

# Using SNIFF+ with CVS

## Introduction

---

CVS ("Concurrent Versions System") is a public-domain Version and Configuration Management System. It is based on RCS (Revision Control System) repository files, therefore migrating from RCS to CVS and back is simple. While CVS has all features known from RCS, it provides additional functionality. Within the SNIFF+ environment, the most useful features are that CVS

- Allows parallel development without the need for branches. Third party sources can be tracked. Changes of parallel development are merged automatically, where possible.
- Allows to access repositories not seen in the file system, minimizing network traffic and enforcing security and access policies.
- Allows to track checked-out revisions, making it possible to freeze (tag) any current configuration.

According to the author, *"RCS is [analogous to using] assembly language, while CVS is [like using] Pascal"*.

### In this paper you will learn how to

- Install the SNIFF+ CVS Adaptor
- Set up your Repository and SNIFF+ Projects using CVS
- Use one of two possible working schemes with SNIFF+ and CVS
- Configure the SNIFF+ CVS Adaptor to your needs
- Avoid the most common pitfalls when using CVS with SNIFF+

All tasks to be done are explained in detailed step-by-step instructions, so you can quickly get a test installation running.

### Assumptions made in this paper

You should have a working CVS installation on your system, and you should be able to execute CVS commands from your Operating System Command Line. Although SNIFF+ helps you performing CVS commands, you should know the CVS concept and how to operate the CVS system from the command line as well.

Note that this paper does not replace an introduction on CVS. For background information, please refer to the resources listed below.

You should also be familiar with the SNIFF+ Project and Working Environment concepts.

---

## Feedback

CVS is a sophisticated system, and there are many ways of working on it. Your feedback on the CVS integration is always welcome. This can be suggestions for better integration, added functionality or hints to use CVS more efficiently with SNIFF+.

Please send all e-mail to the addresses listed below and mention the word CVS in the subject.

## TakeFive Support

### Europe:

sniff-support@takefive.co.at

### USA:

sniff-support@takefive.com

## Other Useful links

SNIFF+ web pages:

- SNIFF+ Frequently Asked Questions  
<http://www.takefive.com/support/faq.html>
- SNIFF+ Support Searchable Knowledge Base  
<http://www.takefive.co.at/support/kb.html>
- SNIFF+ Users Mailing List  
<http://www.takefive.com/support/sniff-list.html>
- Customer Newsletter  
[http://www.takefive.com/news/customer\\_newsletter.htm](http://www.takefive.com/news/customer_newsletter.htm)

---

## Version and Compatibility information

---

|                       |                                     |
|-----------------------|-------------------------------------|
| CVS Adaptor Version   | cvs-3.2.1a (03-Apr-2000) and higher |
| SNiFF+ Versions       | 3.0.1 and higher                    |
| CVS Versions          | 1.9 and higher                      |
| This Document Version | cvs-3.2.1a                          |

This document refers to the python-based CVS Adaptor introduced with SNiFF+ 3.0 and is not to be confused with previous shellsript-based Adaptors. This CVS Adaptor comes as part of the SNiFF+ product from versions 3.0.2 and above and only needs to be enabled (see [Enabling the Adaptor — page 8](#)). More recent releases can always be downloaded from the [SNiFF+ FTP Server \[1\]](#).

Compared to the previous adaptor, the new one has been enhanced in the following areas:

- It is based on python, therefore it runs on both Windows and UNIX.
- CVS remote Repositories can be used with SNiFF+.
- It provides support for SNiFF+ Shared Working Environments, thereby saving both disk space and compile time.
- Custom menus have been extended.

## Where to get CVS Software and Information

You can get the latest CVS distributions from the [CVS Homepage \[2\]](#) currently hosted by SourceGear. [CVS Bubbles \[3\]](#) by Pascal Molli has links to other resources.

The [CVS Manual \[4\]](#) by Per Cederqvist is the definitive reference, is well readable and includes a good introduction. It is available in PDF, HTML and Windows Help formats. For beginners that want to start quickly, the [CVS tutorial \[5\]](#) by Jim Blandy is worth reading. Brian Berliner put together a paper on the theoretical background of CVS [CVS II: Parallelizing Software Development \[6\]](#).

---

# Installing the SNIFF+ CVS Adaptor

---

If you are using SNIFF+ 3.0.2 or later, all adaptor files are already part of the distribution, and you just need to enable it (see [Enabling the Adaptor — page 8](#)). If you are using SNIFF+ 3.0.1 or below, or you are updating to a new version of the CVS Adaptor, you need to complete the following steps:

1. Download the newest adaptor from the [SNIFF+ FTP Server \[1\]](#)
2. Unpack the adaptor archive (`cvsv-<version>.tar.gz` or `cvsv-<version>.zip`) into any directory you like.
3. Copy all files to their corresponding locations in your SNIFF+ installation. This can be performed very easily with the Windows Explorer or with the UNIX `cp -R` command.
4. If you have existing custom menus, you need to merge the new ones (`SiteMenus.CVS.sniff`) with your existing ones (`SiteMenus.sniff`) in the `$_SNIFF_DIR/config` directory. If you are not yet using custom menus, you can just copy the `SiteMenus.CVS.sniff` file to `SiteMenus.sniff`.

## Enabling the Adaptor

To enable the custom menu used for running CVS, you need to edit your custom menu definition file `$_SNIFF_DIR/config/SiteMenus.sniff` and uncomment the corresponding entries: In more recent versions of SNIFF+, you need to remove the tilde character in front of a `^~ProjectEditor` entry, in older versions you need to remove hash (`#`) characters:

- In the **Project Editor**, there are three menus: *CVS*, *CVS Modules* and *CVS Admin*. Don't forget to also uncomment the `^ProjectEditor` entry.
- In the **Source Editor**, there is one menu: *CVS*.

