context menu as shown here:



After modifying a file, the Icon displayed in the WinCvs file window will turn red and the Status field will change to Mod. File as shown the the following example where admin.c, buffer.c and buffer.h have been modified:

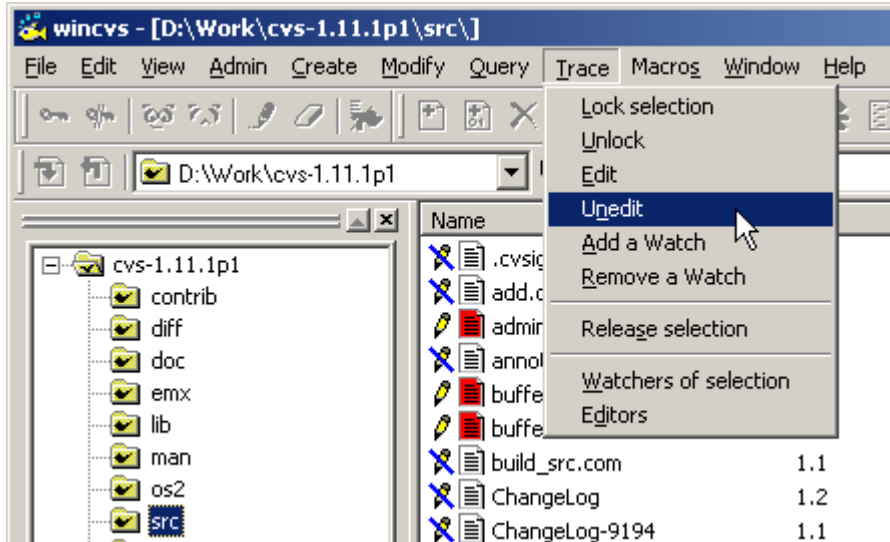


After a file is modified, the next logical step in CVS is to commit the changes to the repository. Refer to Section 4.12 for information about committing files. Note that thorough testing is normally done prior to committing any modifications.

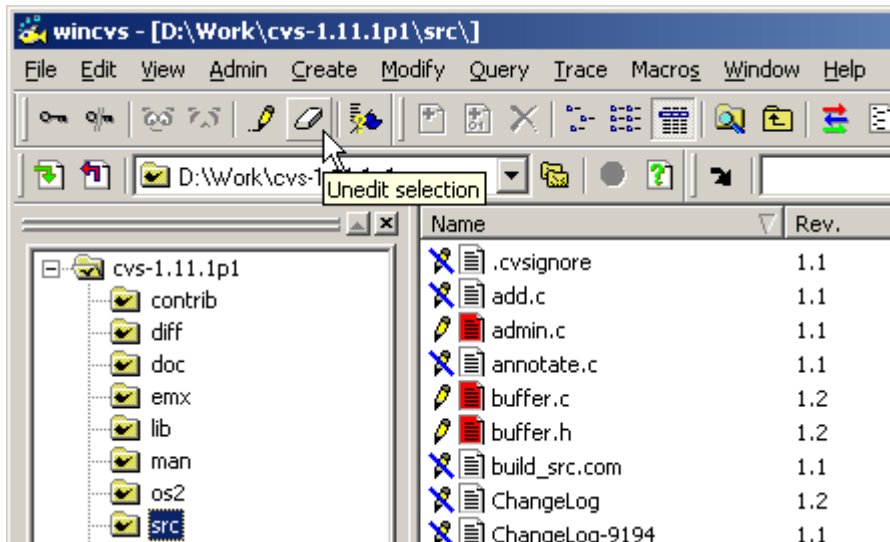
4.10.3 Using the Unedit Command to Remove Write Access

After a file has been given write access with the **edit** command, there are situations where a developer will want to give up write access without committing any changes to the repository. The **unedit** command is provided for this purpose. The **unedit** command would be used, for example, if the **edit** command was accidentally invoked on the wrong file or folder. The **unedit** command reverses the effect of the **edit** command.

The **unedit** command operates on a selected file or folder (recursively). The **unedit** command can be invoked from the WinCvs menu by selecting **Unedit** from the **Trace** menu as shown here:

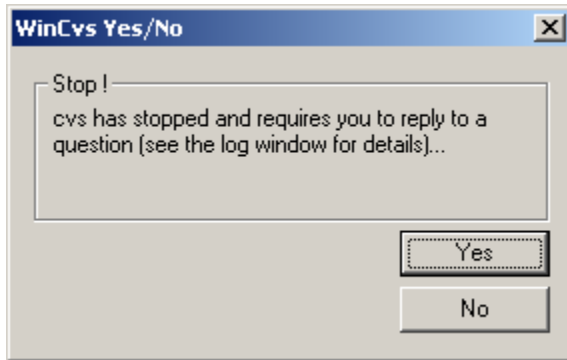


The **unedit** command is also available as an icon on the toolbar as shown here:



The **unedit** command can also be invoked from the context menu by using right-mouse click on the desired file or folder. By default, the **Unedit selection** entry does not appear in the context menu. To add it, select the **Customize this menu...** entry at the bottom of the context menu.

When the **unedit** command is invoked on a file that has been locally modified, any local changes to the file will be discarded. As a precaution, WinCvs will pop up the following dialog for each such file:



This dialog indicates that there is a CVS message in the WinCvs Output Pane that needs a response. The message will indicate which file has been modified and ask if changes should be reverted. Example messages for an **unedit** command on a modified file admin.c are shown here:

```
cvs unedit (in directory D:\Work\cvs-1.11.1p1\src\  
admin.c has been modified; revert changes?
```

Click on the **Yes** button to discard the changes and revert to the original revision. Click **No** to skip the **unedit** operation for the listed file.

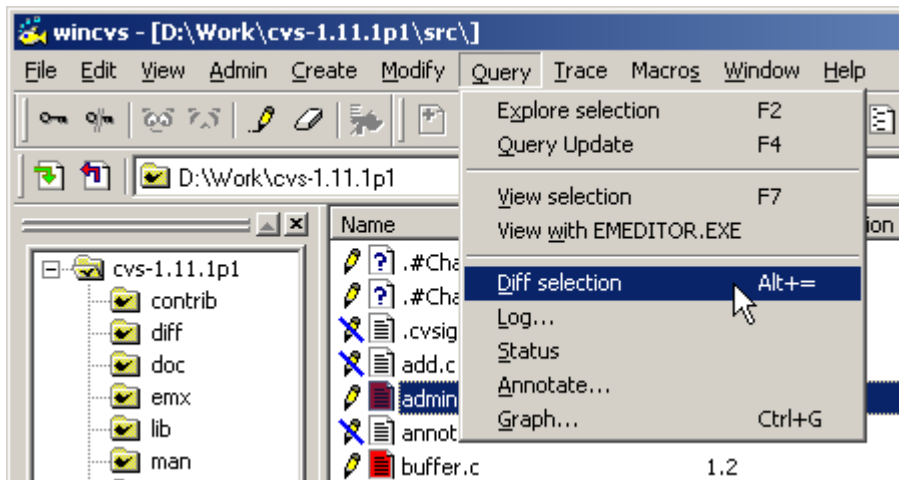
4.11 Checking Diffs Prior to Commit

It is often useful to run **diff** as a precaution prior to committing a file to the repository. This section describes the steps required to compare the version of a file in a working directory to the current revision in the repository. The differences can either be displayed as a standard text diff in the WinCvs status window or graphically (if an external diff program is selected as described in Section 4.2.3).

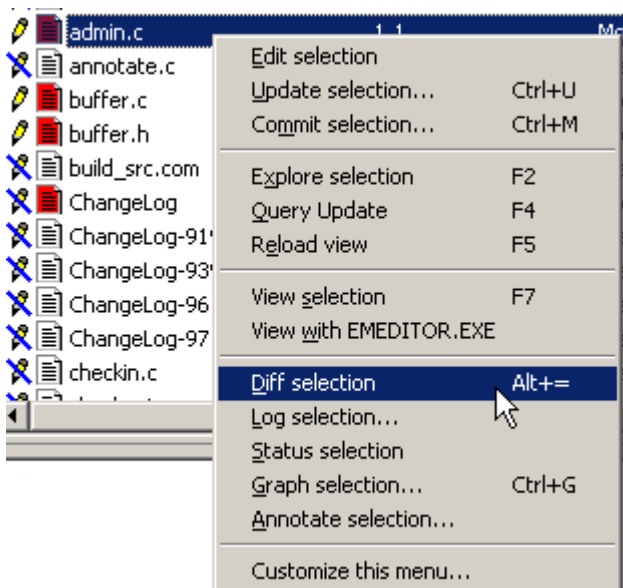
4.11.1 Generating a Text Diff

This section deals with the standard text **diff**. See the following section (Section 4.11.2) for an example of a graphical **diff**.

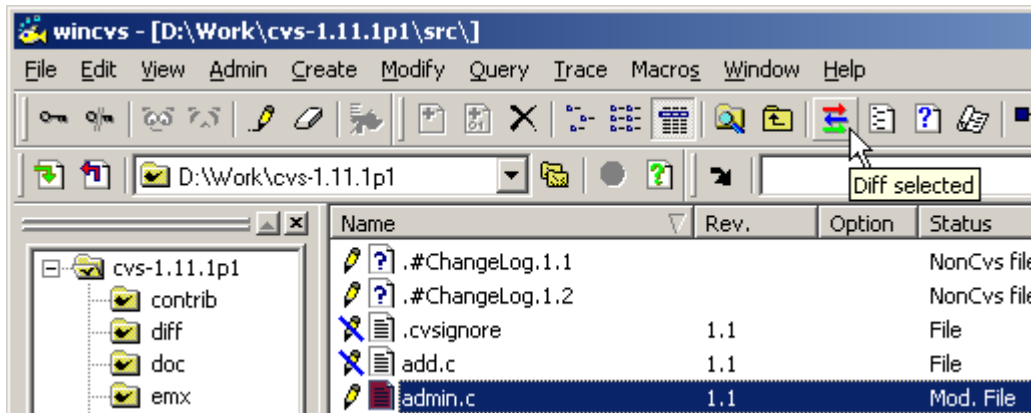
The **diff** command operates on a selected file or folder (recursively). The **diff** command can be invoked from the WinCvs menu by selecting **Diff selection** from the **Query** menu as shown here:



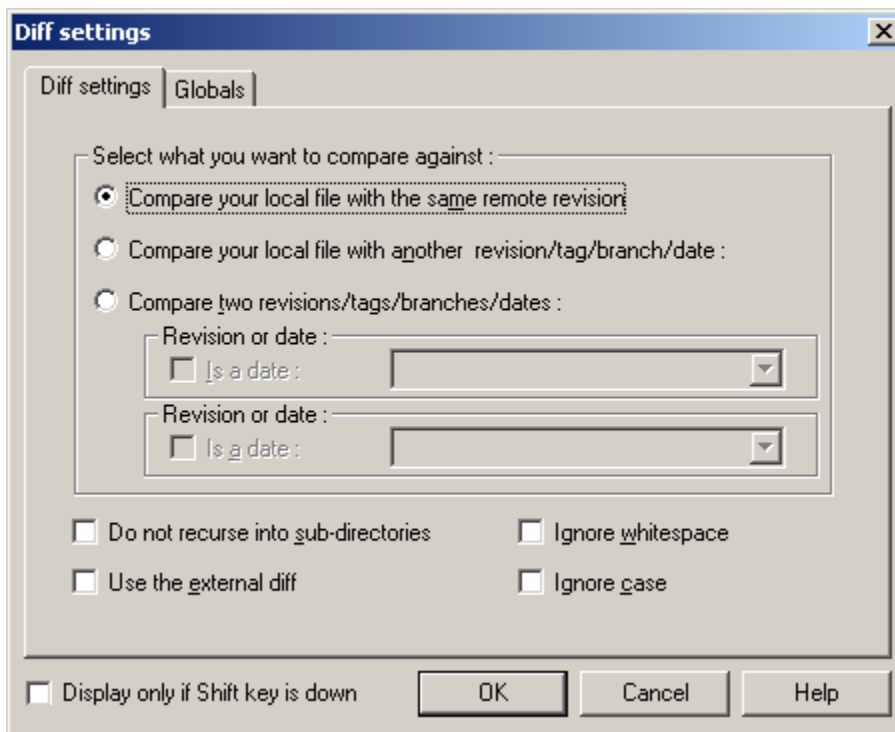
The **diff** command can also be invoked from the context menu by using right-mouse click on the desired file or folder as shown below:



The **diff** command is also available as an icon on the toolbar as shown here:



When **diff** is run using any of the above methods, **Diff settings** dialog will be displayed unless configured to **skip** (see Section 4.2.4). If configured to **skip**, hold down the force key (SHIFT or CTRL) to force the dialog to be displayed as shown here:



The **Diff settings** panel provides three variations of compare, but only the first type makes sense for this example (since we just want to compare a local revision to the repository). In addition, it may be desirable to ignore white space or case by checking the appropriate boxes. Make sure the first option is selected as shown above and then click the **OK** button.

The text version of the file differences (if any) will now be displayed in the WinCvs Output pane as shown here:

```
cvs diff admin.c (in directory D:\Work\cvs-1.11.1p1\src\  
Index: admin.c
```

```
=====  
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/src/admin.c,v  
retrieving revision 1.1  
diff -r1.1 admin.c  
913a914,921  
>  
> /*  
>  * Dummy function to test WinCvs  
>  */  
> void dummy()  
> {  
>     printf("WinCvs Test...\n");  
> }
```

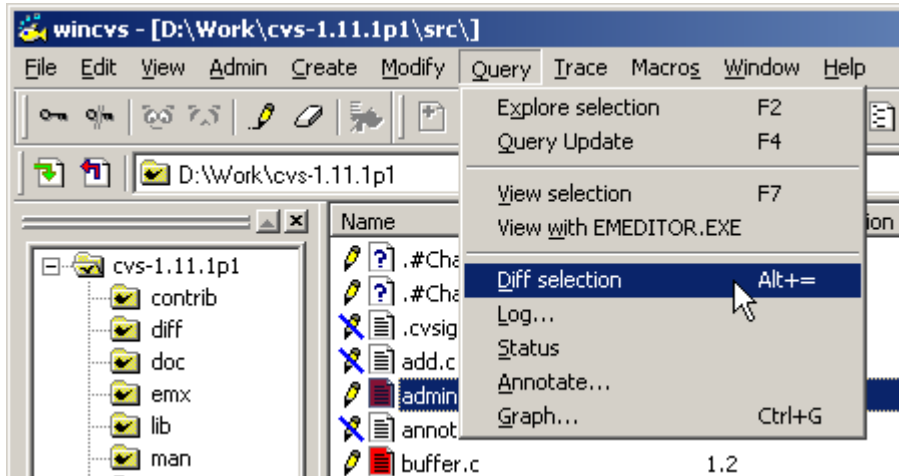
```
*****CVS exited normally with code 1*****
```

The text in the WinCvs Output Pane can be selected, saved to a file, or printed by making the Output Pane active (click anywhere in the Output Pane) and using the Edit and File menus. The Output Pane can also be cleared using **Erase All** from the **Edit** menu.

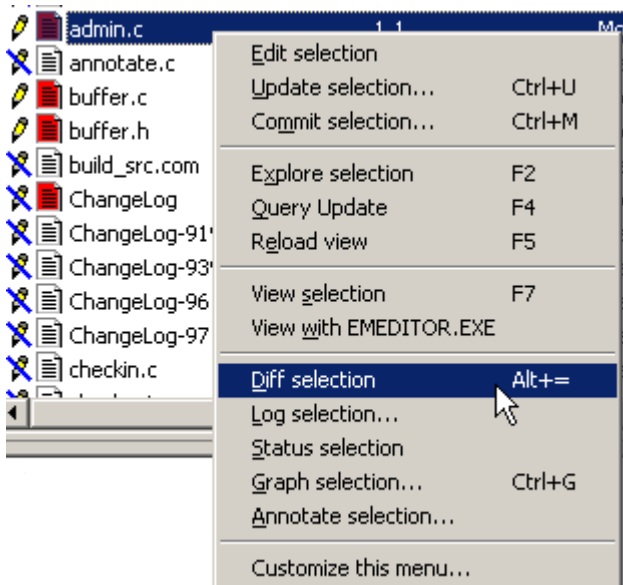
4.11.2 Generating a Graphical Diff

This section deals with graphical **diff**. See the preceding section (Section 4.11.1) for an example of a text **diff**.

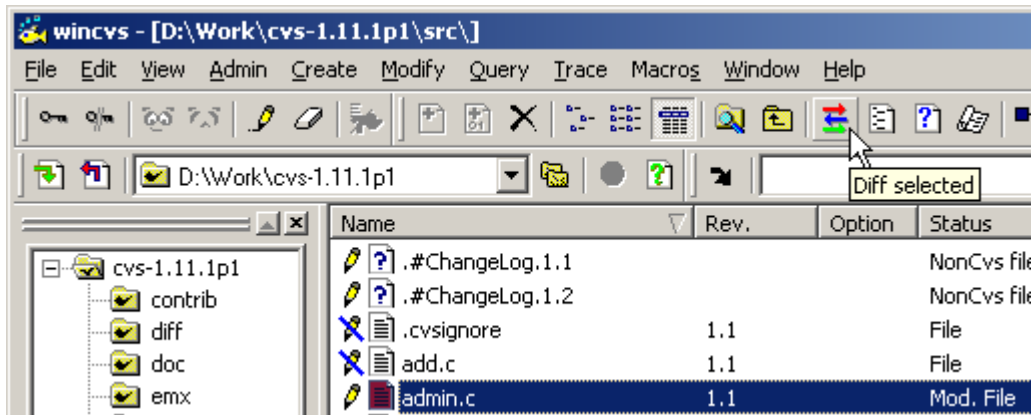
The **diff** command operates on a selected file or folder (recursively). The **diff** command can be invoked from the WinCvs menu by selecting **Diff selection** from the **Query** menu as shown here:



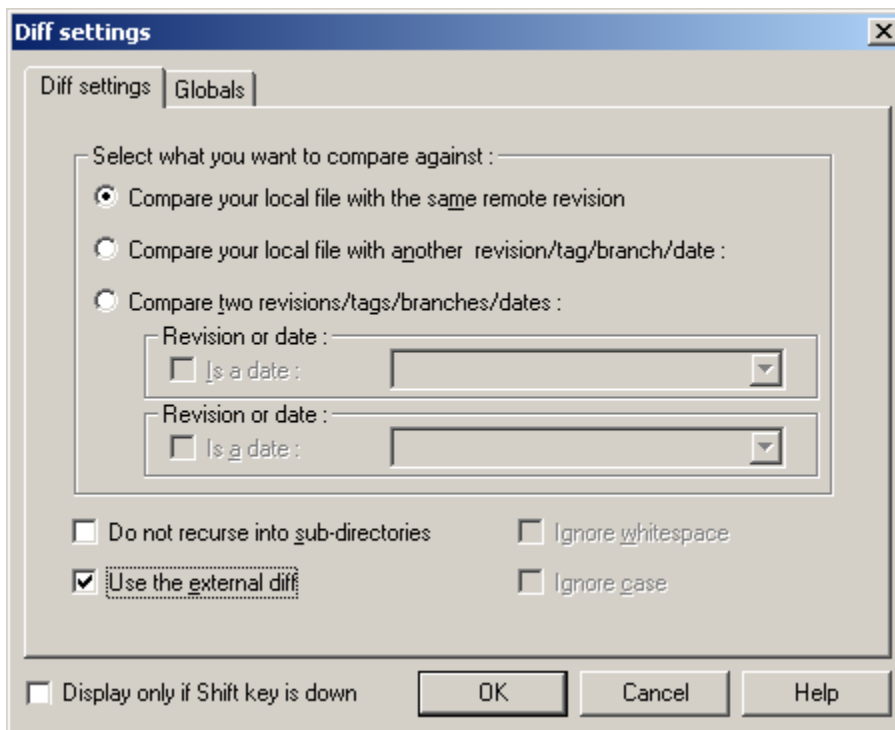
The **diff** command can also be invoked from the context menu by using right-mouse click on the desired file or folder as shown below:



The **diff** command is also available as an icon on the toolbar as shown here:



When **diff** is run using any of the above methods, **Diff settings** dialog will be displayed unless configured to **skip** (see Section 4.2.4). If configured to **skip**, hold down the force key (SHIFT or CTRL) to force the dialog to be displayed as shown here:



The **Diff settings** panel provides three variations of compare, but only the first type makes sense for this example (since we just want to compare a local revision to the repository).

To run the external diff rather than a text diff, check the Use the external diff box. Note that this option will have no effect if an external diff program has not been configured. See Section 4.2.3 for more information.

Make sure the options are selected as shown above and click the **OK** button.