The entries in the **CVS Admin** menu are used for administrative purposes only. Therefore we suggest you leave these commented out in your `SiteMenus.sniff` and only add them to the CVS Administrator's `UserMenus.sniff`.

We also recommend to keep the **CVS Modules** menu commented out for users who are not yet familiar with CVS. You can perform all necessary actions with the CVS menu as well, while Workspace Administrators and people who are already familiar with CVS might like to use the CVS Modules commands for enhance performance (especially when using a remote repository).

---

# Setting up a SNIFF+ project with CVS

---

In this section, we give step-by-step instructions how to set up a CVS repository with SNIFF+. Although these steps can be completed by any user, we strongly suggest that you appoint a CVS administrator for larger projects. Only the administrator should be allowed to change CVS configuration files or import new modules. Refer to the [CVS Manual \[4\]](#) for details on how to enforce access and security policies.

Most of the steps listed below will not need to be completed if you are already working with CVS outside of SNIFF+. Yet you should quickly read through this passage since it contains information on how to make SNIFF+ fit for using CVS efficiently.

## Defining the Repository

Before starting SNIFF+, set the environment variable CVSROOT to the directory, which will become the repository root:

### On Unix

```
setenv CVSROOT /your/absolute/path/to/repository/root
```

### On Windows

```
set CVSROOT=:local:d:\your\absolute\path\to\repository\root
```

(:local: is needed for Windows to work with drive letters)

The CVSROOT variable specifies the location of your repository and will be used by all CVS commands.

---

**Note**

Windows UNC pathnames (like \\host\path) are **not** supported with the current version. However, CVS Remote repositories can also be used with the CVS Adaptor. Please refer to [Advanced topics — page 19](#) for details.

Note that the SNIFF+ CVS Adaptor does not depend on CVSROOT being set, because it takes the information from the RWE; however, it is recommended to set CVSROOT in order to be able to work with CVS from outside SNIFF+ as well.

If you are setting up a new repository which doesn't have any files in it yet, you should now initialize it by executing the command

```
cvs init
```

---

from your command line.

**Note**

We assume that your CVS software is already installed and configured. In most cases it should suffice to set put the CVS executable in your path and set the CVSROOT environment variable. Please refer to the [CVS Manual](#) for detailed information on how to set up CVS.

You should now check if access to your repository works by entering the command

```
cvs co -p CVSROOT/modules
```

on your command line. This should print out the list of modules defined by CVS. Please refer to the [CVS Manual](#) for troubleshooting up to this point.

## Setting up the Repository for SNIFF+

In order for the integration to work smoothly, SNIFF+ needs to know the CVSROOT and CVS needs to know about SNIFF+ temporary file types. To complete these settings:

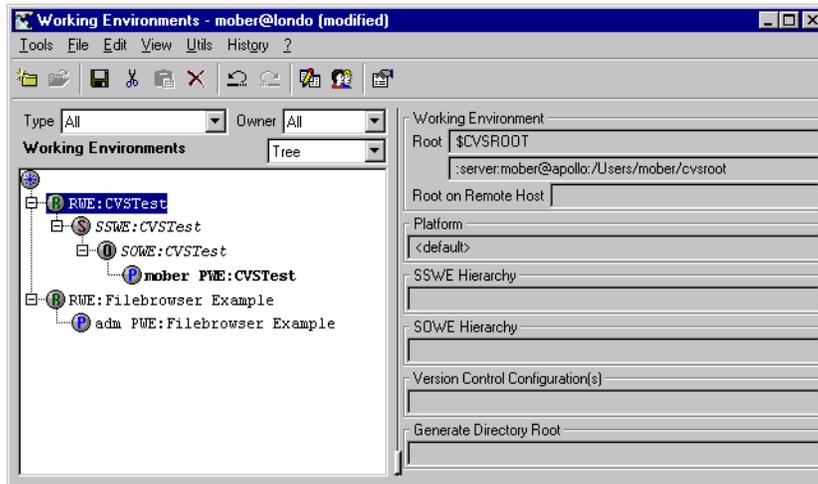
1. Start SNIFF+.
2. In the Working Environments Tool, create a new RWE for your CVS Repository. Enter \$CVSROOT as the directory root for this repository. Verify that the CVSROOT environment variable is correctly expanded.
3. Below the CVSROOT RWE, Create a shared source that will hold a checked-out version of your CVS controlled projects.

**Note**

The CVS Repository must not be in the same physical directory branch as your other working environments. Having the CVS Repository directory on the same physical directory as your ImportWE, SSWE or PWE does NOT work! For your first-time experiments we propose you create a directory structure like

```
Cvstest
+- rwe
+- sswe
+- pwe
+- ...
```

- Below the CVS SSWE, create SOWEs and PWEs as usual. Your working environments should now look more or less like this:



## Importing your Project into CVS

There are three possibilities to get your sources into the CVS repository:

- **Migrating from RCS** (or other VCS systems). You can do this without losing any information: see [Migrating from other VCS tools to CVS — page 21](#)
- **Using CVS import** for third-party source tracking or fast import of large source trees. See the [Command Reference — page 23](#) on how to accomplish this. If you already have your project in CVS, please proceed with [Initial Checkout of non SNIFF+-Controlled Projects — page 14](#)
- **Using SNIFF+ File Commands for initial check-in**. This is the simplest and most recommended procedure as it works exactly the same way as known from RCS; it is a bit slower than the CVS import variant, but it doesn't create an unwanted vendor branch.

The third option above is the simplest one, therefore we describe it in detail in terms of the complex example here:

1. Copy the *complex* sources from the SNIFF+ distribution into the Shared Source or Private Working Environment you created in step 4 [Setting up the Repository for SNIFF+ — page 10](#)

On Windows, you can use the Explorer; on Unix, you can type

```
cd <your CVS Import Working Environment>
cp -R $SNIFF_DIR/example/c++/complex_dir/user/complex.
```

2. In the Working Environments Tool, select your SSWE or PWE and choose **File > New Project... > with Defaults....** In the Directory Name Dialog, select your main project directory (i.e. *complex*).