The graphical diff will be displayed as in the following example. Note that in this example, ExamDiff Pro has been configured as the external editor:

```
ExamDiff Pro - C:\Documents and Settings\harpedo1\Local Settings\Temp\admin_orig_6.c | D:\Work\cvs-1.11.1p1\src\admin.c
Files Edit View Navigation Search Info
Diff 1: Delete lines 907-908 (left file) Skip
C:\Documents and Settings\harpedo1\Local Settings\Temp\admin_orig_6.c D:\Work\cvs-1.11.1p1\src\admin.c
892      exitfunc:
893      freevers_ts (&vers);
894      return status;
895  }
896  }
897  /*
898  /*
899  * Print a warm fuzzy message
900  */
901  /* ARGSUSED */
902  static Dtype
903  admin_dirproc (callerdat, dir, repos
904      void *callerdat;
905      char *dir;
906      char *repos;
907  char *update_dir;
908  List *entries;
909  {
910      if (!quiet)
911          error (0, 0, "Administrating %s"
912              return (R_PROCESS);
913  }
914
892      exitfunc:
893      freevers_ts (&vers);
894      return status;
895  }
896  }
897  /*
898  /*
899  * Print a warm fuzzy message
900  */
901  /* ARGSUSED */
902  static Dtype
903  admin_dirproc (callerdat, dir, repos, update_dir, e
904      void *callerdat;
905      char *dir;
906      char *repos;
907  {
908      if (!testing)
909          error (0, 0, "Administrating %s", update_dir);
910      return (TEST_TEST_CHANGE);
911  }
912  /*
913  /*
914  * Dummy function to test WinCvs
915  */
916  void dummy()
917  {
918      printf("WinCvs Test...\n");
919  }
920  }
```

Ln 1, Col 1 INS Ln 1, Col 1 INS

4 differences (12 lines + 2 words in changed lines) found

Added[8+0] Deleted[2+0] Changed[2] Changed words in changed[2]

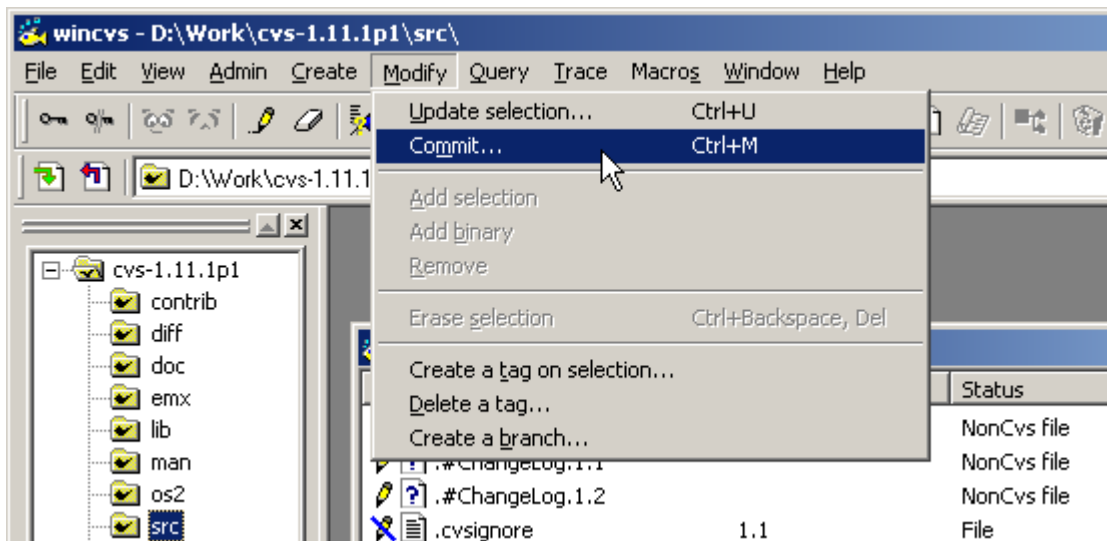
Notice that the diff program shown in the above example (ExamDiff Pro) has many options including the ability to customize colors, ignore white space, etc (from the **View->Options...** menu). There is also an option to save a UNIX style text diff (from the **File->Save Unix Diff File...** menu).

4.12 Committing Files and Folders

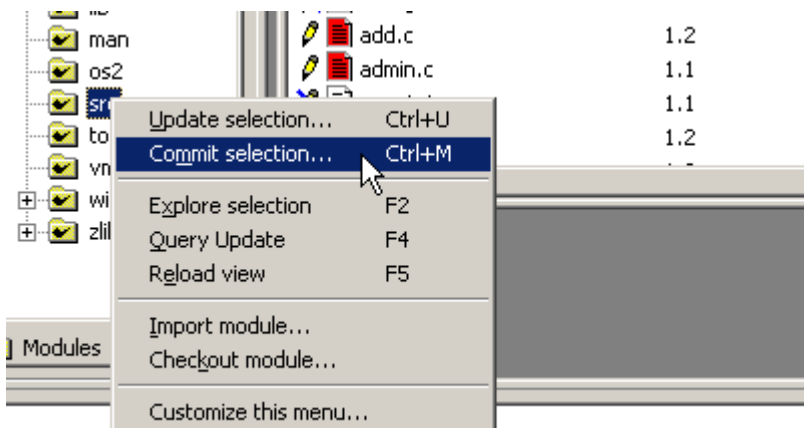
After one or more files have been edited or added, individual files or an entire hierarchy of files can be committed to the repository using the **commit** command. Prior to committing the files, it may be useful to use the **diff** command to verify the modifications that will be made to the repository. Refer to Section 4.11 for information about the **diff** command. The **diff** command can also be invoked from the **Commit settings** dialog as explained in this section.

The **commit** command is used to check files into the repository. It is important to note that **commit** requires that files to be committed be up to date with the current revisions in the repository. Also, files with sticky tags other than branch tags cannot be committed. This section contains examples of successful and unsuccessful commit operations.

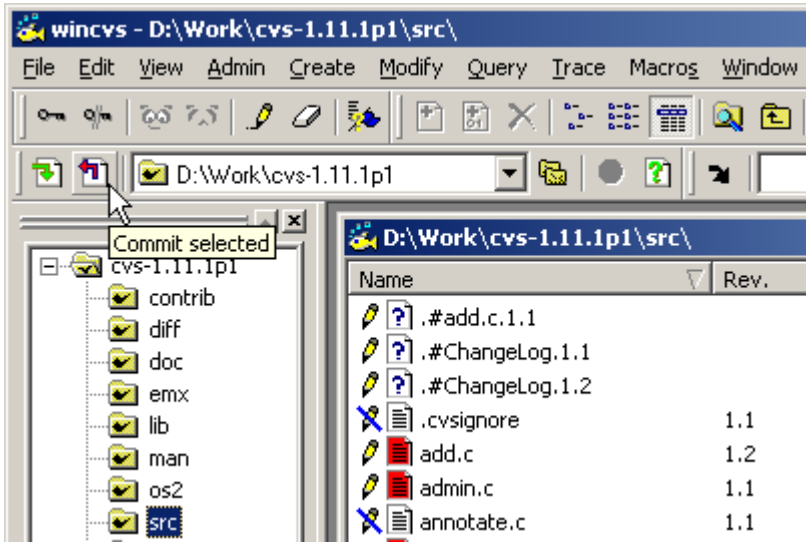
The **commit** command operates on a selected file or folder (recursively). The **commit** command can be invoked from the WinCvs menu by selecting **Commit** from the **Modify** menu as shown here:



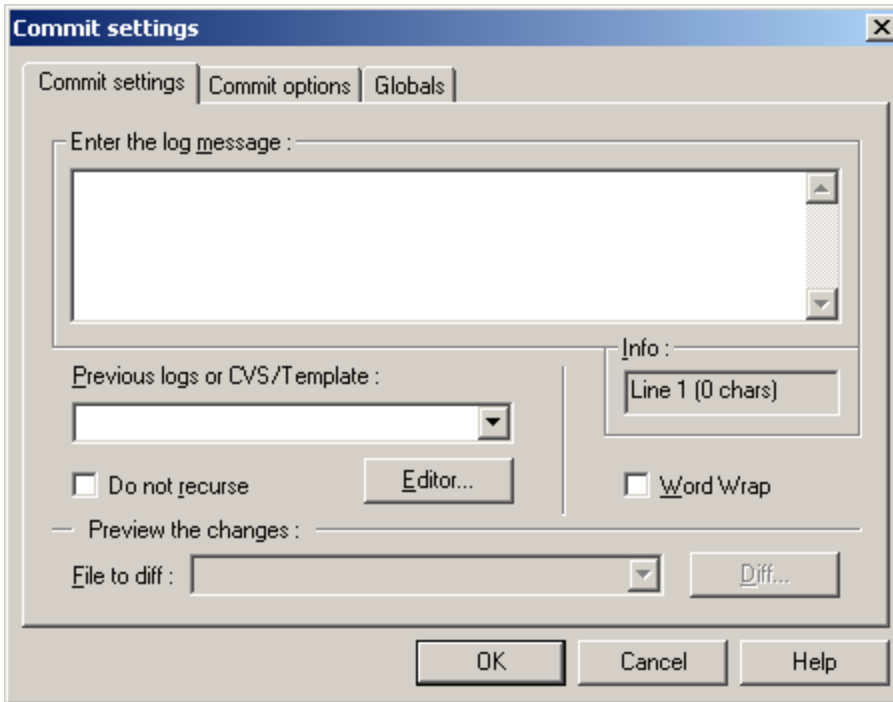
The **commit** command can also be invoked from the context menu by using right-mouse click on the desired file or folder as shown below:



The **commit** command is also available as an icon on the toolbar as shown here:



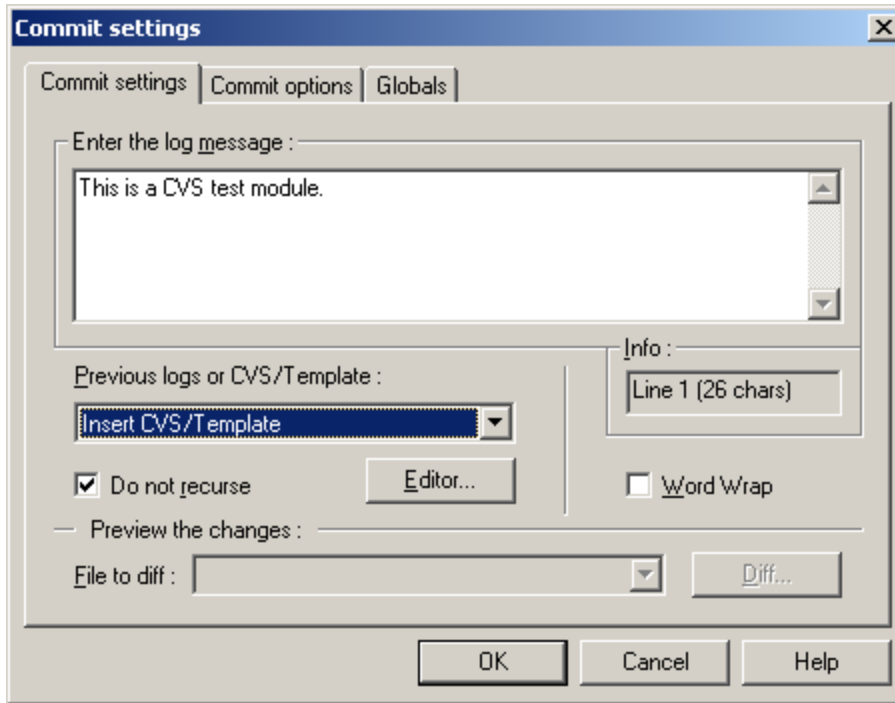
When **commit** is run using any of the above methods, the Commit settings dialog will be displayed as shown here:



As shown above, there are 3 tabs on the Commit settings dialog.

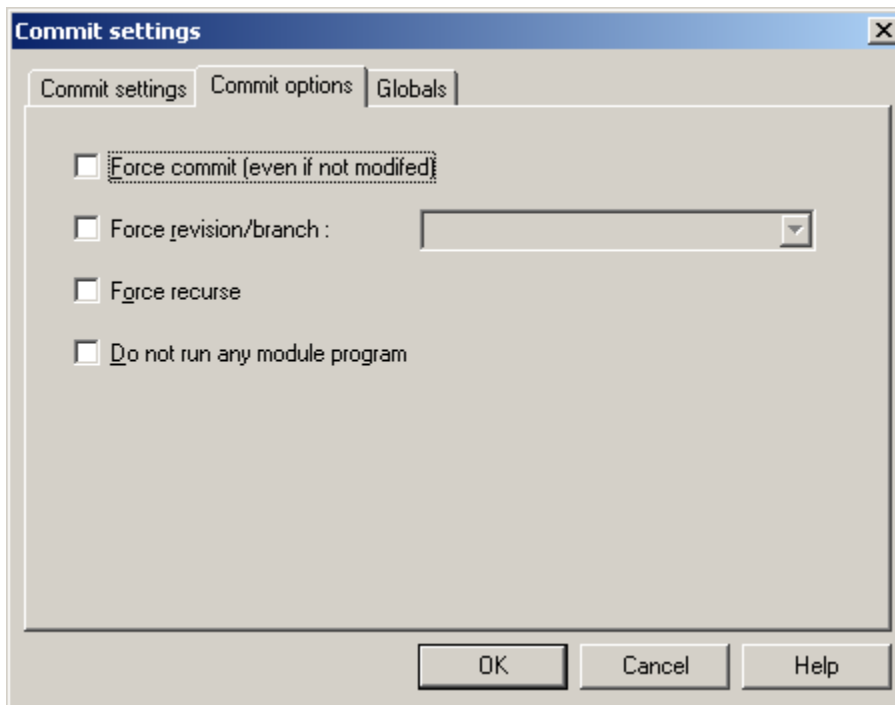
The Globals tab has the standard global options available for other commands as well as in the WinCvs Preferences and will not be explained here.

The main function of the Commit settings tab of the Commit settings dialog is to enter a log message for the **commit** operation. The log message can be entered directly in the log message window, selected from previous log messages or from the file CVS/Template (pulldown list), or typed into an editor with the **Editor...** button. In the following example, the message was copied from the CVS/Template file in the folder where the **commit** operation was started:



In addition to the log message, the Commit settings tab has a check box to inhibit the automatic recursion that is done when committing a folder. Also, when committing a file rather than a folder, the **Diff...** button can be used to run a diff to preview the changes being made to the repository.

The Commit options tab on the Commit settings dialog has several advanced options as shown here:



If the **force commit** box is checked, a new revision of the selected file or folder will be created even if no changes were made to any of the affected files. This option can be useful if CVS does not realize that the file was modified (due to a timestamp issue) or when forcing a revision or branch (see next option).

If the **force revision/branch** box is checked, the file will be checked into the specified branch (branch tags begin with upper or lower case letters) or revision number (as long as the revision number is greater than any existing revision number). It is often useful to use this option in conjunction with the **force commit** option described above.

If the **force recurse** box is checked, folders will be committed recursively. This is normally the default for folder commit operations but is turned off by default if **force revision/branch** is selected.

If the **do not run any module program** box is checked, programs specified to run on commit for the relevant module will not be run. Refer to Sections C.1.5 and C.3 of the Cederqvist manual for more information.

After setting the desired options on the Commit settings and Commit options tabs, click the **OK** button to commit the selected files or folder. A simple commit of one file with no errors would result in messages like this in the Output Pane:

```
cvs commit -m "This is a CVS test module." buffer.c buffer.h (indirectory
D:\Work\cvs-1.11.1p1\src\
Checking in buffer.c;
/Store/Cvs-Admin/cvs-1.11.1p1/src/buffer.c,v <-- buffer.c
new revision: 1.3; previous revision: 1.2
done
```

```
*****CVS exited normally with code 0*****
```

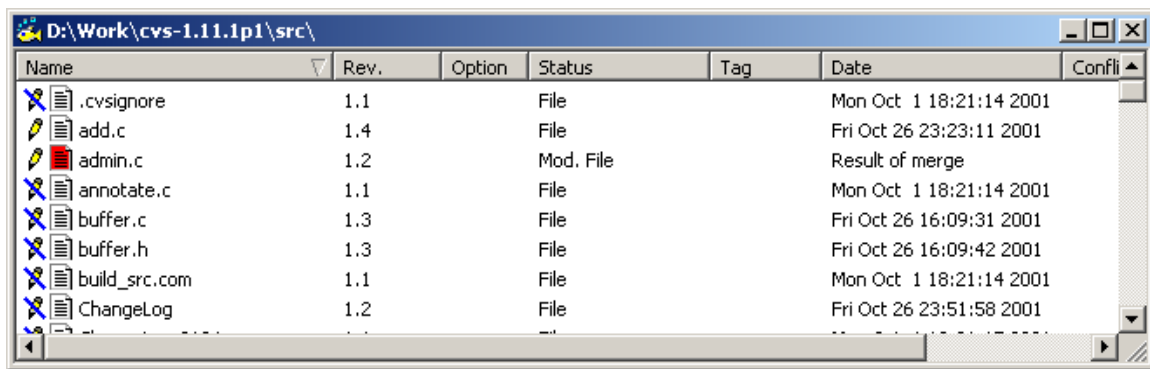
Errors on a **commit** operation result in messages similar to the following:

```
cvs commit -m "This is a CVS test module." (in directory D:\Work\cvs-1.11.1p1\src\)  
cvs.exe commit: Examining .  
cvs server: Up-to-date check failed for `admin.c'  
cvs [server aborted]: correct above errors first!
```

```
*****CVS exited normally with code 1*****
```

In the above example, the file `admin.c` was found to be out of date. This means that the revision that was checked out and modified in the working directory was older than the head revision of the file in the repository. Most likely, this error happens because a second developer modifies and commits a file while the first developer is editing the same file. To fix this error, the update command must be used as explained in Section 4.9.

Updating the file `admin.c` will cause CVS to attempt to merge the changes between the 2 revisions with the local modifications. If this process is successful the result of the merge will be placed into the local working directory. The WinCvs file browser will display Result of Merge in the Date field as shown here:



Also, the WinCvs Output Pane will have informational messages such as the following:

```
cvs update admin.c (in directory D:\Work\cvs-1.11.1p1\src\)  
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/src/admin.c,v  
retrieving revision 1.1  
retrieving revision 1.2  
Merging differences between 1.1 and 1.2 into admin.c  
M admin.c
```

```
*****CVS exited normally with code 0*****
```

At this point, the commit command can be invoked again to check in the merged file. The Output Pane will display messages similar to the following:

```
cvs commit -m "This is a CVS test module." (in directory D:\Work\cvs-1.11.1p1\src\)  
cvs.exe commit: Examining .  
Checking in admin.c;  
/Store/Cvs-Admin/cvs-1.11.1p1/src/admin.c,v <-- admin.c  
new revision: 1.3; previous revision: 1.2  
done
```

```
*****CVS exited normally with code 0*****
```

4.13 Adding Files or Folders to the Repository

Adding files or folders to a CVS repository can be done using either the CVS **add** or **import** command. Developers should normally use the **add** command to add files or a small number of folders. There are only two situations where the **import** command should be used:

- If the files to be added require the creation of a new module (because, for example, no appropriate module exists), the **import** command **must** be used. To maintain a consistent naming convention for modules, it is a good idea to leave module creation to a CVS administrator. The number of modules in a repository should also be controlled to avoid confusion.
- If a multi-level hierarchy is being added to an existing module, it can be tedious to add each level manually. The **add** command allows addition of all files and sub-folders of a single parent folder to be added in one operation. The same operation would, however, have to be repeated manually within each of the sub-folders down to the leaves of the tree. In addition, binary files need to be added in a separate operation from text files making the task even more tedious. In this situation, it would be much simpler to use the **import** command to import the entire hierarchy to a sub-folder of an existing module. It still might be a good idea to get help from a CVS administrator when attempting file imports in this way.

If the **add** command sounds more appropriate (the normal case), read Section 4.13.1 for documentation on using the **add** command. If the **import** command sounds better, read Section 4.13.2.

4.13.1 Adding Files or Folders Using Add

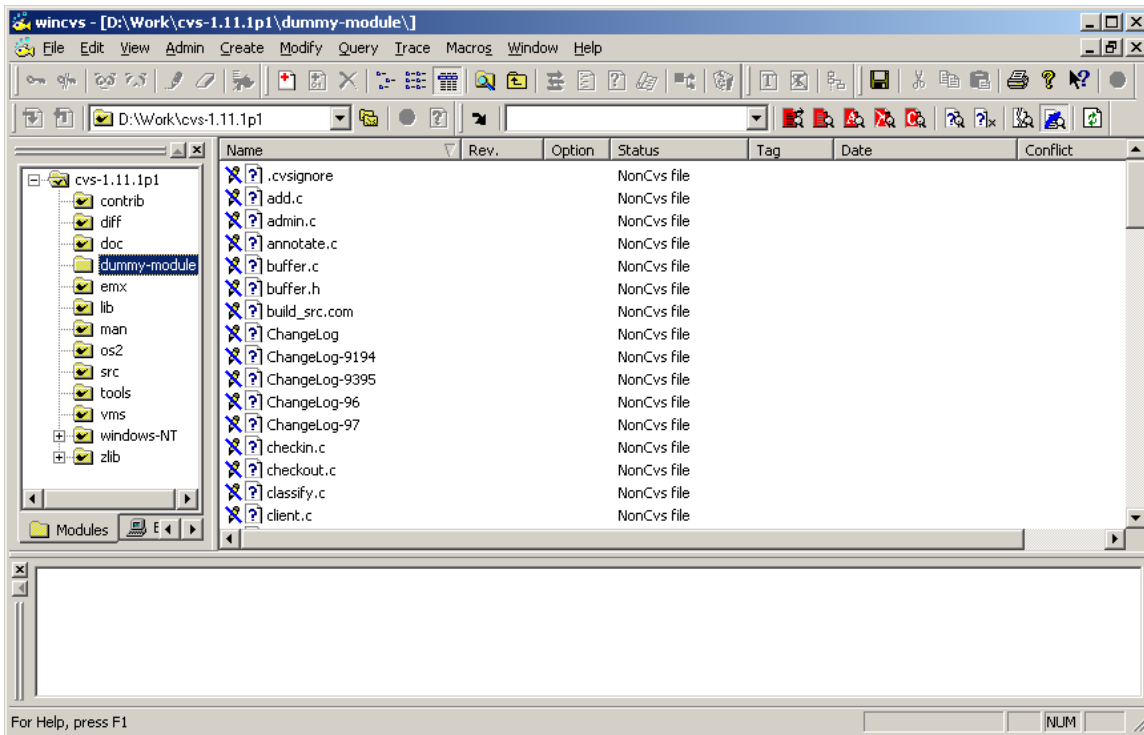
Before files or folders can be added to a module using the **add** command, a working directory must exist with the module already checked out. This obviously means that the module must already exist in the repository. If the module does not exist, refer to Section 4.6 for information about adding modules to the repository. Files and folders can be created within an existing working directory using any appropriate method such as:

- create an empty folder from the Windows Explorer
- copy files or folders from another location
- copy an existing hierarchy from another location (consider using **import**, Section 4.13.2)
- create files using a text editor, Microsoft Word, or other application

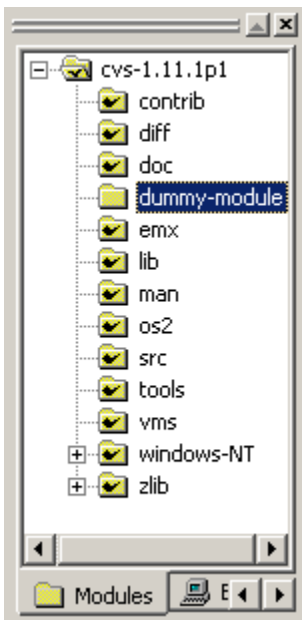
There are three important facts that need to be mentioned regarding the **add** command:

1. The **add** command is NEVER recursive in CVS. This means that adding an existing hierarchy to an existing module is a manual process. Files at each level of the tree must manually. When adding more than one level to the existing repository, consider using the **import** command (Section 4.13.2).
2. CVS needs to be told which files to treat as binary and which files to treat as text files. WinCvs will usually warn you if an attempt is made to add a binary file as a text file but it is safer to add text files with the **Add** versions of WinCvs commands, and binary files with the **Add Binary** versions.
3. The **add** command just adds the file to the local working directory. The **commit** command (Section 4.12) must be used to permanently add the files to the repository.