- 
3. In the Attributes of New Project Dialog, choose the **General** node and make sure that the **Create Subproject Tree** checkbox is activated.
  4. Select the **Version Control System** Node and select **CVS** from **VCS Tool**.
  5. Choose OK to create the project. The Project Editor opens. You should see the three CVS custom menus.
  6. Select any file to enable the custom menus.

---

**Note**

For most CVS commands in the custom menus (including the module related commands), it is necessary to select a file first. SNIFF+ will use the *name of the SNIFF+ project to which the file belongs* as its CVS module name.

7. If you don't have a CVS Repository yet, select first **CVS Admin > Init CVS Repository**. This will perform a `cvs init` command to create initial CVS administrative files in the Repository.
8. Select **CVS Admin > Add SNIFF+ cvsignore entries**. This will add the SNIFF+ specific file patterns like `.sniffdir` to the CVS ignore list, which is necessary to allow correct import of projects (otherwise the SNIFF+ temporary files would also be version controlled after an import command).
9. Choose **CVS Admin > Show cvsignore file** to verify that the entries have been correctly added to the repository.
10. Select all your files and choose **File > Check In...**. A dialog box with the standard SNIFF+ parameters for Check In appears. For initial Check In, you can leave these empty.

---

**Note**

As you check in your sources, with every directory that was unknown before, there will appear a dialog box asking for the repository directory where you want to put your sources. In most cases, you can leave the guess by SNIFF+ as it is; however, if you want to put your sources in a different remote path than the local one, you may change the directory location.

Watch the messages CVS produces in your shell. Now, your sources are in the CVS repository. If you only want to use the File related CVS commands, you are done; only if you want to use `cvs rtag` and other purely module related commands, you need to define CVS modules.

## Defining CVS Modules

All those SNIFF+ projects that you want to be able to access as "master modules" need to be defined in your CVS modules file. A "master module" is one that is accessible by the **CVS Module** commands. The connection between a SNIFF+ project and a CVS module is by its name: a CVS module name is always constructed by removing the `.shared` extension from the corresponding SNIFF+ project name. So, for example, the SNIFF+ Project `complex.shared` must be assigned the CVS module `complex` no matter where in the directory hierarchy this module resides!

You can easily edit the modules file by choosing **CVS Admin > Edit modules File**.

In the modules file, lines starting with a hash sign (#) are comments and can be ignored. Your new entries should be added at the end of the modules file.

Each line contains the name of a module and, separated by whitespace, it's location relative to the repository root.

If you imported your entire Working Environment into "." (= the repository root) before and want to be able to access all modules directly, this is what you should enter into your modules file:

```
complex          complex
complexlib      complex/complexlib
iolib           complex/iolib
```

If you imported your project into `sniff/examples`, this is what you would enter:

```
complex          sniff/examples/complex
complexlib      sniff/examples/complex/complexlib
iolib           sniff/examples/complex/iolib
```

So the modules file provides the connection from SNIFF+ project locations to the repository locations.

Note that since all CVS module commands will work with the given module and all its subdirectories, it is not necessary to assign a CVS module to all SNIFF+ modules. In the example above, it would indeed suffice to define the `complex` module only since accessing `complex` will automatically access `iolib` and `complexlib` as well. However, in that case it would not be possible to check out the `complexlib` module by itself.

For more information about other options in the modules file, please read [CVS Manual](#).

---

## Initial Checkout of non SNIFF+-Controlled Projects

Since you don't have any SNIFF+ controlled project yet, you need to check out your existing sources using the cvs commandline interface. The usual way of working with CVS in SNIFF+ is to have all files read-only and change them to writable only by executing a `cvs edit` command. Therefore, you should also check out everything read-only.

### Note

You can configure CVS to automatically do all check-out operations read-only by editing your `$HOME/.cvsrc` file or by setting the environment variable `CVSREAD` to 1.

For optimal SNIFF+ project setup, you should check out your source modules from the command line in a way that allows maximum sharing of submodules: For SNIFF+, it is important that submodules used in different supermodules can always be accessed at the same point in your directory hierarchy; usually, this is accomplished by having the repository directory structure match the structure of the checked-out sources as closely as possible.

### Note

If you are not using the CVS modules file, you can safely ignore the previous paragraph. It only applies for CVS setups with a sophisticated modules file.

Set up your SNIFF+ project in a workspace as usual. In the Project Attributes, select **CVS** as the version control tool (**VCS Tool**). You can do this most easily by selecting all projects and choosing **Project > Attributes of Checkmarked Projects...** When your project descriptions are set up, select all project description files, add them to the CVS repository and do a commit (See [Adding and Removing Files — page 17](#)).

---

## Working with CVS in SNIFF+

---

There are two different ways to use CVS together with SNIFF+. One way is to use the well-known file-based SNIFF+ version control commands and the other one is to use the CVS module-based commands:

- **File-based** commands behave exactly the same way as known from RCS, so they make it easy for you to migrate; you can use the SNIFF+ concepts of Shared Working Environments, Default Version Control Configurations and Branching as known from RCS. In addition to the SNIFF+ typical commands, there are some more CVS file commands in the **CVS** custom menu for providing additional features.

File-based commands have well-defined outcome under all circumstances, but they are slower (with remote repositories, sometimes *much* slower) than module based commands.

- **Module-based** commands are CVS-specific and provide enhanced performance. However, due to the nature of cvs they interact poorly with the SNIFF+ concepts of Default Version Control Configurations and Shared Working Environments. They *can* be used in such circumstances as well, but you should know exactly what you are doing.