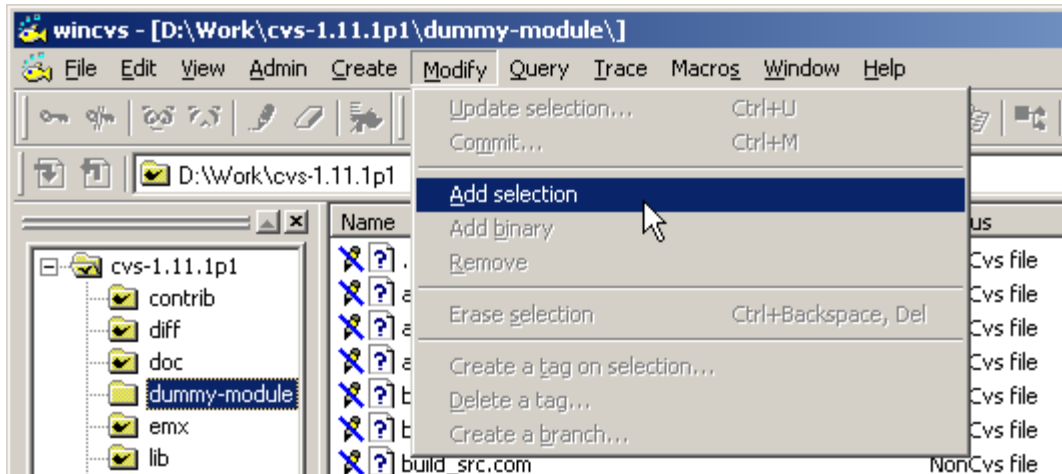
Once all the files and folders are in place in the working directory, they can be added to the repository. As with other commands, WinCvs provides several ways to invoke the **add** command. In the following example, a folder called dummy-module will be added to the repository. This folder contains both text and binary files. Viewing the new folder from WinCvs shows a ? icon for each file and lists the Status of the files as NonCvs files as shown here:



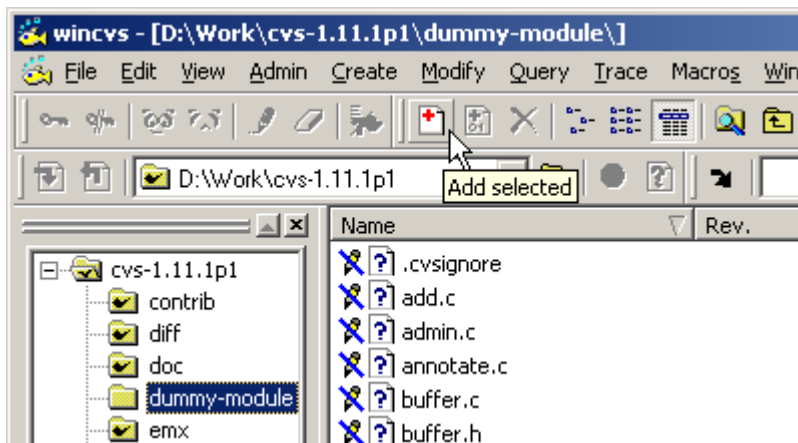
The first step is to add the parent folder for the folder to be imported. Notice that prior to adding the folder, the icon is missing the check mark as shown below. This means that the folder is unknown to CVS:



To add the folder, select it in the Workspace pane as shown above and run the **add** command. The **add** command can be invoked from the WinCvs **Modify** menu as shown here:



The **add** command is also available as an icon on the toolbar as shown here:



After invoking the add command for the dummy-module folder, messages similar to the following will be displayed in the Output Pane:

```
cvs add dummy-module (in directory D:\Work\cvs-1.11.1p1\)\
? dummy-module/.cvsignore
? dummy-module/add.c
.
.
? dummy-module/zlib.c
Directory /Store/Cvs-Admin/cvs-1.11.1p1/dummy-module added to the repository

*****CVS exited normally with code 0*****
```

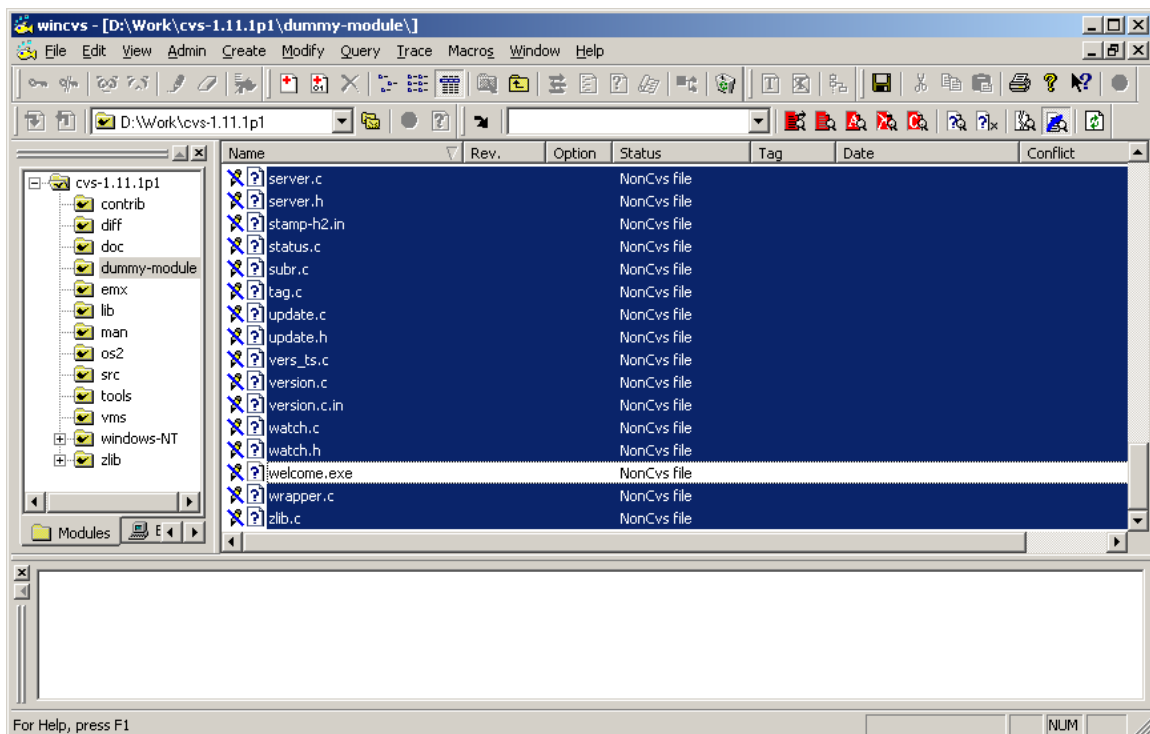
These messages indicate that the folder was successfully added but WinCvs is warning that the files inside the folder have not yet been added.

The next step is to add the files inside the folder. In this example, one of the files is a binary file: `welcome.exe` while the remaining files are all text files. There are two ways to add the files:

- Add all files except the binary file `welcome.exe` in one operation using the **add** command. Add `welcome.exe` in a separate operation using the **add binary** command.
- Add an entry to the `cvswrappers` file to instruct CVS to assume that all files that match `*.exe` are binary files. If this is done, all files can be added using a single **add** command. The `welcome.exe` file will be added as a binary file automatically. For more information on the `cvswrappers` file, refer to Section C.2 of the Cederqvist manual.

In this example, the text and binary files are added in two separate operations.

The first step will be to select all files in the `dummy-module` folder except for the binary file `welcome.exe`. The display should look similar to the following:



Invoke the **add** command from the menu or toolbar as described previously. During the add operation, several types of messages will be displayed.

First, some messages may be displayed warning about non-printable characters. These messages are only warnings and do not affect the addition of the files. If possible, these characters should be removed from the files prior to committing them to the repository but this is not required. An example is shown here:

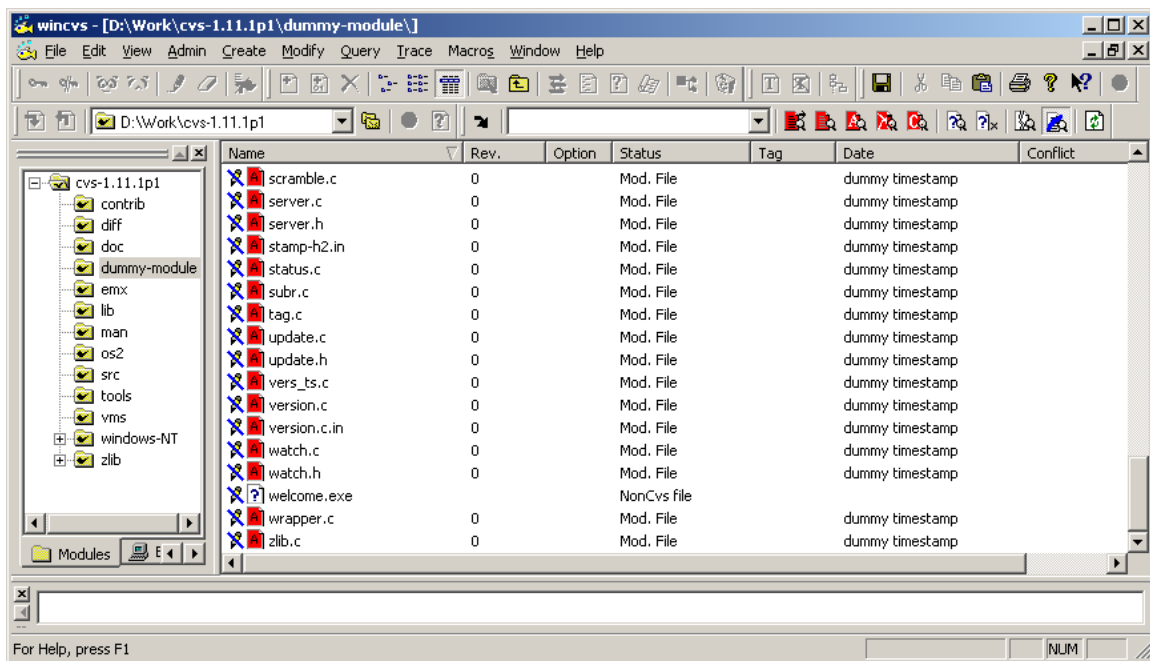
```
Warning : 'ChangeLog-9194' has some escape characters in it (0x00-0x20, 0x80-0xFF), you should correct it first
```

The remainder of the messages should be similar to the following indicating that the file has been added to the working directory and scheduled for addition to the repository. As noted at the end of the messages, the CVS **commit** command must be used to add the files to the repository permanently.

```
cvcs server: scheduling file `add.c' for addition
.
.
.
cvcs server: use 'cvcs commit' to add these files permanently

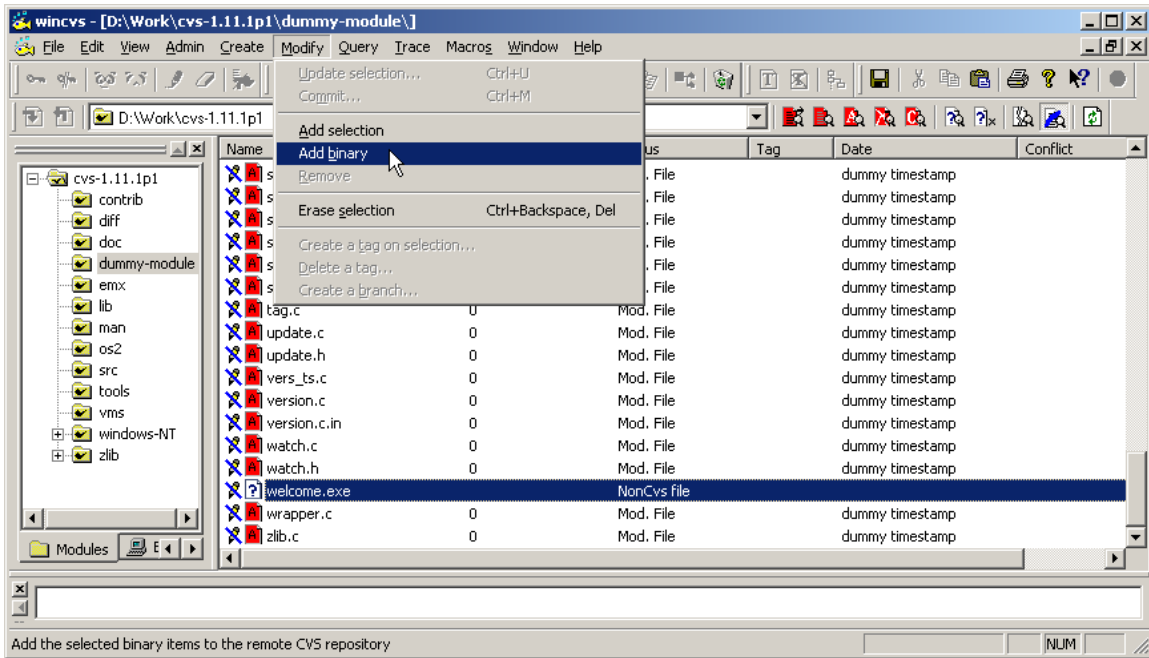
*****CVS exited normally with code 0*****
```

At this point, the display should look like this:

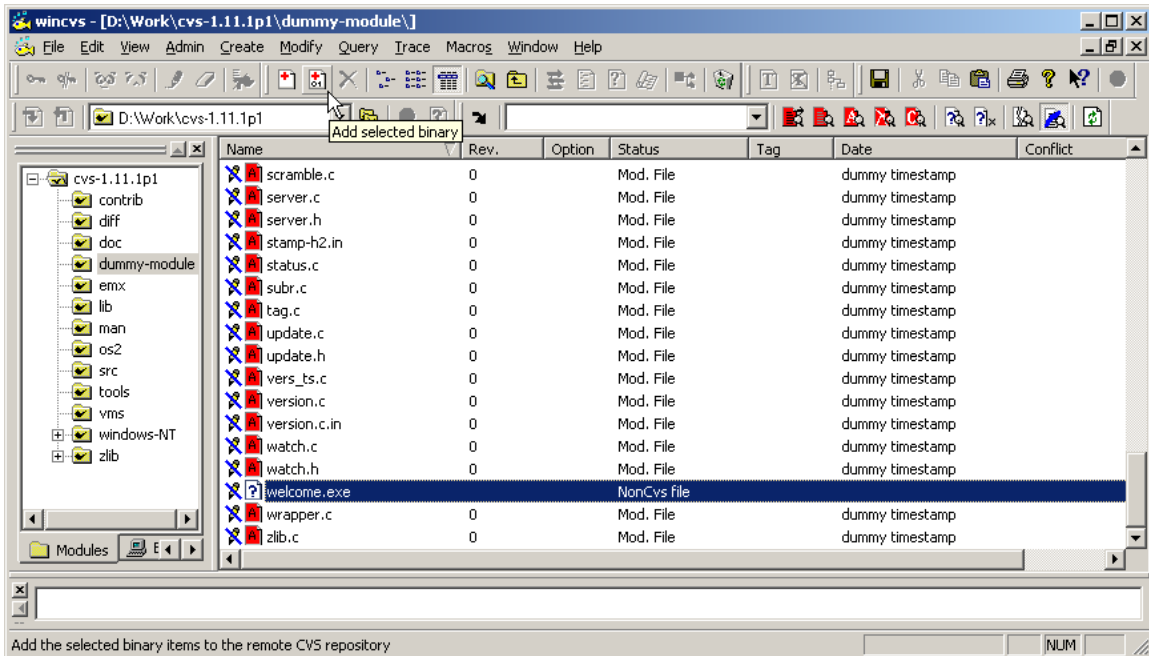


Note that the added files now have a red **A** icon indicating that the files are in an add pending state and the timestamp of each file is listed as dummy timestamp. Note that the binary file welcome.exe is still listed as a NonCvs file since it has not yet been added.

The next step will be to select the binary file `welcome.exe` and invoke the **add binary** command. The **add binary** command operates on the selected files and can be invoked from the WinCvs menu by selecting **Add binary** from the **Modify** menu as shown here:



The **add binary** command is also available as an icon on the toolbar as shown here:



During the add binary operation, messages similar to the following will be displayed in the Output Pane:

```

cvs add -kb welcome.exe (in directory D:\Work\cvs-1.11.1p1\dummy-module\
cvs server: scheduling file `welcome.exe' for addition
cvs server: use 'cvs commit' to add this file permanently

```

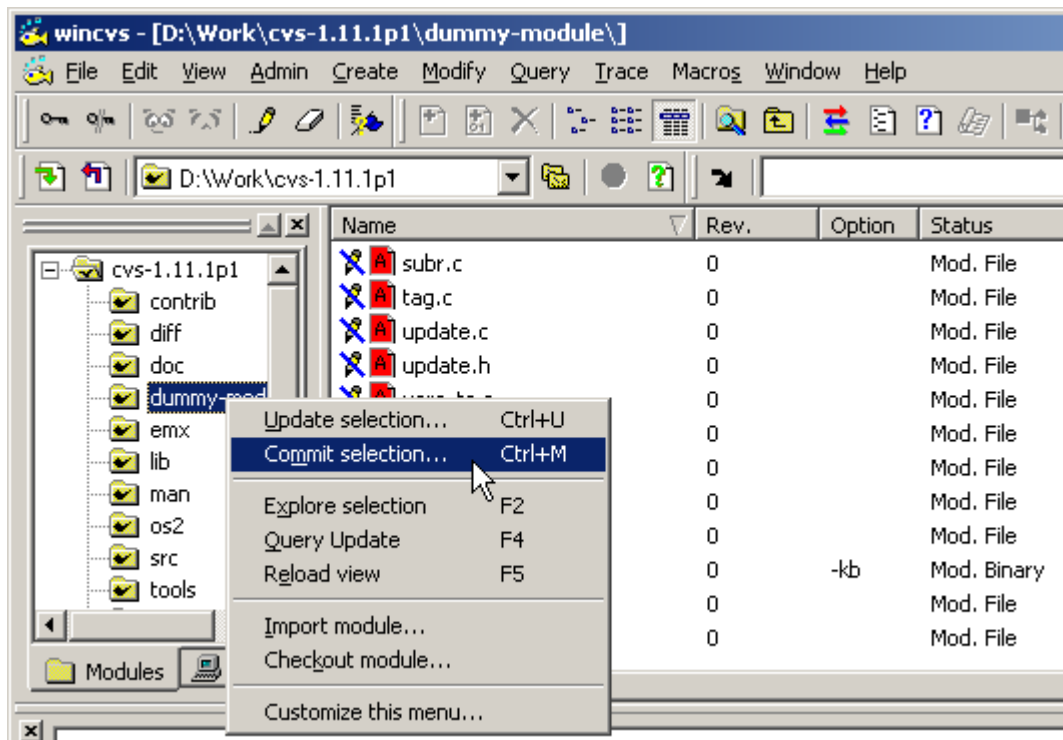
```
*****CVS exited normally with code 0*****
```

As indicated, the file or folder still requires a commit operation to actually add the file to the repository. The File pane of the WinCvs display should now look like this:

Name	Rev.	Option	Status	Tag	Date	Conflict
subr.c	0		Mod. File		dummy timestamp	
tag.c	0		Mod. File		dummy timestamp	
update.c	0		Mod. File		dummy timestamp	
update.h	0		Mod. File		dummy timestamp	
vers_ts.c	0		Mod. File		dummy timestamp	
version.c	0		Mod. File		dummy timestamp	
version.c.in	0		Mod. File		dummy timestamp	
watch.c	0		Mod. File		dummy timestamp	
watch.h	0		Mod. File		dummy timestamp	
welcome.exe	0	-kb	Mod. Binary		dummy timestamp	
wrapper.c	0		Mod. File		dummy timestamp	
zlib.c	0		Mod. File		dummy timestamp	

Notice that the file icon for welcome.exe is a red **A01** indicating the status of pending binary add. Note also the Option field which shows **-kb** indicating a binary file.

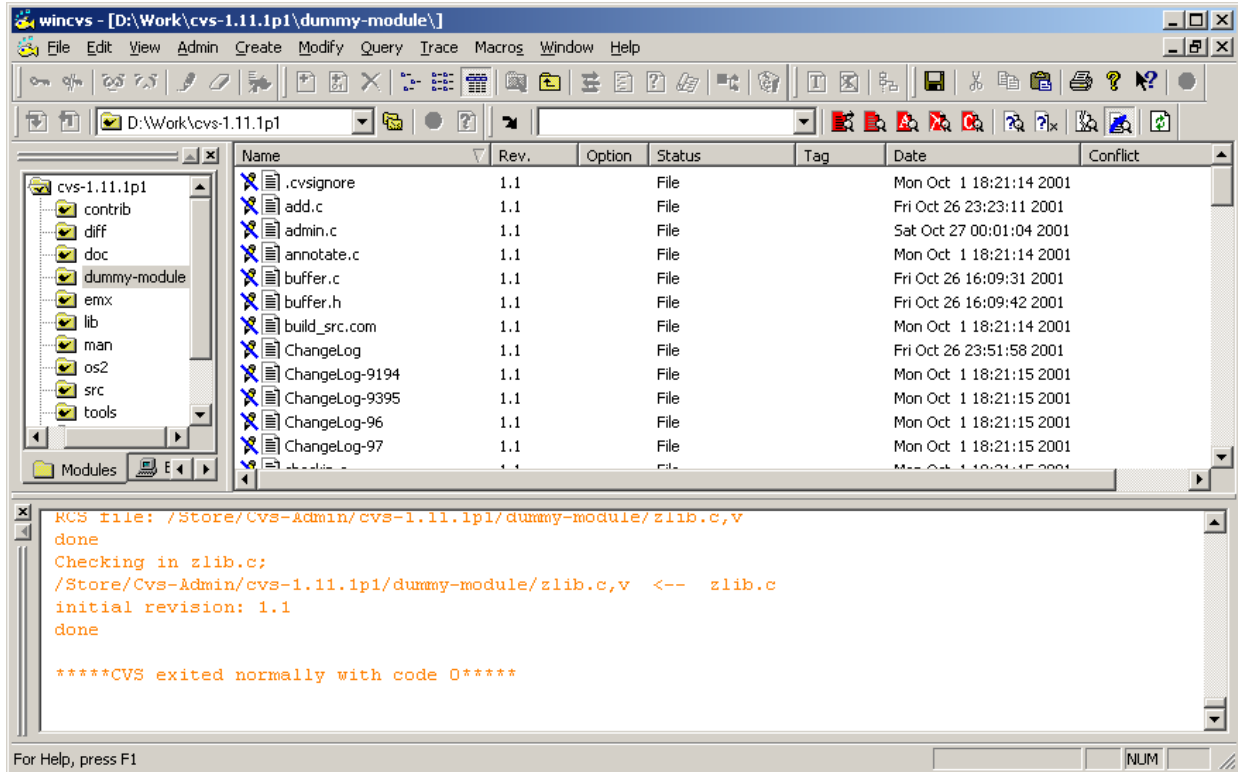
The final step in adding the folder is to commit the added files to the repository as shown in the following example using the context menu. Refer to Section 4.12 of this manual for more information on the **commit** command.