Generally, we recommend using the file-based commands with local repositories and for small operations (like checking in two or three files). You can extend the file-based features with module-based commands for enhanced performance, especially when using remote repositories and when not using a shared source.

Thanks to the fast cvs update mechanism, it is particularly useful to not use a shared source on the Windows platform where symbolic links are not available for sharing source and object files.

More differences, advantages and disadvantages of using a shared source or not will be explained in the following sections.

### Working with a Shared Source (file-based approach)

The main advantage is the sharing of source and objects files. Just a small part of all source and object files need to reside in the Private Working Environments.

After the creation of the Private Working Environment the whole project is visible for SNIFF+'s browsing and code comprehension tools. Since the Shared Working Environments are read only, it is necessary to check out modules and files to be modified. The checked out files will then reside in the private workspace, all other files will be shared from the Shared Source Working Environments.

Objects files can be shared by using a Shared Object Working Environment (SOWE).

| Advantages   | Disadvantages   |
|--|---|
| <ul style="list-style-type: none"> <li>■ Sharing of source files by using SSWE(s).</li> <li>■ Complete browsing information for the whole project.</li> <li>■ Sharing of object files by using SOWE(s).</li> <li>■ Less compilation time.</li> <li>■ Modules can be checked out from SNIFF+'s GUI.</li> <li>■ Automatic creation of the private WE by SNIFF+.</li> </ul> | <ul style="list-style-type: none"> <li>■ Cvs update should be used for single files only, because it makes all files local; use SNIFF+ update instead, which is slower</li> <li>■ Higher network bandwidth requirements due to shared source</li> </ul> |

The typical development cycle when using a shared source is like this:

- Open the project in a PWE. All files are shown from the Shared Source, no files are local. The Shared Source must be CVS-Controlled.
- Browse the files. If a file needs to be edited, execute **File > Check out...** (with concurrent lock) to get a local copy of the file to be edited.
- Edit the file until you are satisfied; execute **CVS > Update File** to incorporate any changes made by other developers. Resolve any Conflicts.
- Execute **File > Check in File** to check in single files or **CVS Modules > Commit Module** to commit all modifications made in a single step.
- Execute **File > Update File** or **Project > Update Workspace** to clean any local copies of files that have already been updated in the Shared Source.

For other special commands, see the following sections.

## Working without a Shared Source (module-based approach)

When using CVS, it is not necessary to use shared workspaces to share source and object files. The sharing of source files can also be done by just checking out the necessary modules into the Private Working Environment. The main advantage is that it is not necessary to create and maintain (update) Shared Working Environments; instead, cvs update can be used which requires less network bandwidth.

The main disadvantage is that object files can not be shared by CVS itself, therefore no sharing of object files will be possible. Furthermore just source files of checked out modules are visible within the SNIFF+ browsers - no browsing information about non checked out

source files is available. Therefore it is necessary to check out all files which are of interest for the development work or necessary for building a target. This may lead to a large number of checked out modules in the Private Working Environments.

| Advantages   | Disadvantages  |
|--|--|
| <ul style="list-style-type: none"> <li>■ No Shared Working Environment required.</li> <li>■ No update of shared WE's is necessary</li> <li>■ Less administrative work (no WE administrator required).</li> <li>■ Less network bandwidth requirements.</li> </ul> | <ul style="list-style-type: none"> <li>■ No sharing of object files.</li> <li>■ Increased compilation time.</li> <li>■ Only checked out modules are visible in SNIFF+</li> <li>■ Decreased Browsing Information.</li> <li>■ Check-Out Modules must be done from the Command Line.</li> </ul> |

The typical development cycle when working without a shared source is like this:

- Check out all files from CVS into your PWE. This can be done from the command line, or by using the **CVS Modules > Check out module into...** menu in the project editor. opening. Depending on your preferred way of working, the files can either be read-only (CVSREAD=1) or writable.
- Browse the files. For any modifications, load the file into the editor and execute **CVS > Edit file** to make it writable. You can watch other files to get informed when other users edit files by setting **CVS > Watch file(s)** or by executing **CVS > Show editors of file(s)**.
- Execute **CVS Modules > Update Module** from your toplevel module to perform a  `cvs update` on your whole workspace and get the newest versions from the Repository.
- Use the **CVS Modules > Commit Module** custom menu to commit your modifications.

## Adding and Removing Files

Since both SNIFF+ and CVS keep record of what files belong to a specific project or a specific configuration, the SNIFF+ **Project > Add/Remove Files to/from *projectname...* dialog** should *NOT* be used for removing files. Instead, you should use the corresponding CVS custom menu entry **CVS > Remove file(s)**.

Note that your Project Description File must be writable in order for this command to work.

## Changing Project Attributes

It is important that no CVS update conflicts occur in the SNIFF+ project description files, because the CVS conflict lines would confuse the SNIFF+ project loading mechanism. Therefore, Project Description files should be made writable only for a short time and committed after any modifications. Moreover, CVS update should be executed on the SNIFF+ project description files with great care. Use **File > Update** instead to make sure no conflicts occur.

---

## Setting tags (freezing configurations)

You should set tags by using the `cvs` custom menus; this is much faster than the SNIFF+ Configuration Manager.

To freeze a current checked-out configuration, check out or update all files to the configuration you would like to freeze; then, select all files and execute **CVS > tag selected file(s)** to tag the current checked-out version.

To freeze the overall HEAD, the HEAD of a branch, rename or delete configurations, you should use the **CVS Modules > define tag** and **CVS Modules > delete tag** custom menu entries. They work directly on the repository without taking your current checked-out versions into account and are therefore even faster.

## Working with Branches

To create a new branch, you can either use the SNIFF+ file-based commands or the CVS module-based commands.