During the **commit** operation, messages such as the following will be displayed for each file added:

```
Checking in welcome.exe;  
/Store/Cvs-Admin/cvs-1.11.1p1/dummy-module/welcome.exe,v <-- welcome.exe  
initial revision: 1.1  
done
```

After the **commit** operation is complete, the WinCvs display should look like this:



Note that all files were checked in as revision 1.1 and have a valid timestamp.

4.13.2 Adding Files or Folders Using Import

This CVS **import** command provides a way to add an existing hierarchy of files and folders to the repository by either creating a new module or adding to an existing module. As discussed in Section 4.13, there are several reasons this command should be used primarily by a CVS administrator.

Unlike the **add** command, the hierarchy to be imported should NOT be in the local working directory prior to running the import command. The entire hierarchy can later be checked out to the local working directory after the **import** is complete.

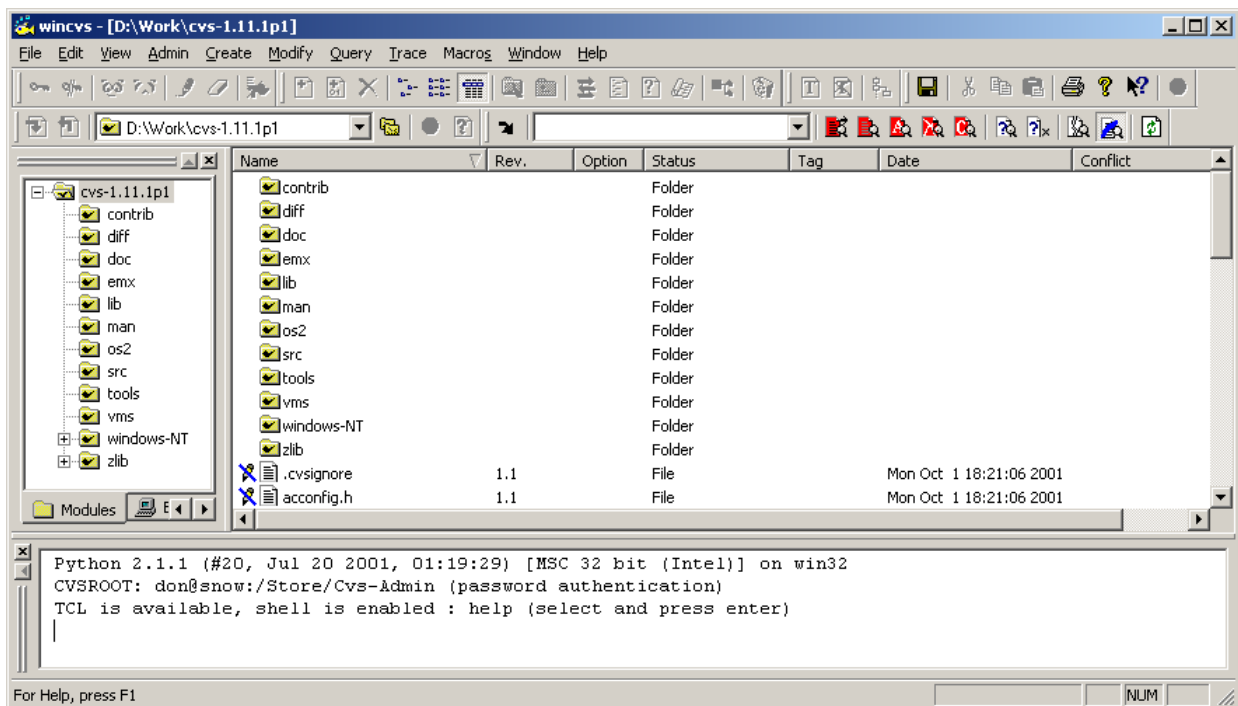
In the previous section (4.13.1), a folder of text and binary files was added to an existing repository. Even though it was only a one level hierarchy, it was a long manual process to first add the folder itself, add the text files, and add the binary files all in separate operations.

The **import** command can make this process much less tedious. The **import** command will create all levels of the imported hierarchy automatically and will also make good guesses about text and binary files.

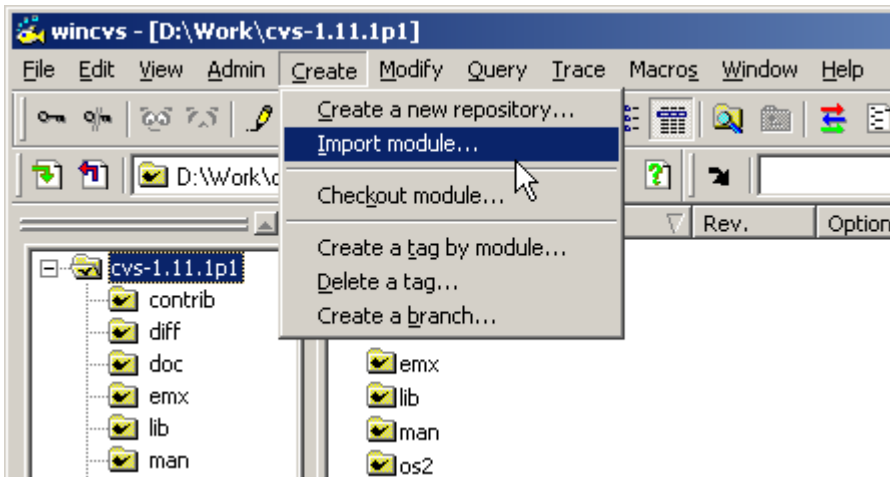
Most of the features of the steps involved in importing hierarchies with the **import** command have already been explained in Section 4.6.2 of this manual. Importing into an existing module is slightly different, however, so a second example is included here.

Just as in the previous section, a folder named **dummy-module** containing a number of text files and one binary file needs to be imported. As mentioned earlier, when using **import**, the tree to be imported should NOT be located inside the working directory. For this example, assume that the folder **C:\dummy-module** will be imported to a module **cvs-1.11.1p1** checked out as **D:\Work\cvs-1.11.1p1**. If the tree to be imported is already inside the working directory, it would be a good idea to move it somewhere outside the working directory prior to attempting the **import**.

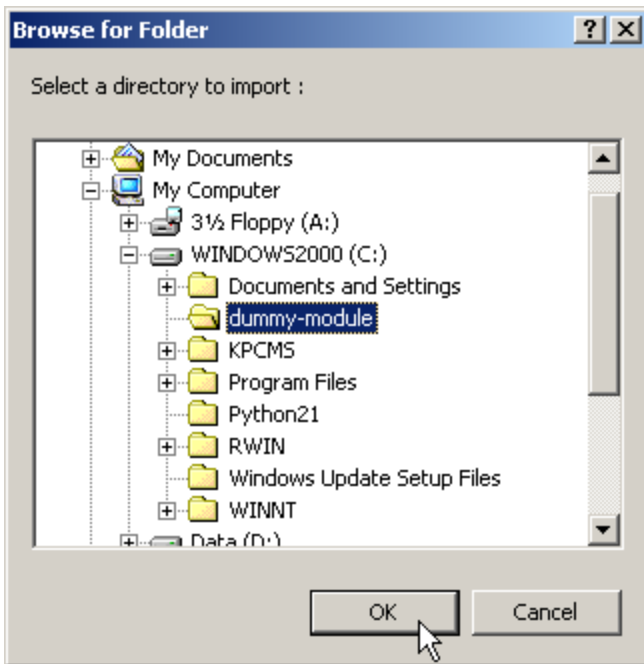
Prior to adding the new tree to the module, the working directory looks like this:



To begin the **import**, select the top level module folder `cvcs-1.11.1p1` and invoke the **import** command from the WinCvs **Create** menu as shown here:



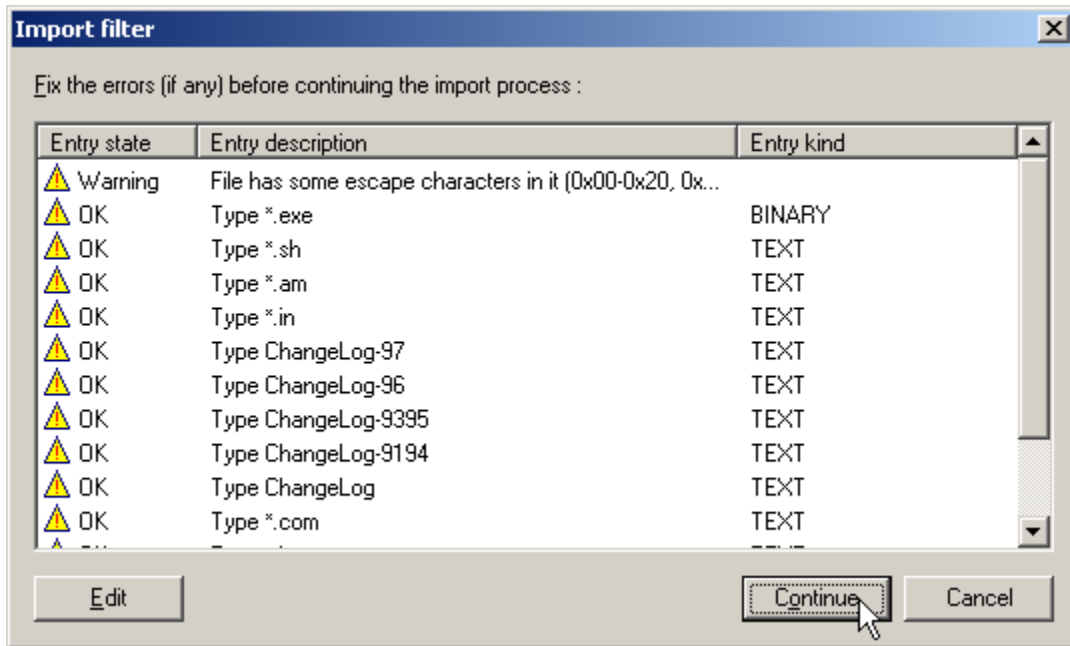
The import command will begin by displaying the Browse for folder as shown below. Browse to the folder to be imported and click the **OK** button as shown here:



Depending on the size of the tree to import, there may be a delay at this point while WinCvs determines the suggested file types for import (text or binary). During this time, messages similar to the following will be displayed in the WinCvs Output Pane:

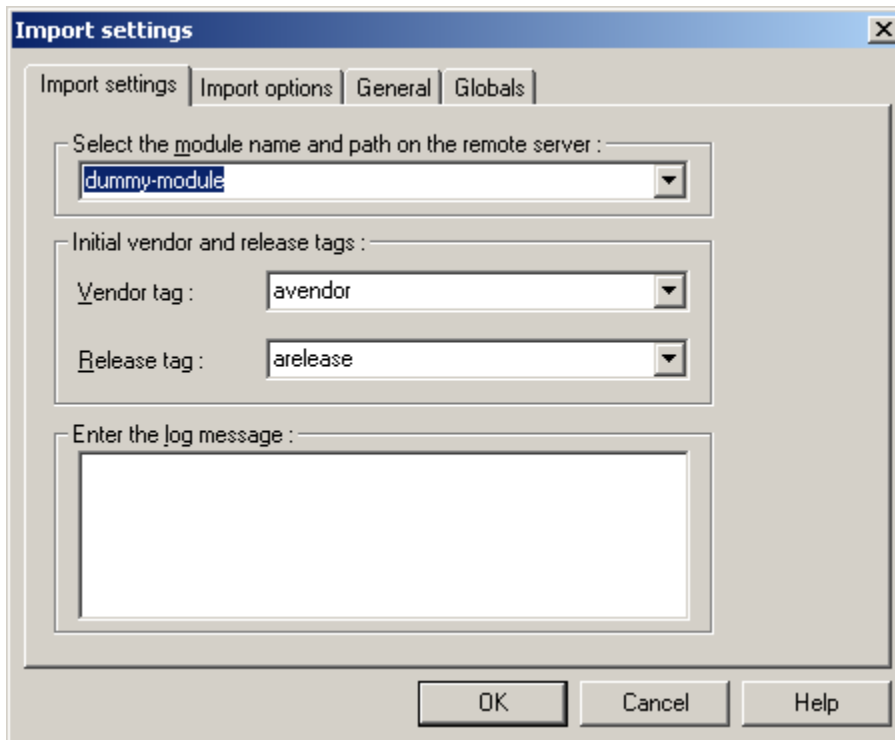
```
Filtering 'C:\dummy-module'...
```

When the filtering process is complete, the Import filter dialog will be displayed as shown here:



Notice that WinCVS has correctly determined that *.exe files should be imported as binary files. If any file types have been incorrectly assigned, they can be modified using the **Edit** button. When the file types are correct, click the **Continue** button to proceed with the import.

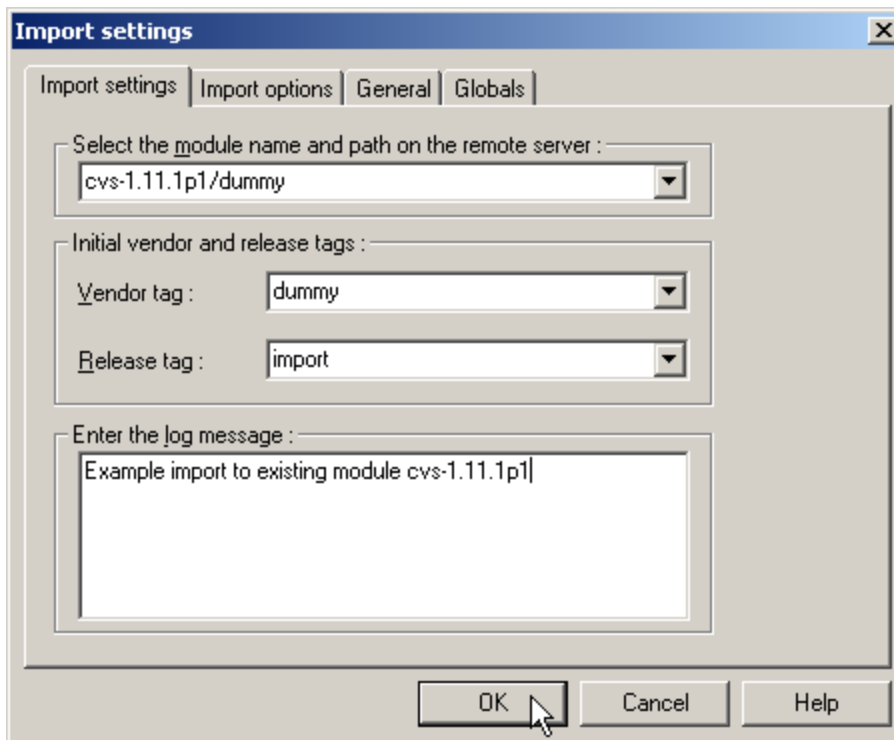
Next, the Import settings dialog will be displayed as shown here:



In this example, we want to create a new folder dummy under the existing cvs-1.11.1p1 module. Note that the imported folder C:\dummy-module will not appear in the repository. All the files and subfolders of C:\dummy-module will be imported to the repository under the dummy folder. This is example was done to show that a folder can be imported with a different name. The following fields need to be entered prior to initiating the import:

- Enter cvs-1.11.1p1/dummy as the module name and path. Note that CVS **REQUIRES** forward slashes `'/'` to separate directory names. Specifying a backslash `'\'` will create a new module called cvs-1.11.1p1\dummy in the repository root instead of a new directory dummy under the existing cvs-1.11.1p1 module.
- Enter something for the vendor and release tags. If meaningful names are not needed, specify the folder name dummy as the vendor tag and import as the release tag. These tags are often useless and can normally be deleted. CVS requires them, however, due to the history of the **import** command. They can be deleted after the **import** by selecting the top level folder and using **Delete a tag** which is available under the **Modify** menu.
- Enter an appropriate log message. This message will be part of the files log information for the initial import version and will appear in each file where the \$Log keyword is specified (text files only).

The Import settings panel should look something like this:



After all the fields are entered, hit the **OK** button to initiate the import.

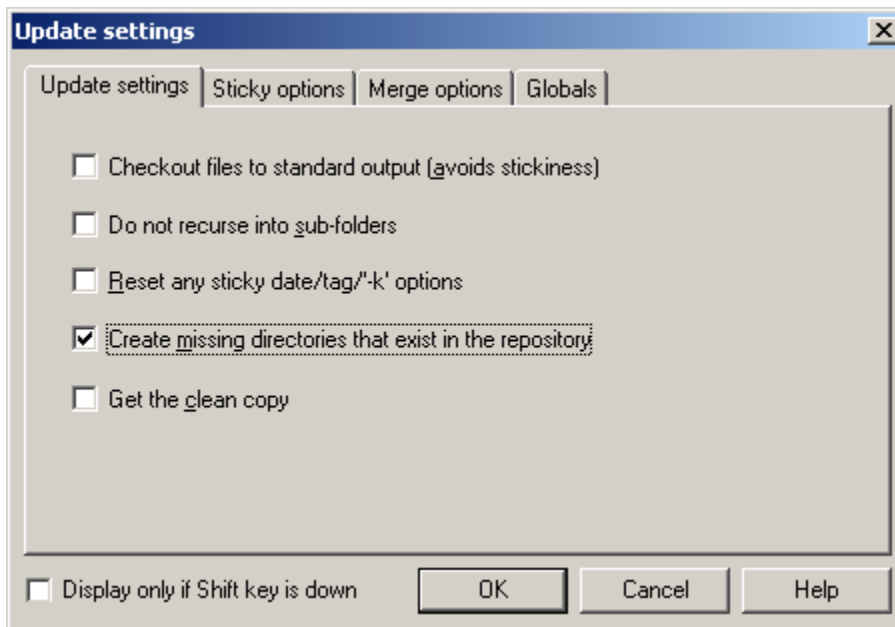
During the import the WinCvs Output Pane will display messages like this:

```
cvs import -I ! -I CVS -W "*.exe -k 'b'" -m "Example import to existing module
cvs-1.11.1p1" cvs-1.11.1p1/dummy dummy import (in directory C:\dummy-module)
N cvs-1.11.1p1/dummy/add.c
.
.
N cvs-1.11.1p1/dummy/zlib.c

No conflicts created by this import

*****CVS exited normally with code 0*****
```

Now that the files have been imported to the repository, they may be checked out to the working directory by using the **update** command on the parent folder of the imported tree. In this example, select the `cvs-1.11.1p1` directory in the WinCvs Workspace Pane and select **update** with the right mouse button. The Update settings panel is displayed:



Check the **Create missing directories that exist in the repository** option and hit the **OK** button to retrieve a working copy of the imported files to the local working directory.

In some cases it may be undesirable to use the **update** command as in the previous example since folders other than the desired `dummy` folder may be checked out as well. Another way to retrieve the working copy of the new folder is to use the **Checkout module** command available from the WinCvs **Create** menu. In the preceding example, specifying `cvs-1.11.1p1/dummy` as the module name would have the desired result of checking out only the new folder `dummy`.

4.14 Multiple Developer Coordination

The default model in CVS for managing changes by multiple developers is known as *unreserved checkouts*. In this model, multiple developers can each edit a *working* copy of a file simultaneously. The first developer to **commit** the file will succeed with no additional work required. The second developer, however, will need to merge in the changes from the first developer prior to committing the file. The merge feature is handled automatically by CVS and only requires manual merging when CVS finds conflicting modifications that it cannot resolve.

CVS also provides limited support for a *reserved checkout* model. The support is limited in that CVS does not require a lock before allowing a file to be edited although it does not allow **commit** to a file that is locked by another developer.

4.14.1 Understanding Merging and the Unreserved Checkout Model

When developers use the *unreserved checkout* model in CVS, any number of developers may be working on the same file simultaneously. As explained in Section 4.10.1, the **edit** command should be used to add write access to a file before making any modifications. At any time, the list of developers currently editing a file can be displayed by selecting the file and using **Editors** command from the **Trace** menu.

When a developer tries to **commit** a modified file, one of two things will happen:

- If no newer revision of the file is found in the repository, a new revision will be created and the file will be committed to the repository directly as described in Section 4.12. This is the normal case where either no other developer is working on the same file or at least no other developer has committed any changes.
- If a newer version of the file is found in the repository, CVS will issue a warning and abort the commit operation.

4.14.2 Example Commit with No-Conflicts Merge

In the case where CVS aborts the **commit** operation due to a newer version found in the repository, an Up-to-date warning like the following will be displayed in the WinCvs Output Pane. The commit is not performed:

```
cvs commit -m test admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\  
cvs server: Up-to-date check failed for `admin.c'  
cvs [server aborted]: correct above errors first!
```

```
*****CVS exited normally with code 1*****
```

In this case, the **update** command must first be used to merge in the missing changes. If the merge succeeds, messages similar to the following will be displayed in the Output Pane:

```
cvs update admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\  
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/dummy/admin.c,v  
retrieving revision 1.2  
retrieving revision 1.3  
Merging differences between 1.2 and 1.3 into admin.c  
M admin.c
```

```
*****CVS exited normally with code 0*****
```

In this case, the status of the file will remain Mod. File but the modification date will change to the text: Result of merge as shown here:

Name	Rev.	Option	Status	Tag	Date	Conflict
.#admin.c.1.1			NonCvs file			
.#admin.c.1.2			NonCvs file			
.cvsignore	1.1		File		Sat Oct 27 03:03:55 2001	
add.c	1.1		File		Sat Oct 27 03:03:55 2001	
admin.c	1.3		Mod. File		Result of merge	
annotate.c	1.1		File		Sat Oct 27 03:03:55 2001	
buffer.c	1.1		File		Sat Oct 27 03:03:55 2001	

At this point the **commit** command can be run again causing the merged file to be copied to the repository.

4.14.3 Example Commit with Conflict Resolution

In the case where the update command fails because CVS is unable to automatically merge local changes with the repository version, messages similar to the following will be displayed:

```

cvs update admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\)
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/dummy/admin.c,v
retrieving revision 1.1
retrieving revision 1.2
Merging differences between 1.1 and 1.2 into admin.c
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in admin.c
C admin.c

```

*****CVS exited normally with code 0*****

Notice the ‘C’ indicating conflicts were found. In this case the file icon will change to a red **C**, the status will change to Conflict, and the Date field will change to Result of merge as shown here:

Name	Rev.	Option	Status	Tag	Date	Conflict
.#admin.c.1.1			NonCvs file			
.cvsignore	1.1		File		Sat Oct 27 03:03:55 2001	
add.c	1.1		File		Sat Oct 27 03:03:55 2001	
admin.c	1.2		Conflict		Result of merge	Sun Oct 28 18:48:2
annotate.c	1.1		File		Sat Oct 27 03:03:55 2001	
buffer.c	1.1		File		Sat Oct 27 03:03:55 2001	
buffer.h	1.1		File		Sat Oct 27 03:03:55 2001	

In this case, the resulting merged file must be re-edited to resolve the conflicts manually. Conflicts can be found in the file by searching for the <<<<<<, =====, and >>>>>> character strings as shown in this example:

```
/*
 * Dummy function to test WinCvs
 */
<<<<<< admin.c
void dummy_2( int param1 )
=====
void dummy_3( int param1, int param2 )
>>>>>> 1.5
{
<<<<<< admin.c
    printf("WinCvs Test 2...\n");
=====
    printf("WinCvs Test 3...\n");
>>>>>> 1.5
}

<<<<<< admin.c
void merge_test_2()
=====
void merge_test_3()
>>>>>> 1.5
{
}
}
```

The developer must manually remove the lines containing <<<<<<, =====, and >>>>>> and resolve the conflicts on the indicated lines. After the conflicts have been resolved, the **commit** command may be re-invoked.

If the developer attempts to commit the file without resolving the conflicts, the following error message will be printed by CVS and the commit will be aborted:

```
cvs commit -m test admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\)
cvs server: file `admin.c' had a conflict and has not been modified
cvs [server aborted]: correct above errors first!
```

```
*****CVS exited normally with code 1*****
```

If the developer resolves some of the conflicts but leaves some unresolved conflicts in the file, CVS will print the following warning message when the file is committed:

```
cvs commit -m test admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\)
cvs server: warning: file `admin.c' seems to still contain conflict indicators
Checking in admin.c;
/Store/Cvs-Admin/cvs-1.11.1p1/dummy/admin.c,v <-- admin.c
new revision: 1.6; previous revision: 1.5
done
```

```
*****CVS exited normally with code 0*****
```

Note that in this case, the commit operation was successful. The user should manually verify that all the conflict markers have been removed from the file. If not, the file should be modified to resolve any errors.

4.14.4 Understanding Locking and the Reserved Checkout Model

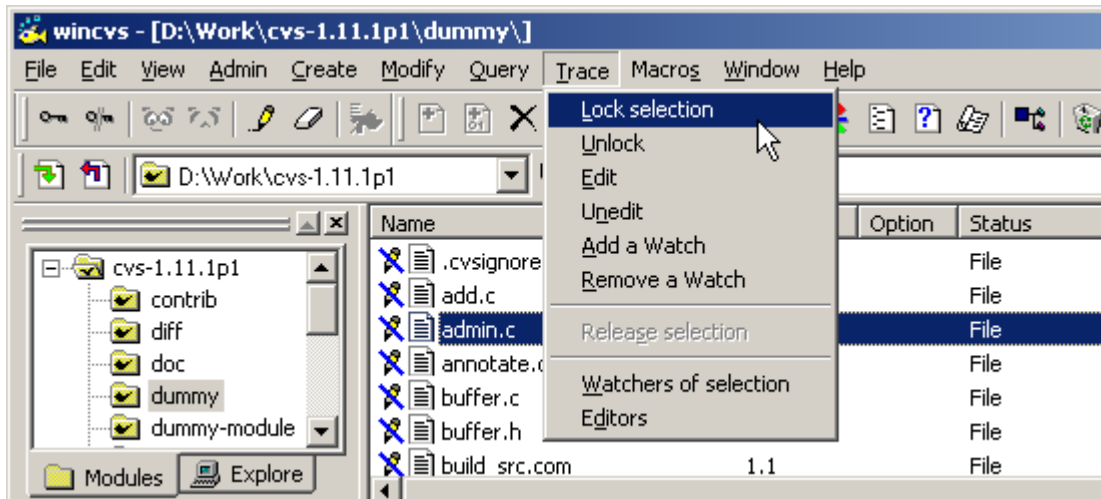
As stated in Section 4.14, CVS does provide a mechanism to support the *reserved checkout* model. The **lock** command is the basis of this model. Locking a file in CVS prevents another developer from either **locking** or **committing** a new version of the file. This is safe in that it does **prevent** the other developer from checking in a new version of the file but it does **allow** another developer to edit the file. This could lead to frustration, for example, if a developer forgets to get a file lock but just edits the file. When the developer attempts to commit the file, the operation will fail as shown in the following example:

```
cvs commit -m test admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\  
cvs [server aborted]: Revision 1.6 is already locked by don
```

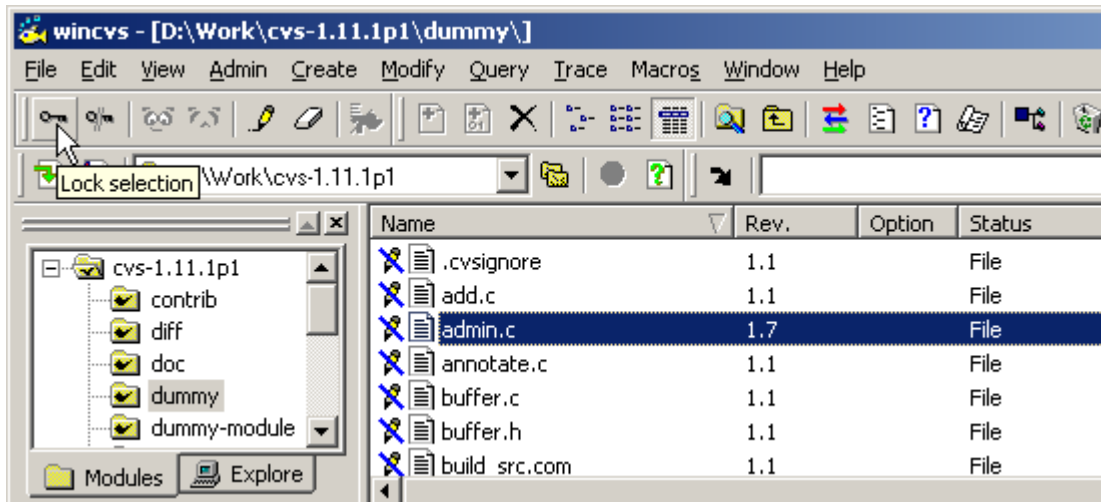
```
*****CVS exited normally with code 1*****
```

If all developers cooperate and ensure that files are locked prior to editing, this situation need not arise. Alternatively, the *unreserved checkout* model could be used.

The **lock** command operates on a selected file or folder (recursively). The **lock** command can be invoked from the WinCvs menu by selecting **Lock selection** from the **Trace** menu as shown here:



The **lock** command is also available as an icon on the toolbar as shown here:



The **lock** command can also be added to the context menu (right-mouse click) by selecting the **Customize this menu...** entry at the bottom of the context menu.

If a **lock** request succeeds, the messages similar to the following will be displayed on the Output Pane:

```
cvs admin -l admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\)  
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/dummy/admin.c,v  
1.7 locked  
done  
  
*****CVS exited normally with code 0*****
```

If a **lock** request fails, information including the owner of the lock will be displayed in the Output Pane such as in this example:

```
cvs admin -l admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\)  
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/dummy/admin.c,v  
cvs [server aborted]: Revision 1.7 is already locked by don  
  
*****CVS exited normally with code 1*****
```

4.14.5 Using Log to Determine Lock Status

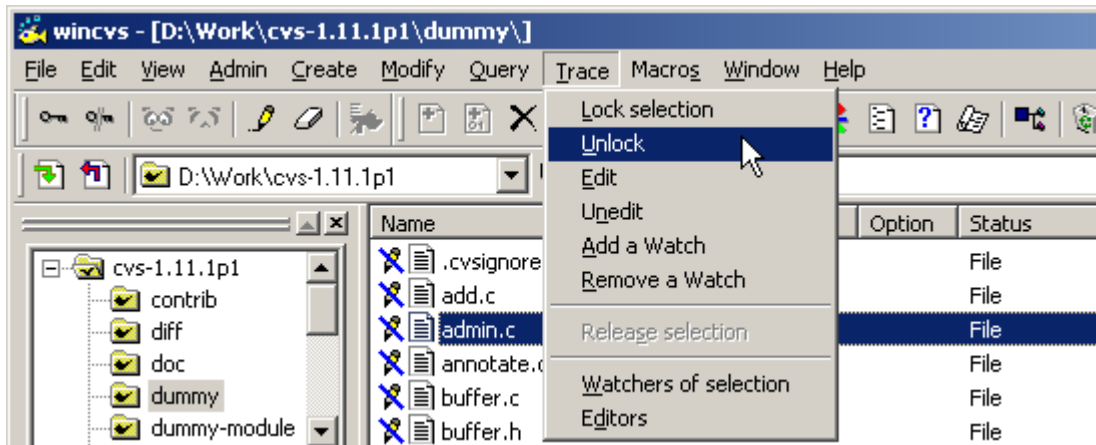
The **log** command is available from all menus and the tool bar to determine the locking status of a file. The information is displayed in the Output Pane as shown in the following example:

```
cvcs log admin.c (in directory D:\Work\cvcs-1.11.1p1\dummy\  
  
*****CVS exited normally with code 0*****  
  
Rcs file : '/Store/Cvs-Admin/cvcs-1.11.1p1/dummy/admin.c,v'  
Working file : 'admin.c'  
Head revision : 1.7  
Branch revision :  
Locks : strict  
        1.7 : 'don' ← indicates file is locked by don  
Access :  
Symbolic names :  
        1.1 : 'import'  
        1 : 'dummy'  
Keyword substitution : 'kv'  
Total revisions : 7  
Selected revisions : 7  
Description :
```

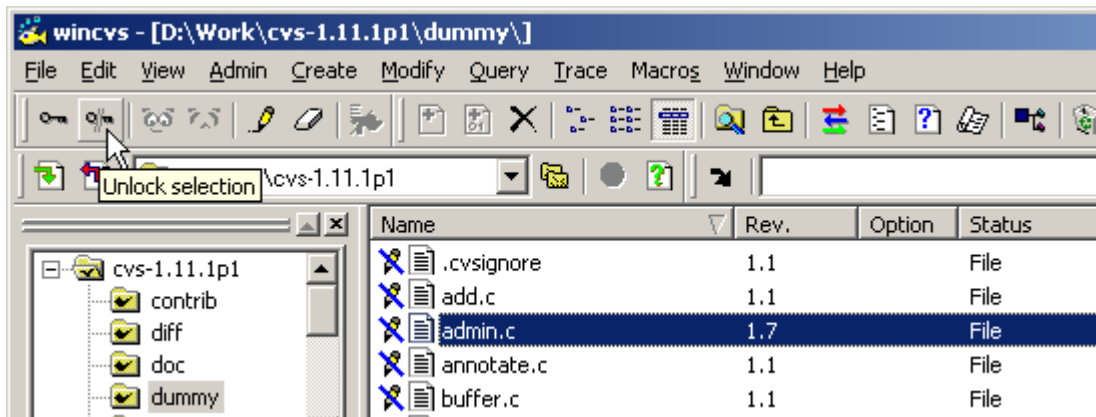
4.14.6 Unlocking Files

Files are automatically unlocked when a **commit** operation succeeds on the file. The **unlock** command is provided to manually unlock a file when the lock is no longer needed.

The **unlock** command operates on a selected file or folder (recursively). The **unlock** command can be invoked from the WinCvs menu by selecting **Unlock** from the **Trace** menu as shown here:



The **unlock** command is also available as an icon on the toolbar as shown here:



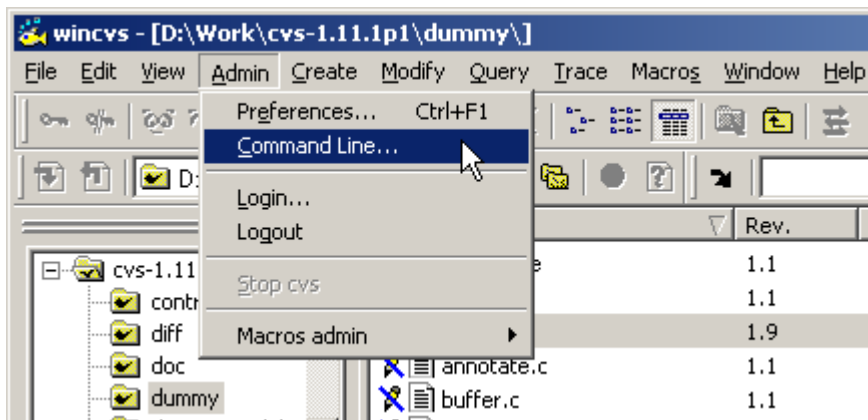
The **unlock** command can also be added to the context menu (right-mouse click) by selecting the **Customize this menu...** entry at the bottom of the context menu.

There are some cases where a developer may end up with locks on multiple revisions of a file. Attempting to unlock the file will result in an error message as shown here:

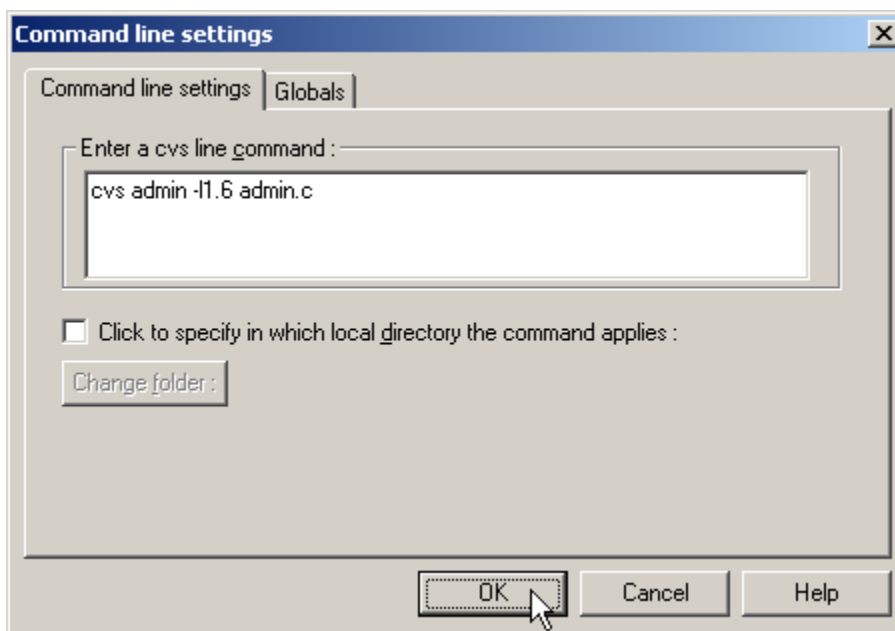
```
cvs admin -u admin.c (in directory D:\Work\cvs-1.11.1p1\dummy\  
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/dummy/admin.c,v  
cvs server: /Store/Cvs-Admin/cvs-1.11.1p1/dummy/admin.c,v: multiple revisions  
locked by cvs; please specify one  
cvs server: RCS file for `admin.c' not modified.
```

```
*****CVS exited normally with code 1*****
```

In this case, a specific revision of the file must be unlocked using the CVS **admin** command from the Command line settings dialog (advanced users can type it directly in the Output Pane). Open the Command line settings dialog by selecting **Command line** from the WinCvs **Admin** menu as shown here:



The Command line settings dialog will be displayed. Enter the `cvs admin -l<revision>` command as shown in the following example where revision 1.6 of admin.c is being unlocked:



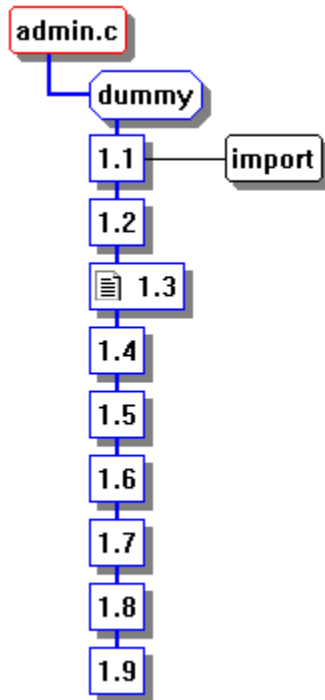
After clicking the **OK** button, the **unlock** command will execute in the Output Pane as shown here:

```
cvs admin -l.6 admin.c
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/dummy/admin.c,v
done

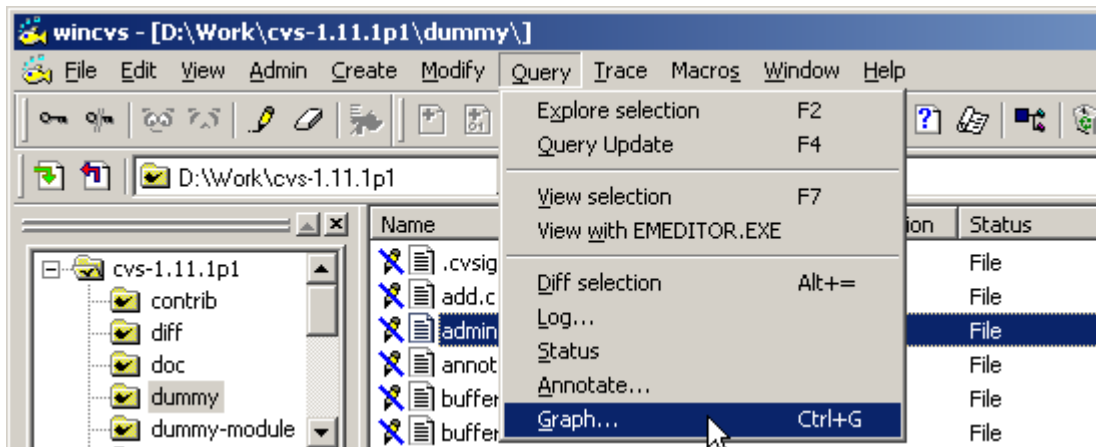
*****CVS exited normally with code 0*****
```


4.15 Revision History – Graph Command

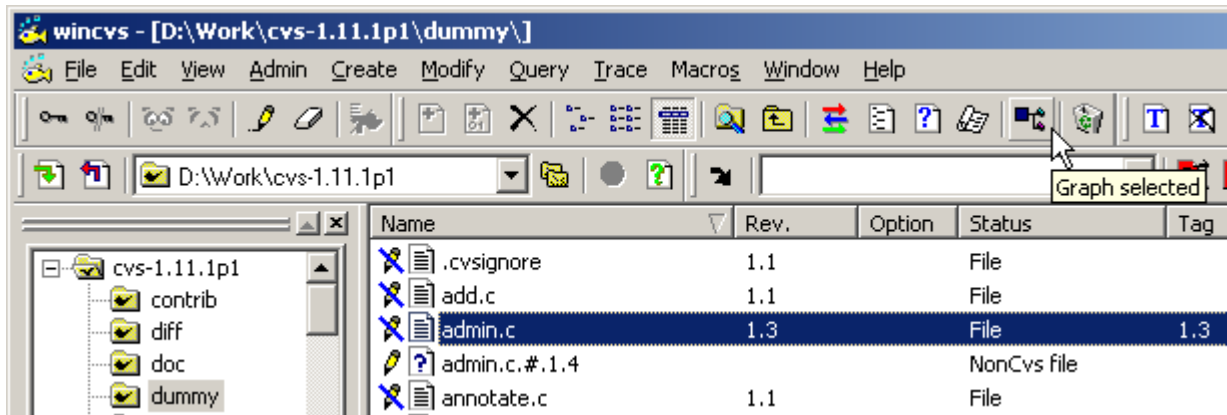
The **graph** command is a useful way to view the history of a file in the repository. The **graph** command is actually implemented using the CVS **log** command which returns the revision history for the selected file. WinCvs takes this output and generates a graphical representation of the revision history as shown here:



The **graph** command operates on a selected file and can be invoked from the WinCvs menu by selecting **Graph...** from the **Query** menu as shown here:

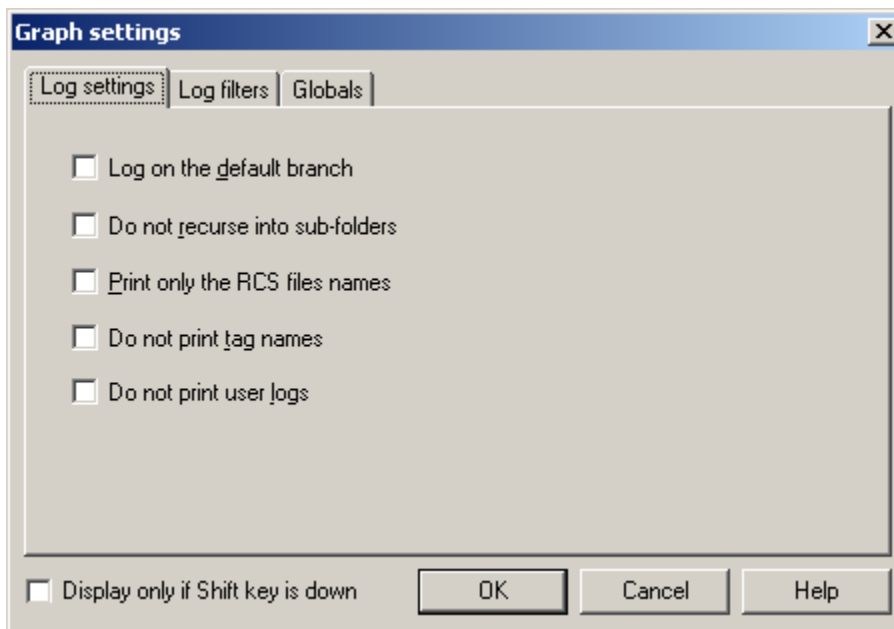


The **graph** command is also available as an icon on the toolbar as shown here:



The **graph** command can also be available by default on the context menu (right-mouse click).