The file-based commands are simple to handle and work as known from RCS; but they are slow, since for every File/Check in, SNIFF+ needs to execute up to 4 `cvs` commands (`cvs tag` for creating a branch tag, `cvs commit` for committing, `cvs tag` for setting a change set name and `cvs edit` for setting a lock).

For module-based branching, you first need to define a branch tag, and then execute the **CVS Modules > Commit Module into branch** custom menu. The branch sticky tag will be set for all files of the module, but only changed files will be really checked in. All further modifications will be committed to the same branch.

To revert your branch to the main trunk, execute **CVS Modules > Update Module Tree** and update to the default branch (`cvs update -A`).

## Other Actions

For all other actions, see the [Command Reference — page 23](#).

# Advanced topics

## Customizing the Adaptor

### SiteMenus and UserMenus

The CVS custom menus are currently rather large; however, it is easily possible to adapt these menus to your specific needs.

One customization that is particularly recommended is removing the whole **CVS Admin** menu for developers and enable it for the CVS Administrator only. This can easily be accomplished by first copying the `SiteMenus.sniff` file to the Administrator's `UserMenus.sniff` file and then commenting out all entries from the **CVS Admin** menu.

### CVSPrefs.py

More customizations can be done by editing the file

`$SNIFF_DIR/lib/python/Sniff/CVSPrefs.py`. This file defines constants that customize the behavior of the CVS Adaptor in many ways, and allows to define Access Control Lists for Administrative commands. Below is a list of the most often requested configuration possibilities in `CVSPrefs.py`, together with their default values as shipped:

| CVSPrefs Entry            | Meaning   |
|---------------------------|---|
| LockersShowExclusive=1    | Show exclusive lockers in the lockers column. This makes the "lockers" display very slow with remote repositories. Set this to 0 to switch off displaying exclusive locker information. Note: you can still toggle exclusive lockers display through the CVS custom menu even if the feature is switched off by default here. |
| AllowExclusiveLocking=1   | Allow to do exclusive locking. If switched off here, the "Check out / Exclusive Lock" and "Lock" menu entries will print an error message if called.  |
| AllowMergeInSniffUpdate=1 | Allow updating private writable files with cvs update during the SNIFF+ Synchronize File / Synchronize Project command (note that Project Description Files will still be protected against update conflicts)   |
| DoCVSEdit=1               | Run cvs edit for checkout with concurrent or exclusive lock. Set this flag to 0 to just run a local chmod (set writable) command instead. This might be useful if you have a dialup connection only and you want to avoid going on-line for every Edit.   |

|                              |  |
|------------------------------|--|
| ChangeRemoteRepUser=1        | For RWE Settings like :pserver:user@host, exchange the user@ setting with the actual current user ID. Set this to 0 if you are using \$CVSROOT environment variable for your RWE Directory setting and your CVS login ID is different than your local login ID.  |
| AllowBranchOpsInCM=0         | The "Create Branch", "Delete Branch" and "Rename Branch" operations in the SNIFF+ Config Manager are critical, because they introduce irreversible changes to the Repository. In particular, "Rename Branch" only works correctly only with very few CVS versions. Set this setting to 1 if you still want to use these commands.  |
| EditRemoveBaseForPDF=0       | Set this to 1 if you get confused by seeing superfluous SNIFF+ Project description files in the "Open Project" Dialog which are left over in the CVS/Base directory.   |
| RequireComment = 0           | If set to 1, check in operations will only be allowed if a comment is entered. Note: this will not be active for the "commit" dialog!  |
| RequireChangeset = 0         | If set to 1, check in operations will only be allowed if a SNIFF+ Changeset is entered. Note: this will not be active for the "commit" dialog!   |
| DiffOptions = "-b"           | If files differ by blank characters (spaces, tabs) only, don't show this as difference in the Diff/Merge tool. Set this to "" (empty string) to see blank's differences as well.   |
| RetainLockMode=gRetainExIfEx | This entry controls how pressing the "Retain Lock" button in the Check In dialog is interpreted. The default setting will do a concurrent lock if the file was concurrently locked before, or an exclusive lock if the file was exclusively locked before. For other settings, see the CVSPrefs.py file.   |
| CheckoutTimestampFix=0       | It was observed with some versions of CVS, that when a file is checked out for the first time (for instance, after checking out and reloading a project where this file was newly added), the timestamp for this file is not set to the time of the checkout but to the time when it was created. When a Shared Object Working Environment is used, the wrong timestamp may lead to the file not being compiled although it should. If this switch is set to 1, SNIFF+ will work around this problem by touching the file if it was not already set to the timestamp of the checkout by CVS. |
| DoFixDollar=0                | Allow to use filenames with a Dollar Sign in their name. This fix is not thoroughly tested and thus OFF by default.  |

|                   |  |
|-------------------|--|
| AdminACL= [ ]     | <p>This Python List may be set to a list of Strings which represent User IDs. If set, then only the users listed here are allowed to perform the following actions:</p> <ul style="list-style-type: none"> <li>■ "Delete Version"</li> <li>■ "Delete Symbol" in the CM</li> <li>■ "Rename Symbol" in the CM</li> <li>■ "Delete Branch name" in the CM if also AllowBranchOpsInCM is set on</li> <li>■ "Set Symbol for branch" in the CM if also AllowBranchOpsInCM is set on</li> <li>■ "Rename Branch" in the CM if also AllowBranchOpsInCM is set on</li> </ul> <p>If empty (default setting), everyone may perform these actions.</p> |
| BreakLockACL= [ ] | <p>This Python List may be set to a list of Strings which represent User IDs. If set, then only the users listed here are allowed to break locks. If empty (default), everyone may break locks.</p>  |
| BranchACL= { }    | <p>This Python Dictionary may be set to a list of branch names each associated with a list of User IDs. If set, then only the members of the list may check in to the corresponding branch. This can be used, for instance, to enable branching policies. Branch names not mentioned may be checked-in to by everyone.</p>   |
| TagACL= { }       | <p>This Python Dictionary may be set to a list of tag names each associated with a list of User IDs. If set, then only the members of the list may set or move the corresponding tag. Names not mentioned may be set or moved to by everyone.</p>  |

For more detailed information, please contact the WindRiver support team at [sniff-support@takefive.co.at](mailto:sniff-support@takefive.co.at)

## Migrating from other VCS tools to CVS

Since CVS uses RCS for its internal files, no history or configuration information will be lost when you migrate from RCS to CVS. For in-depth information on migrating to CVS (also from other revision control systems), please refer to the [CVS Manual \[4\]](#).