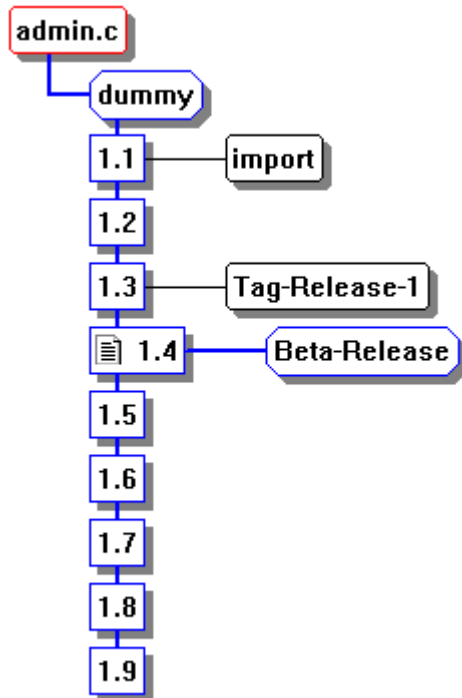
When the **graph** command is run using any of the above methods, the **Graph settings** dialog will be displayed unless configured to **skip** (see Section 4.2.4). If configured to **skip**, hold down the force key (SHIFT or CTRL) to force the dialog to be displayed as shown here:



There are many options in the **Graph setting** panels which the user may explore as necessary. In the following example, the **Do not print tag names** option has been selected. This may be a commonly used option to prevent too much information on the graph. Tag names in CVS include branch names (which are useful to see) but also include “tagged versions” which are probably not interesting to developers (these are usually of interest to release managers).

Once all the desired options are selected, click the **OK** button to display the graph. Note that a bug in early beta versions of WinCVS 1.3 caused the graph window to be hidden behind the file display. To view the hidden graph window, use the **Cascade** command under the **Window** menu.

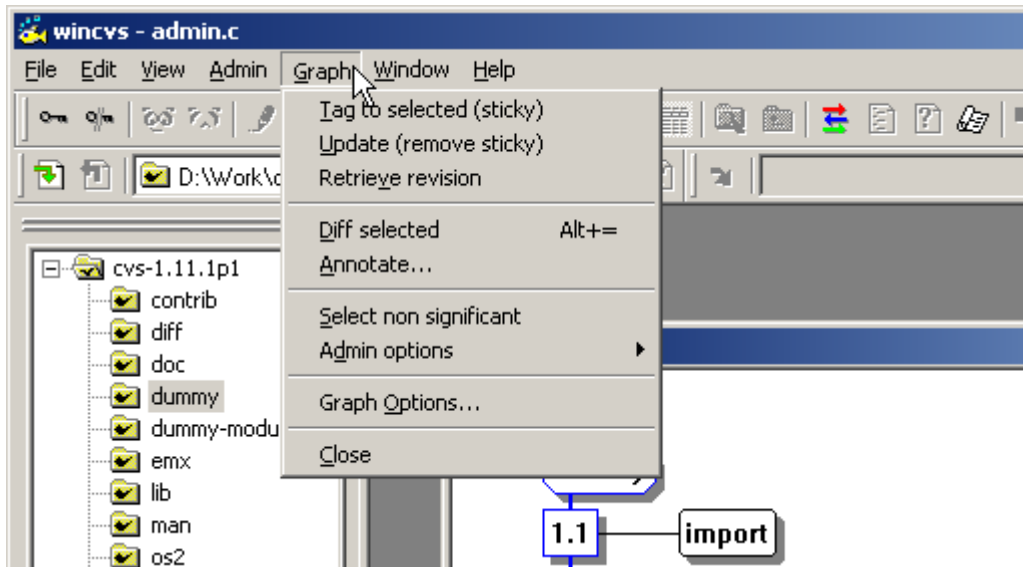
Within the graph window, file revisions are displayed in blue boxes, tags are displayed in black boxes with rounded corners, and branch tags are displayed in blue octagonal boxes. Note that the default colors can be changed from the Graph menu. The revision of the file that is currently in the working directory is marked with the 'file' icon as shown in the following example where revision 1.4 is checked out in the working directory:



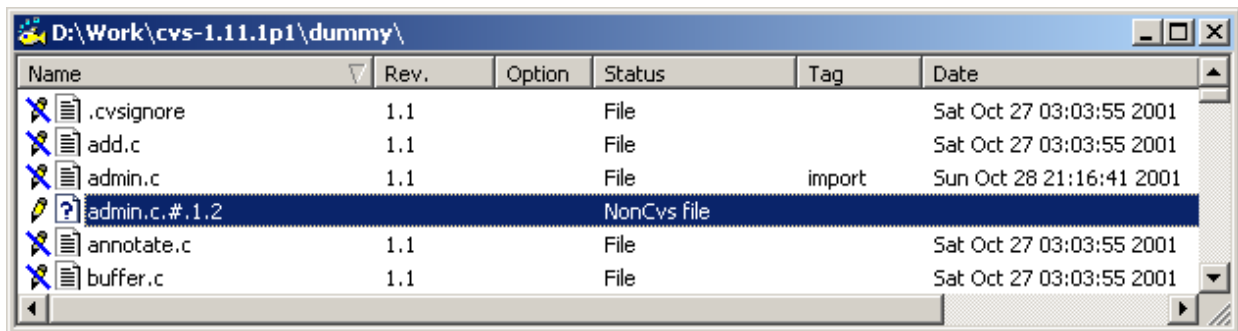
Several useful functions are available from the graph window. Clicking on a revision displays the revision information such as the author, modification timestamp, and log message for the selected revision in the WinCvs Output Pane as shown here:

```
Revision : 1.4
Date : 2001/10/28 18:54:2
Author : 'cvs'
State : 'Exp'
Lines : +4 -2
Description :
test
```

The WinCvs **Graph** menu is available when the graph window is active. This menu has several useful functions as shown here:



The **Retrieve revision** command is a useful way to view a previous revision of a file without replacing the revision currently in the working directory. Retrieving revision 1.2 of admin.c, for example, will create a file called admin.c.#.1.2 which will appear in the file window as shown here:



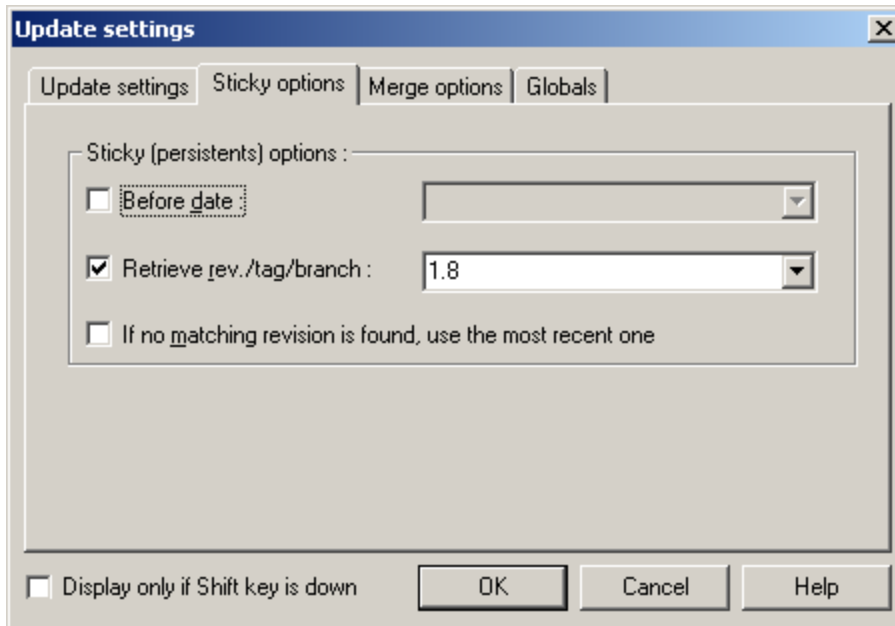
The file can then be viewed using any of the view commands described in Section 4.10.2. Note that the file icon is a ? and the status is listed as **NonCvs** file indicating that this file is not associated with any file in the repository. When finished with this file, it can be removed using the **Erase selection** command in the **Modify** menu or using the 'recycler' icon on the toolbar.

Diff selection is another very useful command in the **Graph** menu. If a single revision of a file is selected using the left-mouse button, **Diff selection** will produce a graphical display of the differences between the file in the current working directory and the selected revision from the repository. Using left-mouse click while holding down the CTRL key to select **two** revisions results in a graphical diff of the selected revisions in the repository.

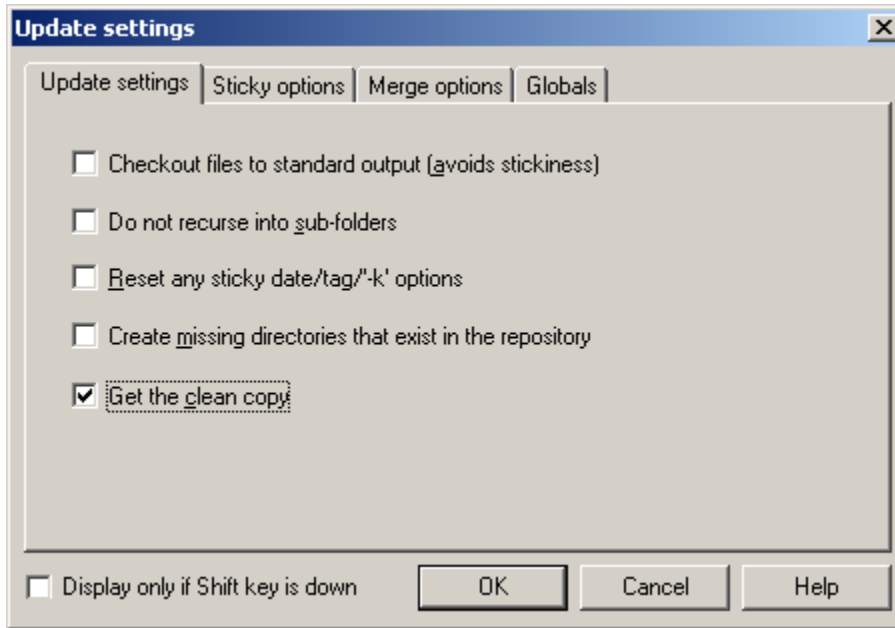
4.16 Viewing a Previous Revision using Update

It is often useful to view the differences between different revisions or working copies of a file. It is also useful to view an entire copy of a previous revision of a file. As mentioned in Section 4.15, the graph command is a good starting point for revision comparison or checking out a temporary copy of a previous revision.

Since viewing a previous revision of a file is such a common operation, it is often done using the **update** command rather than going through the graph window. The disadvantage of **update** is that a revision of file modified in the working directory (if present) will be over-written. To retrieve a previous revision of a file, run the **update** command as described in Section 4.9.2. Check the **Retrieve rev/tag/branch** box and enter the desired revision number, branch or tag as shown in the following example where revision 1.8 of the file is requested:

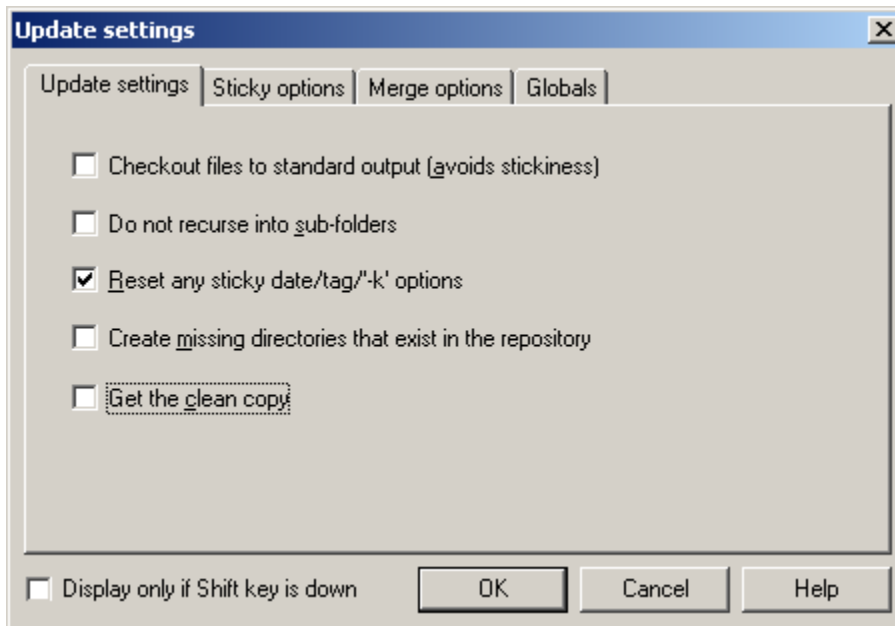


This operation will not have the desired effect if the working copy of the file had been modified prior to running the **update** command. Instead of getting revision 1.8, a merged revision of the local working copy of the file with revision 1.8 will be created. If a locally modified revision of the file exists, the modifications should be discarded by selecting the **Get the clean copy** option as shown here:



Note that the locally modified revision will not actually be over-written. Instead, it will be renamed with a name such as `#.admin.c.1.4`. This file should be removed when it is no longer needed.

After the previous revision has been viewed, the working directory should be restored to the current revision. In the case of a standard working directory working on the trunk of the repository, this is done by selecting the **Reset any sticky date/tag/'-k' options** box on the **Update settings** panel as shown here:

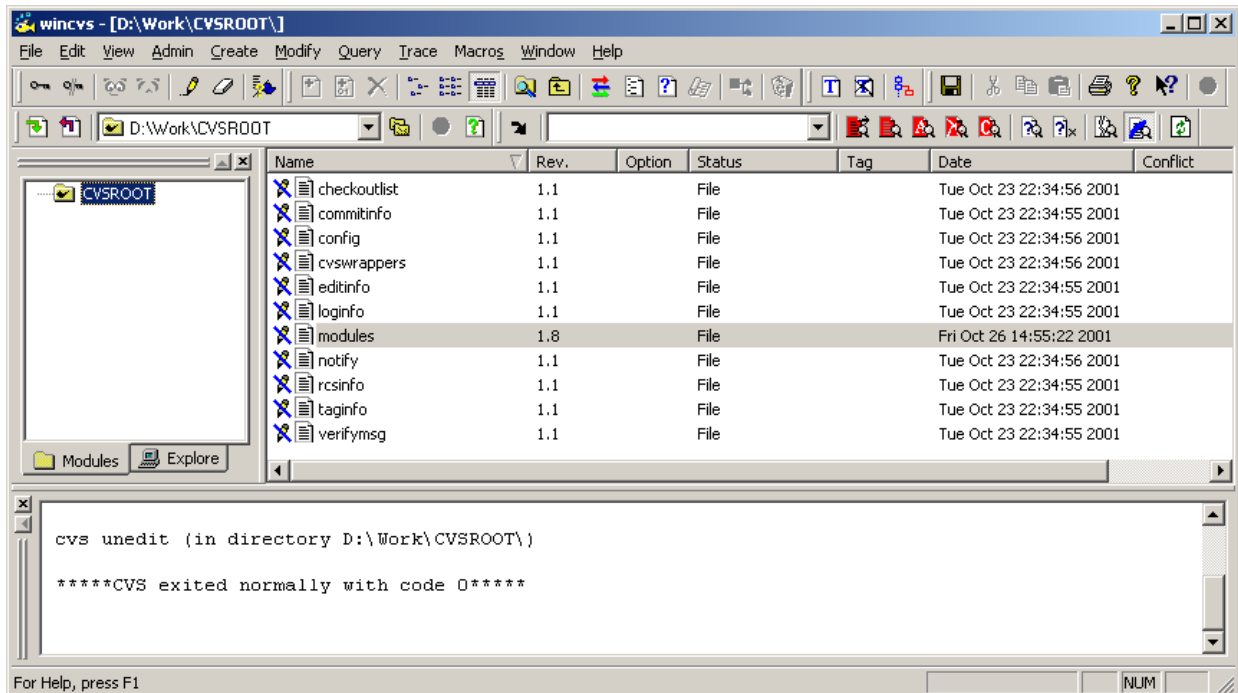


Section 5 – Administrative Commands

This section covers activities that should normally be used only by a CVS administrator.

5.1 Maintaining CVS Administrative Files

To edit any administrative files, the CVSROOT module must be checked out to a local working directory. Refer to Section 4.7 for information on the **checkout** command. A working directory with a checked out version of CVSROOT is shown here:



Notice that there are 11 administrative files in the CVSROOT module that provide a way to manage many advanced features of CVS. See Appendix C of Cederqvist for a complete description of all these files and features.

The following section shows an example of editing the CVS modules file. Other administrative files can be maintained in the same way.

5.1.1 Editing the Modules Administrative File

The modules file in CVS lists names for folder hierarchies that can be checked out from the repository using the **Checkout module** command in the WinCvs **Create** menu. Using the modules file, any part of a single module hierarchy or multiple existing modules can be aliases with a single module name.

If a module is imported to CVS and not listed in the modules file, the module may still be checked out but it will not be listed in response to the **Admin->Macros admin->Get the cvs modules** command in the WinCvs menu.

In order to edit the modules file, the CVSROOT module must first be checked out as mentioned in the introduction to Section 5. The file must also be given write access as discussed in Section 4.10.1.

A sample modules file is shown here:

```
# Three different line formats are valid:
#   key-a   aliases...
#   key [options] directory
#   key [options] directory files...
#
# Where "options" are composed of:
#   -i prog      Run "prog" on "cvs commit" from top-level of module.
#   -o prog      Run "prog" on "cvs checkout" of module.
#   -e prog      Run "prog" on "cvs export" of module.
#   -t prog      Run "prog" on "cvs rtag" of module.
#   -u prog      Run "prog" on "cvs update" of module.
#   -d dir       Place module in directory "dir" instead of module name.
#   -l          Top-level directory only -- do not recurse.
#
# NOTE:  If you change any of the "Run" options above, you'll have to
# release and re-checkout any working directories of these modules.
#
# And "directory" is a path to a directory relative to $CVSROOT.
#
# The "-a" option specifies an alias.  An alias is interpreted as if
# everything on the right of the "-a" had been typed on the command line.
#
# You can encode a module within a module by using the special '&'
# character to interpose another module into the current module.  This
# can be useful for creating a module that consists of many directories
# spread out over the entire source repository.
#
CVSROOT -a CVSROOT
cvs-1.11.1p1 -a cvs_1.11.1p1
cvs -a cvs-1.11.1p1
```

Section C.1 of the Cederqvist manual describes the modules file in detail.

After modifications are made to the modules file, it should be committed to the repository just as with any normal file. The messages in the Output Pane include an additional message for administrative files as shown below where the CVS server indicated that the administrative file database was rebuilt:

```
cvs commit -m "no message" modules (in directory D:\Work\CVSROOT\
Checking in modules;
/store/sample/CVSROOT/modules,v <-- modules
new revision: 1.9; previous revision: 1.8
done
cvs server: Rebuilding administrative file database

*****CVS exited normally with code 0*****
```


5.2 Recovering from Locked Repository

Internally, CVS uses a lock in each repository to prevent simultaneous access by multiple users. The lock is actually a directory in the repository named `#cvs.lock`. If a CVS operation aborts, it is possible that this lock will remain in the repository preventing any further operations from proceeding. In this case, the message “waiting for user’s lock” will be printed in the status window as shown in the following example where user `don` is the user holding the lock:

```
cv$ admin -u admin.c
```

```
*****CVS exited normally with code 0*****
```

```
cv$ server: [13:46:22] waiting for don's lock in /Store/Cvs-Admin/cvs-1.11.1p1/src
```

Normally, this message means that user `don` is in the process of updating the repository. This should normally not take very long and the lock will be available as shown in the following example of an **`unlock`** command that was temporarily delayed:

```
cv$ server: [13:46:52] waiting for don's lock in /Store/Cvs-Admin/cvs-1.11.1p1/src
cv$ server: [13:47:22] obtained lock in /Store/Cvs-Admin/cvs-1.11.1p1/src
RCS file: /Store/Cvs-Admin/cvs-1.11.1p1/src/admin.c,v
1.5 unlocked
done
```

In cases where CVS aborted for some reason (such as a network timeout or machine reboot), the lock may need to be removed manually. The `#cvs.lock` directory should be located in the indicated repository and removed. For UNIX repositories, use `rmdir` or a similar command. For Windows repositories, remove the folder from the Windows Explorer.

5.3 Release Management

The basic mechanism for maintaining multiple releases of a product using CVS is the tag. Tags are described in Section 4.4 of the Cederqvist manual. When a product has been tested and ready for release with a given set of file revisions, the files can then be tagged with a symbolic name. Files may be modified after the release but the tagged revision can always be retrieved.