For the RCS to CVS migration with SNIFF+ specifically, there is a shell script `rcs2cvs` which can be obtained through the SNIFF+ Knowledge Base at

<http://www.takefive.co.at/faq/kbMigrateRcsRepositoryToCVS.html>

---

## CVS Remote Repositories and Multi-Site Development

The `cvs login` command for remote repositories using the "pserver" access command is not implemented in the SNIFF+ CVS Adaptor.

Please do `cvs login` from the commandline; your CVS password will be saved in your home directory's `.cvspass` file. SNIFF+ requires this password caching because you don't have a chance to enter passwords during the custom menu and adaptor commands.

## Windows to UNIX Cross-Platform Development with CVS

There are two problems when doing Windows to UNIX Cross Platform development:

1. When `cvs` is run on Windows, it will add a Carriage Return (CR) Character at the end of the line of every file it creates: both CVS internal files and files managed by CVS. Most UNIX Compilers don't work with the resulting CRLF line ending scheme.
2. The `CVSROOT` scheme required by Windows local repositories (`:local:d:\the\repository`) can be saved in the `CVS/Root` file with the sources. UNIX Versions of CVS don't understand this special form of `CVSROOT` and produce an error.

These problems can be solved by using a version of CVS on Windows that has been compiled with the "Cygwin" kit. By compiling CVS with this kit, it will behave more UNIX-like. For more information, see the SNIFF+ Knowledge Base at

<http://www.takefive.co.at/faq/kbUsingCvsInCrossSetup.html>

## Pitfalls when using SNIFF+ with CVS

Both SNIFF+ and CVS keep a database of your software structure and what files belong to a specific version; similarly, SNIFF+ and CVS use different methods to handle working environments with branches: while CVS uses "sticky tags" to make sure branch versions are checked in into the correct branches again, SNIFF+ uses working environments with "Default Configurations" for that purpose.

It is important that SNIFF+ and CVS information are kept in sync: particularly when Shared Working Environments are checked out with CVS sticky tags it is important to make sure that all Private Working Environments based on these use the same default configuration.

## Known Limitations

- SNIFF+ project description hierarchies and CVS Modules files cannot automatically be converted into each other.

# Command Reference

---

## CVS Admin Menu (Project Editor)

The CVS Admin menu contains all commands for administering the CVS Repository and internal files, and for importing new files into CVS.

### Init CVS Repository

Initializes the repository. This menu will invoke the `cvs init` command, which will create all CVS administration files under `$CVSROOT/CVSROOT`.

### Add SNIFF+ cvsignore entries...

Creates a new ignore file (see the [CVS Manual \[4\]](#) for more info about cvsignore files). You will be prompted to insert the name patterns that should be ignored while running update, import and release. The following name patterns are recommended to be ignored when working with SNIFF+:

```
.sniffdir
.ProjectCache
.Snifflast*
*%
```

### Add SNIFF+ cvs wrappers...

This menu item creates a new cvs wrappers file, which may be used to make file types known to CVS as being binary (see the [CVS Manual \[4\]](#) for more info about the cvs wrappers file). You will be prompted to insert the name patterns that should be treated as binary files. The following name patterns are recommended to treat as binary with SNIFF+, and will be preselected in the dialog:

```
*.shared -m 'COPY'
*.proj   -m 'COPY'
*.gif    -k 'b'
*.png    -k 'b'
*.vcl    -k 'b'
```

### Edit cvsignore file

Invokes a little editor window in SNIFF+ which displays the current cvsignore file. The contents can be modified, and upon pressing the “OK” button the master cvsignore file will be automatically updated by committing your changes to the CVS Repository.

---

### Edit cvswrappers file...

Invokes a little editor window in SNIFF+ which displays the current cvswrappers file. The contents can be modified, and upon pressing the “OK” button the master cvsignore file will be automatically updated by committing your changes to the CVS Repository.

### Edit modules file...

Invokes a little editor window in SNIFF+ which displays the current CVS modules file. The contents can be modified to add or remove CVS modules, or change their properties, and upon pressing the “OK” button the master modules file will be automatically updated by committing your changes to the CVS Repository.

### Import sources from current project...

Invokes the import command from the directory of the currently selected file. The import command is used to check-in the sources for the first time. You will be prompted to enter the correct parameters:

**-m** : Log information

**directory** : Directory name (relative to \$CVSROOT) into which the files will be imported.

**tag1** : Vendor Tag – Used to specify a symbolic name for a branch - e.g., takefive.

**tag2** : Release Tag – Used to specify a symbolic name for a release - e.g., start.

### Compare Directories...

Compares the directory rooted at the currently selected file's directory with a second directory you select. SNIFF+ and CVS administrative information is masked out before comparison. This allows you to check for successful CVS import.

### Overwrite Tag on selected file(s)...

This entry allows to move labels or changesets from any other version to the version which is currently in the workspace. It can be used if the standard “**CVS > Tag selected file(s)**” command returns an error because of an already-existing Label name.