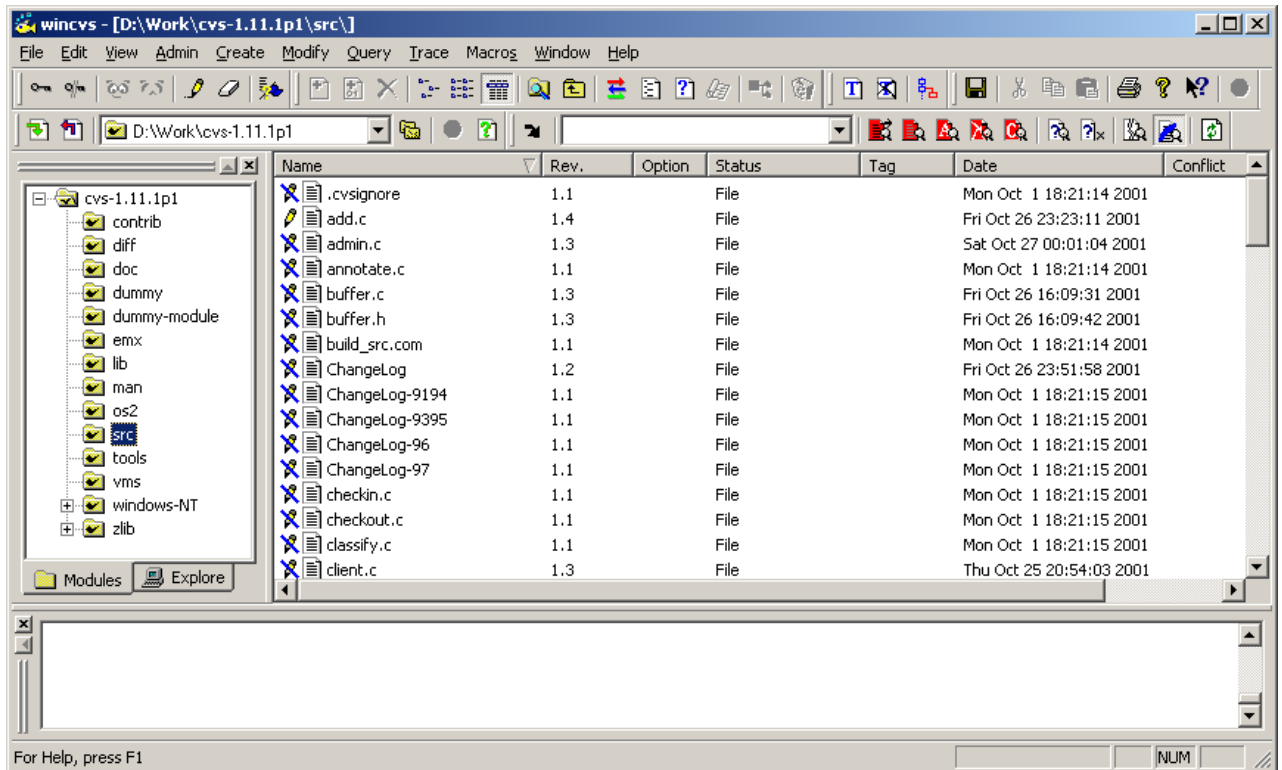
The problem that always comes up in release management is what to do when a bug is found after a product version has been released. Presumably, the files in the repository have been modified since the release and are not likely to be stable enough to be included in a bug fix release.

The solution to this problem is to create a branch from the version of the repository tagged at release time. Files can then be modified in the branch without interfering with continuing development on the trunk. After the fixes have been verified, the original release tag can be moved to the new file revisions or a new release tag can be created.

The following two sections show examples for both moving a tag to a new revision and creating a new release tag.

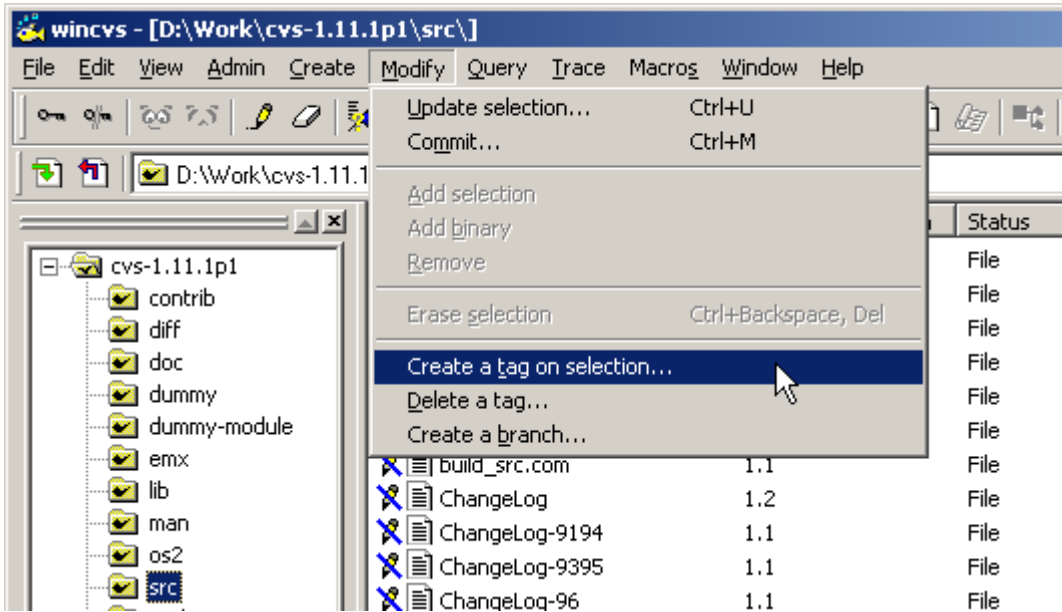
5.3.1 Tagging a Product Release

When a given set of files has been tested and ready for release, the working directory from which the release was built should be tagged. In the following example, a working directory for the `src` module of CVS is displayed. A partial list of files for the `src` folder is shown to illustrate a set of revision numbers (1.1, 1.2, 1.3, and 1.4).

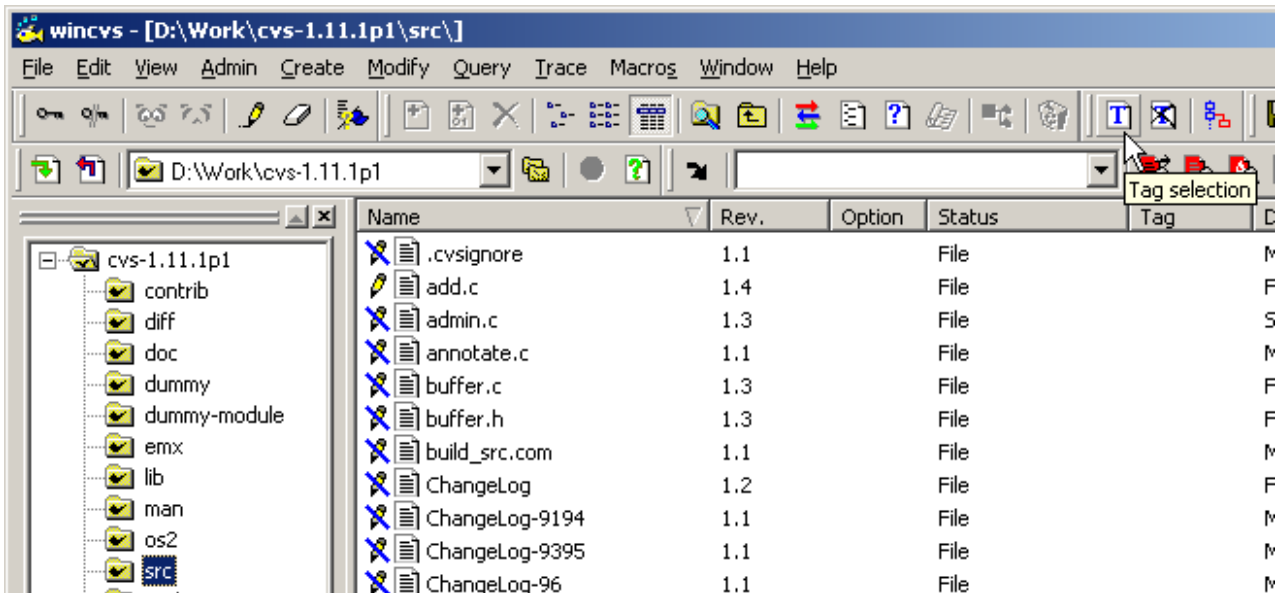


Assuming the `src` working directory represents a tested set of files, the files can be tagged for release using the `tag` command.

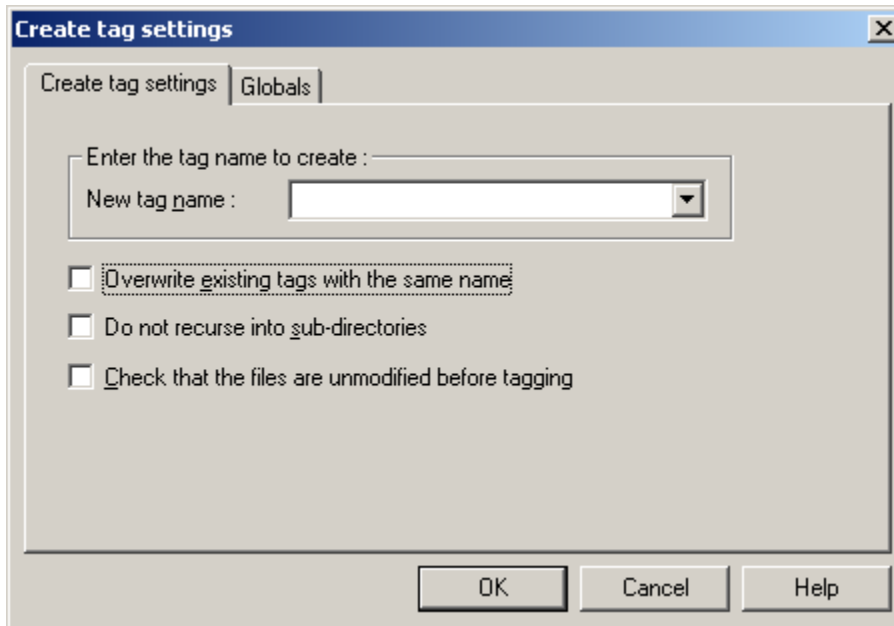
The **tag** command operates on a selected file or folder (recursively). The **tag** command can be invoked from the WinCvs menu by selecting **Create a tag on selection...** from the **Modify** menu as shown here:



The **tag** command is also available as an icon on the taskbar as shown here:



When the **tag** command is invoked, the Create tag settings dialog is displayed as shown here:



The Create tag settings panel provides a field to enter the tag name and three options. The tag name may not contain any of the following characters: \$, . : ; @.

The **Overwrite existing tags with the same name** option specifies that any existing tags found with the same name should be moved to the revision in the current working directory.

The **Do not recurse into sub-directories** option may be useful in cases where a snapshot of a single folder is desired.

It is probably a good idea to use the **Check that the files are unmodified before tagging** option since it does not make much sense to have files in the release working directory that are modified but not committed to the repository.

After the desired tag name and options are selected, click **OK** to begin the tagging process. During the tagging operation, CVS will display the names of the files being tagged as shown here:

```
cvs tag -c march-beta-release (in directory D:\Work\cvs-1.11.1p1\src\)  
cvs server: Tagging .  
T .cvsignore  
T ChangeLog  
T Makefile.am  
T Makefile.in  
T add.c  
T admin.c  
T annotate.c  
T buffer.c  
T buffer.h  
T wrapper.c  
T zlib.c  
  
*****CVS exited normally with code 0*****
```

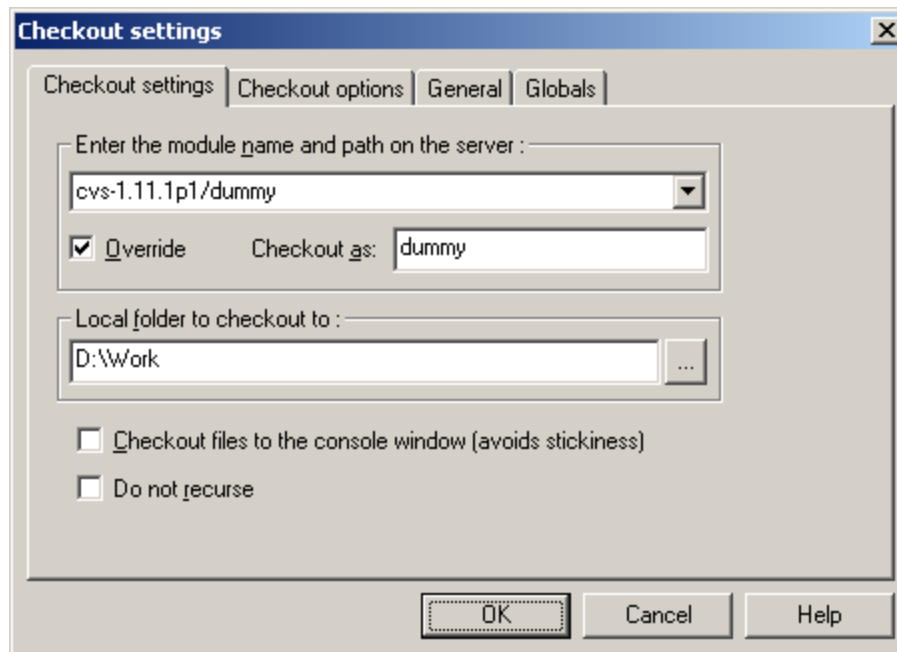
5.3.2 Fixing Bugs after Product Release

When a bug is discovered in a released product, the usual goal is to produce an incremental release with the minimum set of changes required to fix the bug. The first step is to create a working directory that can be used to build the incremental release. Depending on the product, this may require as little as a single file or as much as the entire repository tree. The working directory will be set up to contain the revision of all files tagged at the time of the release.

5.3.2.1 Creating a Working Directory from a Tagged Release

If a working directory for the product or module already exists, the working directory can be updated to the tagged version using the **update** command (Section 4.9.2). This method can easily lead to undesired results, however, such as when modified files exist in the working directory or when the folder hierarchy is different than the original tagged revision.

The preferred method to create a working directory from a tagged release is to use the **checkout** command (Section 4.7) to check out a clean copy of the release to a new working directory. When the **checkout** command is invoked, the **Checkout settings** panel will be displayed as shown here:

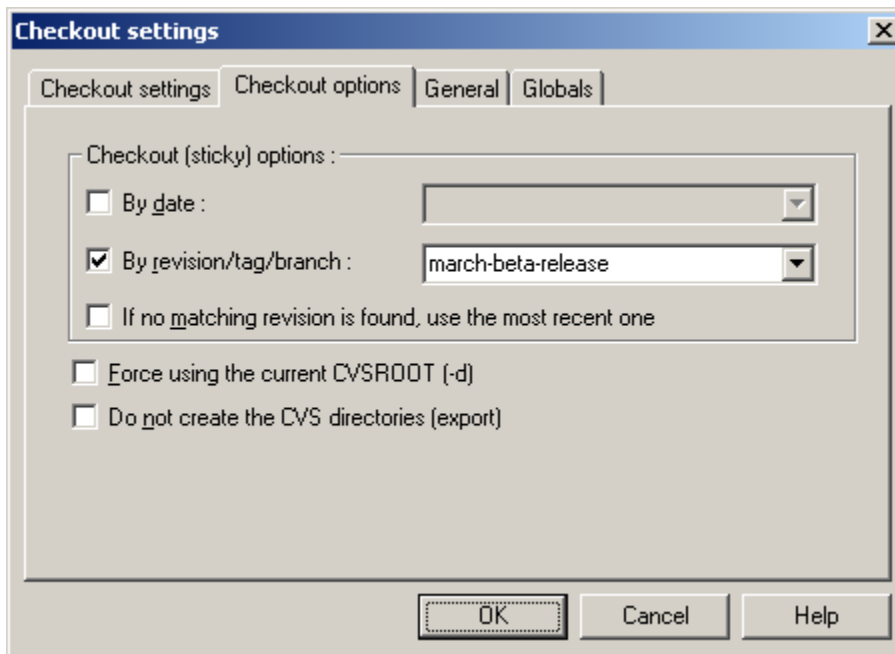


Enter the name of the module or sub-module to be checked out as shown in the example above where the **dummy** directory of the **cvs-1.11.1p1** module is being checked out.

If appropriate, the **Override** box can be checked to checkout the module with the specified folder name. In the above example, this feature has been used to checkout the **dummy** module to **D:\Work\dummy** instead of **D:\Work\cvs-1.11.1p1\dummy** which would be the case without the **Override** box checked.

Select the folder where the working directory will be created in the **Local folder to checkout to** field. A sub-folder will be created in the specified folder to checkout the module.

After the desired folder is selected, display the Checkout options tab as shown here:



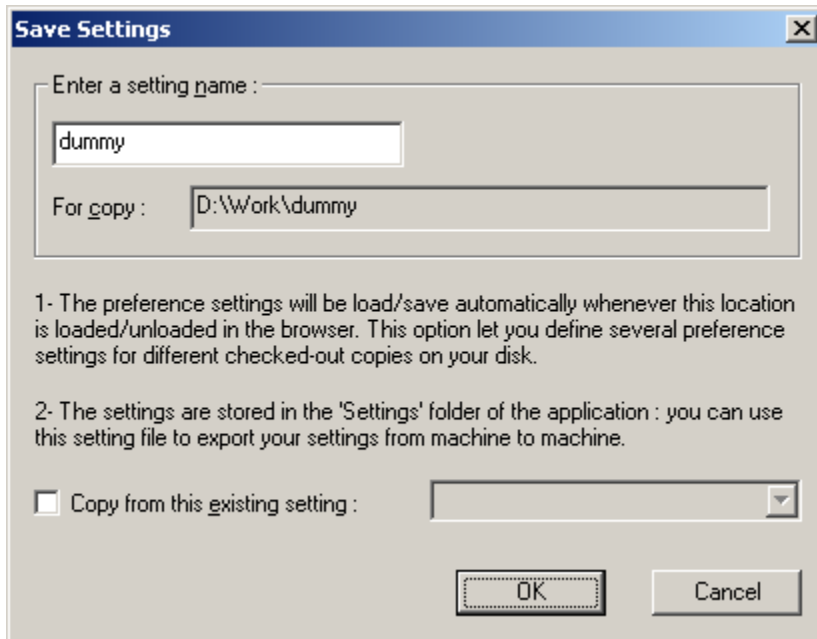
Check the **By revision/tag/branch** box and specify the tag for the desired release. After the tag name is entered, click **OK** to initiate the checkout operation.

During the checkout operation, CVS will display the name of each file in the WinCvs Output Pane as shown here:

```
cvs checkout -r march-beta-release -d dummy cvs-1.11.1p1/dummy (in directory
D:\Work)
cvs server: Updating dummy
U dummy/.cvsignore
U dummy/ChangeLog
U dummy/add.c
U dummy/admin.c
U dummy/annotate.c
U dummy/buffer.c
U dummy/buffer.h
.
.
U dummy/zlib.c

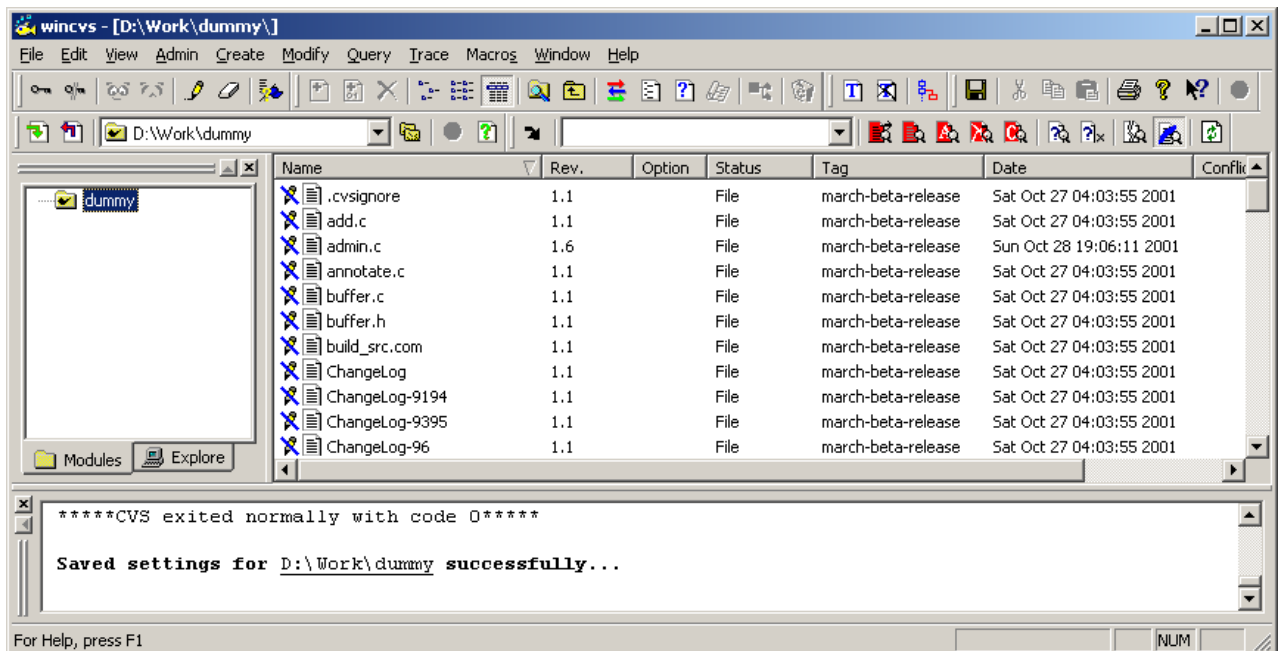
*****CVS exited normally with code 0*****
```

After the checkout is complete, use the **View->Browse Location->Change** command from the menu or tool bar to point to the newly created working directory. The Save settings panel will be displayed as shown here:



Click **OK** save the current WinCVS Preferences for use when accessing this folder. An advanced option exists to copy from another previously saved setting if desired.