## CVS Modules Menu (Project Editor)

The CVS Modules Menu contains all commands for working with multiple files. The current module is always determined by the name of the SNIFF+ project description file to which the currently selected file belongs. You should make sure that for all module related commands, only ONE FILE is selected before executing the command!

### Check out module into...

This command will be used when the module structure cannot be mapped to any project. When you launch this command a directory dialog will appear to select the directory from where you want to perform the check out. You will then be prompted (in the SNIFF+ shell) to enter the name of the module you want to check out.

## Schedule file(s) for addition

Performs a `cvs add` on the files selected. Note that you will need to commit your module to make the addition permanent.

## Show Workspace update status

Changes to the Workspace's toplevel directory and runs the “`cvs -n -q update`” command in the SNIFF+ shell. This command shows what files would be updated if a “`cvs update`” command were run on the Workspace. It will display all files which are not up-to-date without actually changing anything. Therefore, it is a nice feature to check what has been going on with the project.

**Note:** for this command to work correctly in your workspace root directory, you will need to have the “`TopLevelAdmin=yes`” in your `CVSROOT/config` configuration file.

## Update entire Workspace w/options...

Performs a `cvs update` for your entire workspace. Refer to the [CVS Manual \[4\]](#) for a description of what the options mean.

**Note:** for this command to work correctly in your workspace root directory, you will need to have the “`TopLevelAdmin=yes`” in your `CVSROOT/config` configuration file.

## Commit entire Workspace...

Changes directory to your Workspace Root Directory and performs a `cvs commit` operation. A question dialog is opened to allow entering a comment for the commit operation. We recommend to run “**Show update status**” (see above) before committing, to check what files will be committed.

**Note:** for this command to work correctly in your workspace root directory, you will need to have the “`TopLevelAdmin=yes`” in your `CVSROOT/config` configuration file.

## Show update status of current module

Runs the “`cvs -n -q update`” command in the SNIFF+ shell, in the directory / directories in which the currently selected files reside. This command shows what files of the current module(s) would be updated if a “`cvs update`” command were run. This command will display all files which are not up-to-date without actually changing anything in your working directory. It can be used to check what has been going on with the project.

## Update current module w/options...

Performs a `cvs update` for the current module. Refer to the [CVS Manual \[4\]](#) for a description of what the options mean.

## Commit files from current module...

Performs a commit in the directory of the selected file. A question dialog is opened to allow entering a comment for the commit operation. We recommend to run “**Show update status**” (see above) before committing, to check what files will be committed.

---

## Recursively commit current module...

Same as above, but recursively descend into subdirectories

## Commit files from module into branch...

Commits files from current module into specified branch.

## Define tag for module...

You can use this command to assign symbolic tags to the sources of the current module. It prompts for the tag name and then runs `cvs rtag <tag> <module>`. Note that you need to have defined the current module in the `cvs modules` file for this command to work. Also note that the `rtag` command will always freeze the current HEAD or HEAD OF BRANCH configuration. For freezing checked-out versions, you need to use **cvs > tag selected file(s)**.

## Define branch tag for module...

Same as above, but make the tag a branch tag. Again note that you need to use **cvs > tag selected file(s)** to tag checked-out versions.

## Delete tag for module...

Delete the tag, also using `cvs rtag`.

# CVS Menu (Project Editor)

The Project editor's CVS menu contains all CVS commands that go beyond the features of the File menu for working with individual files or complete working environments.

## Status of selected file(s)

Performs a `cvs status -l -v <filename>`. This command displays status information on checked out files.

## Show editors of file(s)

Shows who is currently editing the selected file(s).

## Uncheckout selected file(s)

This command performs a "cvs unedit" operation to revert a local file to the version it had before "edit" or "checkout" was issued. If a Shared Source is used and, because of the unedit, the local file is now the same as the SSWE, the local file is also removed.

## Remove selected file(s)

Physically removes the selected files and removes the entries from the SNIFF+ project description file and the CVS Repository. For this command to succeed, the SNIFF+ Project description file which contains the selected file(s) must be writable!

**Note:** SNIFF+ will open a dialog and ask you for every file that you select to remove. You should therefore not use this command for removing very many files at once. For removing many files, or removing entire subprojects, you should:

1. Check out / no lock the files and projects to your local workspace
2. Use the SNIFF+ Add / Remove file(s) and / or Remove Subproject command(s) to remove the file(s) from SNIFF+
3. Use the **Project > Check Obsolete Files** dialog to physically remove the files
4. Use the “**cv**s **remove**” and “**cv**s **commit**” commands on the commandline to notify CVS of removed files and commit your changes.

## Update selected file(s) w/options...

Does a `cv`s `update` to merge changes made by other developers since the last checkout or update. The most important properties of `cv`s `update` compared to SNIFF+'s File/Update are as follows:

- `cv`s `update` looks at writable files, too. The latest version is applied to a checked-out file by applying a patch. This is more efficient than SNIFF+ Update, but may lead to conflicts that need to be resolved.
- `cv`s `update` will never delete a file to show the SSWE version again.

You may give options to the `cv`s `update` command, like the “-j” option for automatically merging other developer’s work into your local version. See the [CVS Manual \[4\]](#) for more information on options to the `cv`s `update` command.

## Tag selected file(s)...

Sets a symbolic name on the selected file's checked-out version. This is the preferred method for “freezing” a configuration which is currently in the local Working Environment. Note: when very many files (more than 1000) are selected, this command can cause troubles. If you are not using modules, we recommend to use the module-based commandline “`cv`s `tag`” command in that case. If using Shared Workspaces, we recommend splitting the operation into multiple requests of less than 1000 files each.

## Watch selected file(s)

Sets a watch on the selected file: any subsequent `cv`s `edit` commands on the selected file(s) will now notify the watcher by eMail.

## Offline Edit selected file(s)

Makes the selected files (which are already checked out) writable. This command does **not** notify CVS about the file(s) being edited and thus will not need to open a remote connection to your CVS Server. It is meant for offline using CVS.