An example of a working directory created from the march-beta-release tagged revision is shown here:



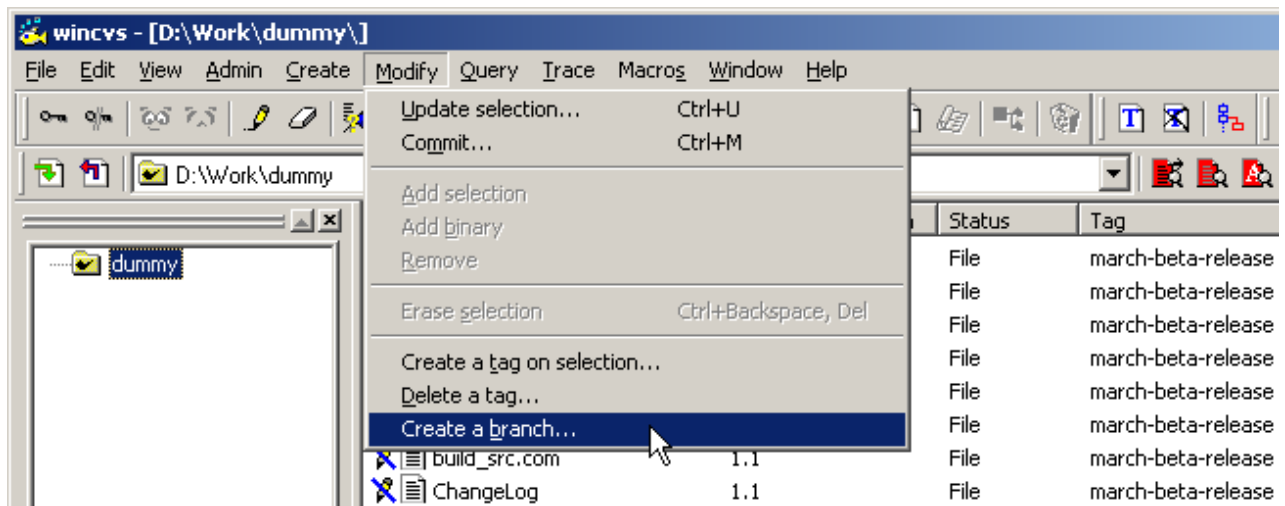
Now that a working directory has been created, a branch must be created to allow modification of the tagged files. The following section explains the way to create a branch from the tagged revisions in the working directory just created.

5.3.2.2 Creating a Branch from a Tagged Working Directory (fork)

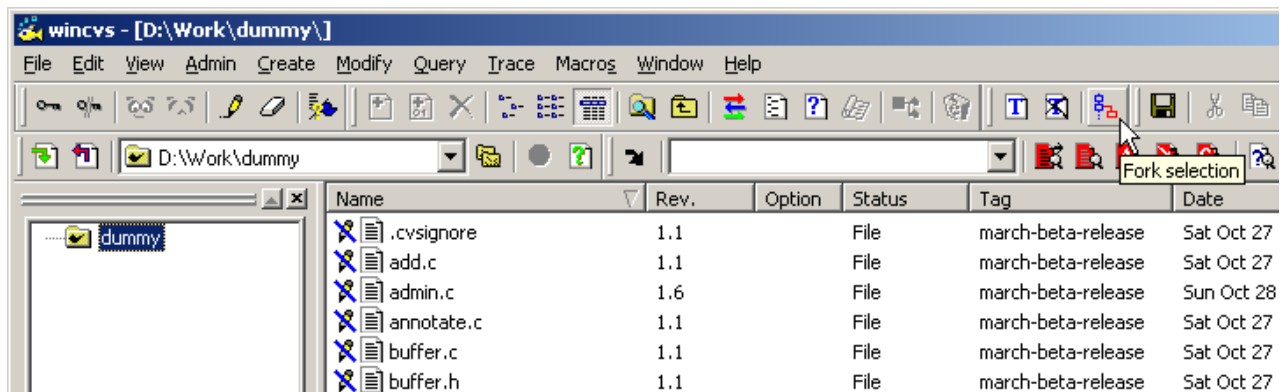
The files in a working directory updated to a tagged release cannot be modified and committed back to the repository. To make modifications, the working directory must be updated to a **branch** tag. If an appropriate branch already exists, use the **update** command (Section 4.9.2) to get the last branch revision into the working directory. If a branch does not exist, create a branch using the **fork** command explained in this section.

Prior to creating a branch, the instructions in Section 5.3.2.1 should be followed to create a clean copy of a tagged release in a working directory. Refer to Section 5.3.2.3 for an example of creating a branch without creating a work area first.

The **fork** command operates on a selected file or folder (recursively). The **fork** command can be invoked from the WinCvs menu by selecting **Create a branch...** from the **Modify** menu as shown here:

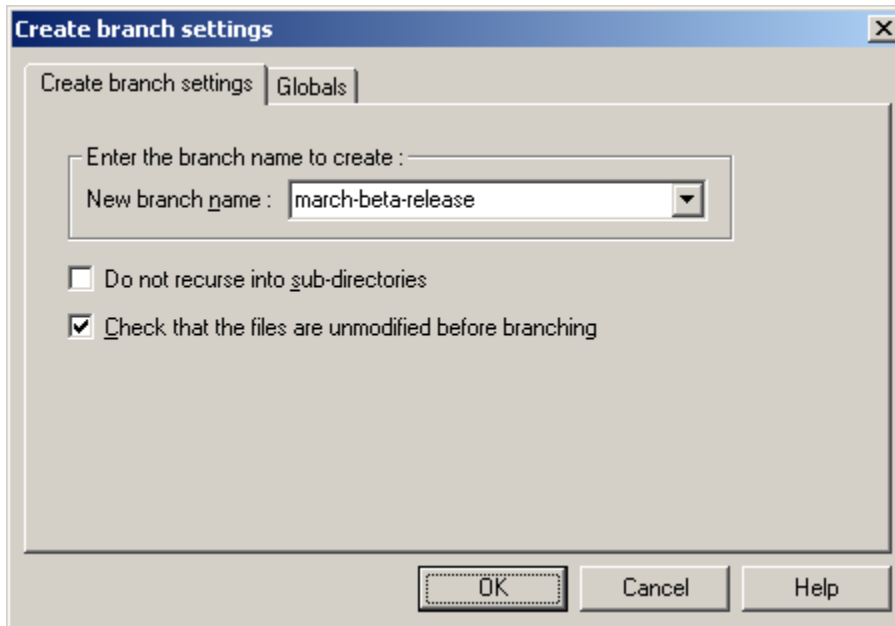


The **fork** command is also available as an icon on the toolbar as shown here:



The **fork** command is very similar to the **branch** command discussed in the following Section 5.3.2.3. In fact, both commands result in the creation of a branch. The only difference is that the **fork** command creates a branch from an existing working copy of the repository. The **branch** command does not make reference to a working copy.

When the **fork** command is invoked, the Create branch settings panel will be displayed as shown here:



It is a good idea to give the branch a name that clearly indicates its purpose. In the above example, the new development is being done from the march-beta-release tagged release so march-beta-branch is a logical name for the branch. Note that the branch name cannot be the same as the tagged release tag since they are both tags and tag names must be unique.

The Create branch settings panel has two options that may be appropriate in certain cases. It is normally a good idea to set the option to check for modified files since it doesn't make sense to branch from a directory of files that haven't been committed. It may be useful to set the **Do not recurse** option also since it is common to branch a single folder and not need branches in the sub-folders (the default).

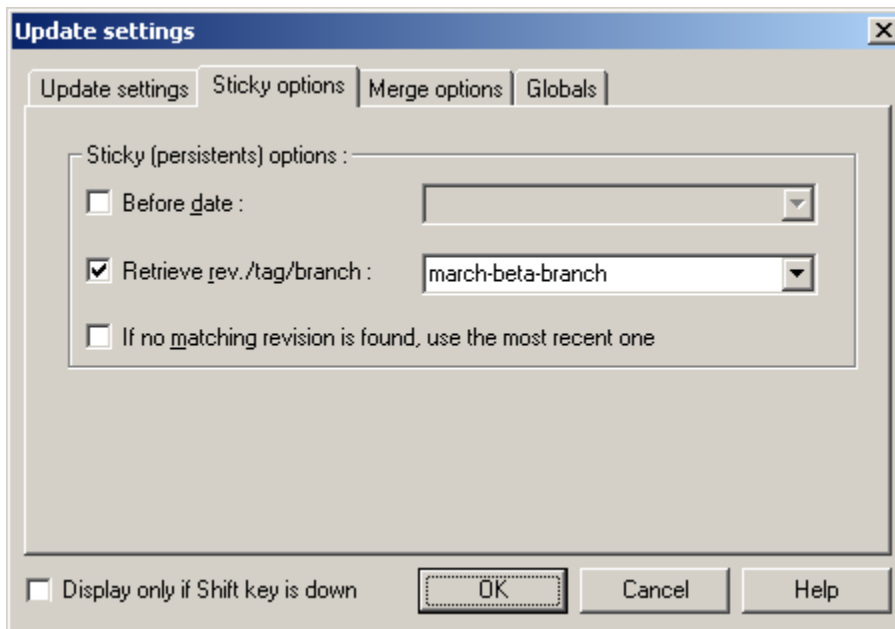
After specifying the branch name and any desired options, click **OK** to start the **fork** operation.

The status window will display the progress of the branch creation as shown here:

```
cv$ tag -b -c march-beta-branch (in directory D:\Work\dummy\)  
cv$ server: Tagging .  
T .cvsignore  
T ChangeLog  
T Makefile.am  
T Makefile.in  
T add.c  
T admin.c  
.  
.  
T zlib.c  
  
*****CVS exited normally with code 0*****
```

The branch will be created with the current revision of files checked out in the working copy. Notice that WinCvs uses the **tag** command with the **-b** option to create the branch. This command can also be entered manually from the WinCvs status window.

Note that simply creating a branch does not update the working directory to the branched version. It is necessary to use the **update** command (Section 4.9) to update to the newly created branch.



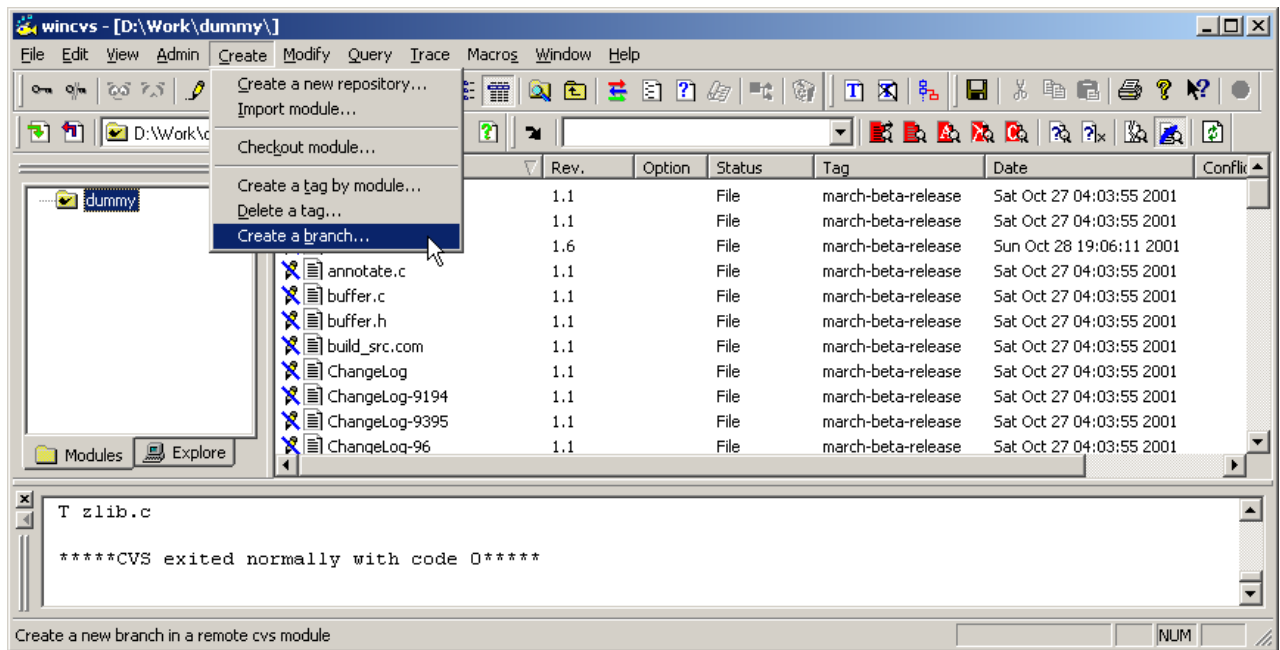
Once the branch is created and the working directory is updated, files can be modified and committed to the branch as described in Sections 4.10 through 4.12. When all modifications are complete and tested. There are two possibilities for creating a new tagged release described in Section 5.2.3.4.

5.3.2.3 Creating a Branch without a Working Directory (branch)

The files in a working directory updated to a tagged release cannot be modified and committed back to the repository. To make modifications, the working directory must be updated to a **branch** tag. If an appropriate branch already exists, use the **update** command (Section 4.9.2) to get the last branch revision into the working directory. If a branch does not exist, create a branch using the **branch** command explained in this section.

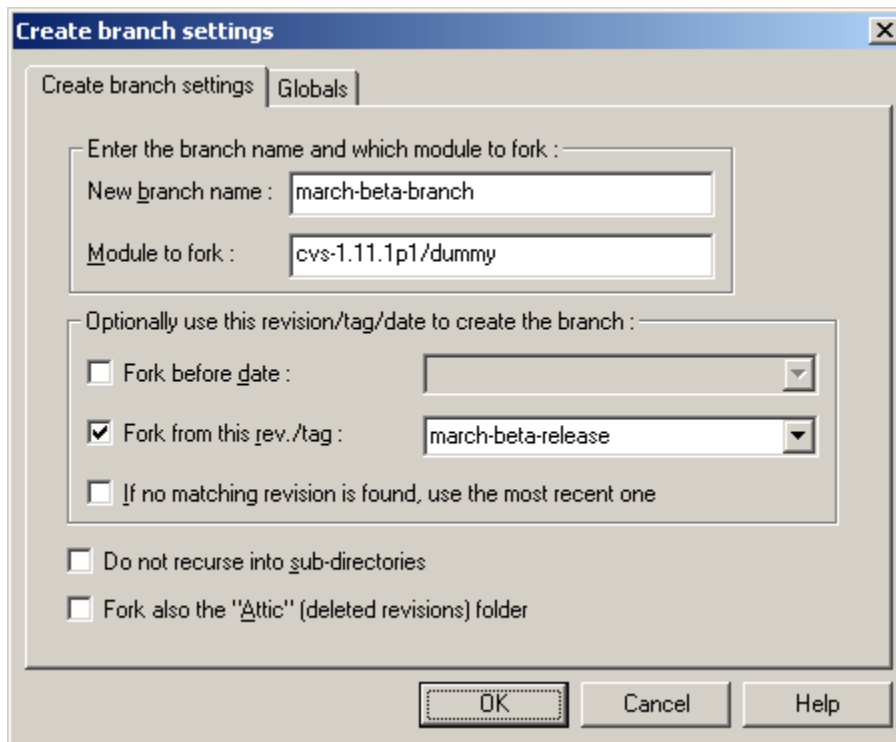
This section describes the creation of a branch from a tagged release in the repository without reference to any working directory. To create a branch from a current working directory, refer to Section 5.3.2.2.

The **branch** command is available only from the WinCvs **Create** menu as shown here:



The **branch** command is very similar to the **fork** command discussed in the preceding Section 5.3.2.2. In fact, both commands result in the creation of a branch. The only difference is that the **fork** command creates a branch from an existing working copy of the repository. The **branch** command does not make reference to a working copy.

When the **branch** command is invoked, the Create branch settings panel will be displayed as shown here:



Enter the desired branch name in the **New branch name** field. It is a good idea to give the branch a name that clearly indicates its purpose. In the above example, the new development is being done from the **march-beta-release** tagged release so **march-beta-branch** is a logical name for the branch. Note that the branch name cannot be the same as the tagged release tag since they are both tags and tag names must be unique.

There are three options in the box labeled **Optionally use this revision/tag/date to create the branch**. If none of these options is checked, the branch will be made from the head revision of each file in the repository trunk.

Enter the module to branch (fork) in the **Module to fork** field. Note that the full path of the module starting at the repository root must be specified.

The **Fork before date** option allows a branch to be created from revisions of all files dated on or before the specified date. In the above example, this option is not used.

The **Fork from this rev/tag** option allows a branch to be created from a tagged revision or from a specific revision number. Specifying a tag like the above example is the most common use of the **branch** command.

The **Do not recurse into sub-directories** option is useful if only one level of the repository needs to be branched.

The attic (deleted revisions) folder is normally not branched as part of a branch operation. To include the attic as part of the branch, check the option labeled **Fork also the “Attic” (deleted revisions) folder**.

After specifying the desired options, click the **OK** button to start the branch operation.

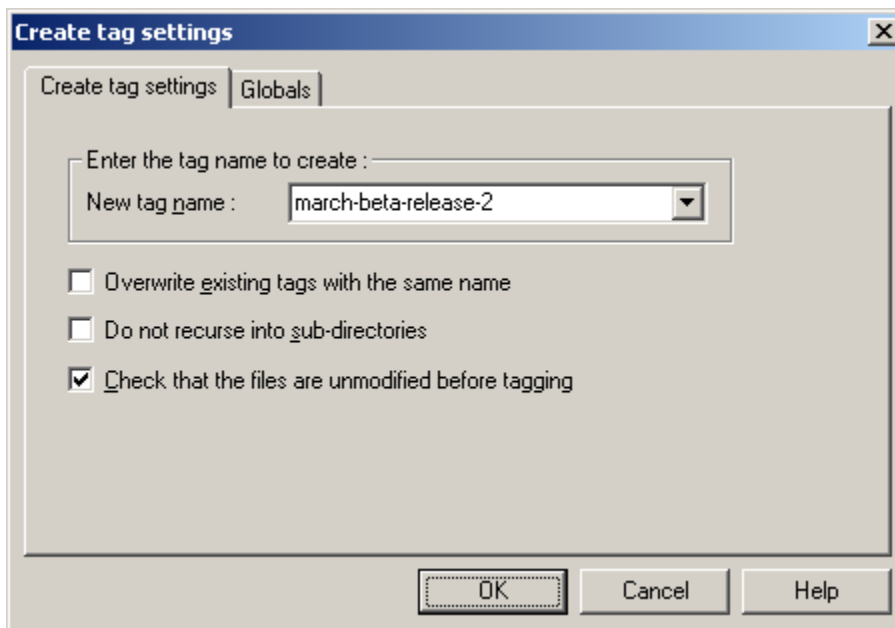
Once the branch is created, a working directory needs to be checked out before files can be modified and committed to the branch as described in Sections 4.10 through 4.12. When all modifications are complete and tested. There are two possibilities for creating a new tagged release described in Section 5.2.3.4.

5.3.2.4 Creating a New Tagged Release

So far, Section 5.3 has included examples of creating a tagged release, creating a branch from a tagged release, and preparing to modify files on the branch. The tagged release `march-beta-release` was created in the module `cvs-1.11.lpl/dummy` and a branch named `march-beta-branch` was created from the tagged release. After bug fixes or other modifications are made to the branched files, a new tagged release must be created for the interim release.

At this point, the release manager has two options: the modified files can be re-tagged with the original tag name `march-beta-release` or a new tagged release can be created. In cases where the original release has already been widely distributed to internal or external customers, the original tagged release should most likely not be modified so that it can be re-created if necessary. If the original release was not distributed, it may be reasonable to re-tag the modified files with the original tag. This is also referred to as moving tags since the tags are effectively moved from one revision to another for the modified files.

In either case, the `tag` command needs to be invoked as described in Section 5.3.1. The following settings would be used to create a new tagged release called `march-beta-release-2`:

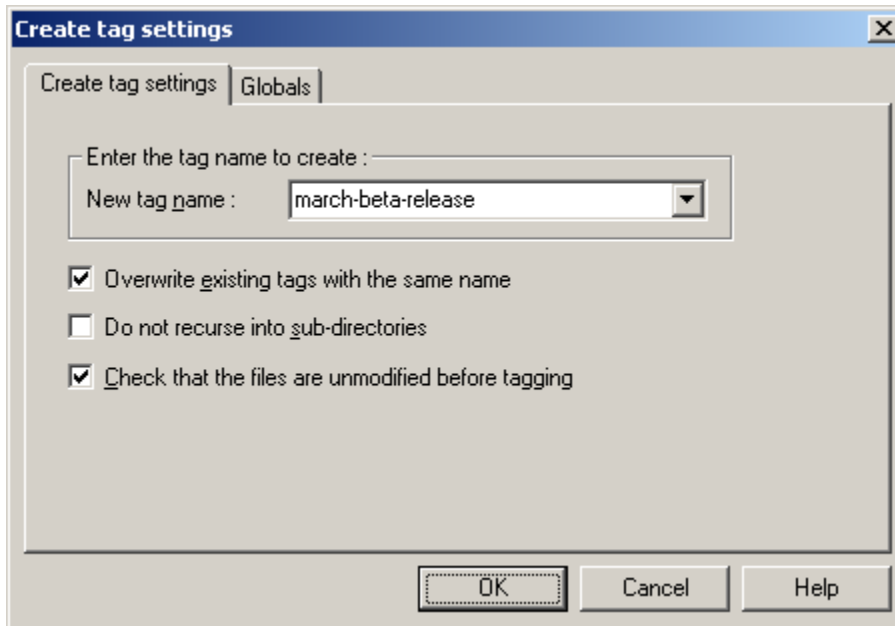


Note that the option to require unmodified files is checked. This is always a good precaution when tagging files since the modifications will not be part of the tagged release.

Click **OK** to start the tag operation. Messages similar to the following will be displayed in the Output Pane:

```
cvs tag -c march-beta-release-2 (in directory D:\Work\dummy\)  
cvs server: Tagging .  
T .cvsignore  
T ChangeLog  
T add.c  
T admin.c  
.  
.  
T zlib.c  
  
*****CVS exited normally with code 0*****
```

The following settings would be used to re-tag the new revisions with the original march-beta-release tag:



Checking the **Overwrite existing tags with the same name** option is what causes tags to be moved rather than trying to create new tags. In the above example, the tag operation would fail without this option since the specified tag already exists in the repository.

Note that the option to require unmodified files is checked. This is always a good precaution when tagging files since the modifications will not be part of the tagged release.

Click **OK** to start the tag operation. Messages similar to the following will be displayed in the Output Pane. Note that only files that were modified and committed since the branch was created will be listed in the output. These are the files whose tags are being moved as shown in this example where the file `admin.c` was modified:

```
cvstag -F -c march-beta-release (in directory D:\Work\dummy\)  
cvstag server: Tagging .  
T admin.c
```

```
*****CVS exited normally with code 0*****
```