---

## Online Edit selected file(s)

Makes the selected files (which are already checked out) writable and notifies CVS that they are going to be edited. This can also be used to re-synchronize offline edit actions that were done before: use the project editor's **writable** filter to show only writable files in your Workspace; these are the ones that your **offline edited** before. You can now select from all these files and **online edit** them (CVS will not choke if you "online edit" files more than once).

## Unedit selected file(s)

Makes the selected files read-only and notifies CVS that they are not edited any more. This command is similar to the **Uncheckout** command (see above), but it will not remove any local file even if it is the same as in the SSWE.

## Toggle exclusive Lockers

Switches on or off the display of who has an exclusive lock in the Project Editor's lockers column. Note: Showing exclusive lockers makes the lockers display **much** slower than showing the CVS Editors only, especially when a CVS Remote Repository is used. We therefore recommend to use this feature only in local area networks.

## Toggle CVS Debugging

Switches on or off the display of detailed CVS specific debug information into the SNIFF+ Log Tool. Note: when using this option, you should set the "**Open Log Window on Output**" option in the SNIFF+ Preferences / Tools node to "off" !

## Toggle vcs Debugging

Switches on or off the display of general SNIFF+ Version Control operation debug information into the SNIFF+ Log Tool. Having this switch "on" will print all the CVS commandline commands as they are executed. Note: when using this option, you should set the "**Open Log Window on Output**" option in the SNIFF+ Preferences / Tools node to "off" !

## Reload CVS Adaptor

Reloads all Python Modules of the CVS Adaptor. May be used when you personally modify CVS Adaptor files, like preferences in the `CVSPrefs.py` file. This command will also print the current CVS Adaptor version.

## CVS Menu (Source Editor)

The editor's CVS menu contains some of the CVS commands from the Project Editor's CVS menu plus one text based command:

### **cvs status <filename>**

Show the CVS status of the currently edited file: this is the version from which editing started, the currently active branch tag and the repository file that is involved.

## Show editors of <filename>

Show who is currently registered by CVS as editing the current file.

## cvs update <filename>

Performs a cvs update to incorporate the latest changes into the current file by patching it. Note that this can introduce conflicts that you need to resolve. This command will not allow you to enter options for the cvs update command (if you need options, use the Project Editor's update command instead).

## Edit <filename>

Makes the currently edited file writable and notifies CVS that it is going to be edited.

## Unedit <filename>

Makes the currently edited file read-only and notifies CVS that no more editing is going to be done. Note that any changes you made will be lost after the next SNIFF+ Update File since read-only files will be brought to the revision requested by update file without any patching. So in case you want to keep your changes, you should keep your files writable (=edited) or commit them to the repository.

## Find next conflict (C-t c)

Finds the next string "<<<<" marking a CVS conflict after an update. The same functionality is available on the Key Sequence <CTRL>-t c.

# File Menu (Project Editor and Source Editor)

The standard SNIFF+ CMVC functionality for individual files is available in the File menu. Most functions are available for CVS although many of them can be accomplished more efficiently by using the CVS custom menus. Especially note the small differences in meaning between the File Menu's functions and the CVS custom menu's functions.

## Check Out...

Check out a selected file from the Repository. Note that your Current Project must have been initialized ("Initialize current Project/Module") for this command to work. You may select a version to check out and a locking option:

**No Lock** - the file is checked out read-only.

**Concurrent Lock** - the file is checked out and `cvs edit` is executed to make the file writable. This should be the default function for you to use!

**Exclusive Lock** - the file is checked out and exclusively locked with `cvs admin -l`. Note that exclusive locking is NOT recommended with CVS!

---

The **File > Check out...** internally uses a `cvsv update` command but deletes the file before sending the command. Therefore no `cvsv merging` can occur, and the whole file is transferred. Therefore, especially when using remote repositories, a `cvsv update` command is more efficient for checking out files.

If you check out an older version than head, a CVS sticky tag for the corresponding version will be set (see the CVS manual on sticky tags).

## Check In...

Checks in the selected file(s) to the repository, makes them read-only and notifies CVS that editing has been finished. There are some things to note:

- Before committing, CVS checks the files to be committed: if conflicts are found, a message is printed and the corresponding file is not checked in. Also, CVS will not check in files that have not been changed.
- When multiple files are selected, this command will perform an individual `cvsv commit` for every file. Therefore, when committing multiple files, the **cvsv commit** custom menu commands are to be preferred, especially if CVS is set up to notify an administrator of every commit that is made. The **File > Check In...** command allows you to specify `SNiFF+ Changesets`. These are implemented as CVS tags. Note that when check-in is not possible due to a conflict, the changeset will also not be set for the corresponding file.

Because of these reasons, we recommend to use the **File > Check In...** command only if few files are affected, if you want to assign a changeset and if no conflicts are assumed. Whenever possible, you should use the **CVS Module > Commit** or **CVS > Commit entire Working Environment** custom menu commands instead and enter changesets with the **CVS > Tag files...** custom menu command.

## Lock...

Sets an exclusive lock on the given file. Note that this is not recommended because exclusively locked files can still be edited by CVS users but can not be committed!

## Unlock...

Removes an exclusive lock. If the file currently selected is currently being locked by a different user as the one calling the action, a "**Break Lock**" is issued. This Break Lock functionality can be restricted to Administrators only in the `CVSPrefs.py` file (see [Advanced topics — page 19](#)). Note that Breaking a Lock is not supported in all versions of CVS.

## Delete Version...

This command is not recommended for CVS, because checked-out files of the version to be deleted can still exist. In the `CVSPrefs.py` file there exists a flag that allows to disable this command or reserve it for Administrators only.

## Replace Description

Change the "file description" of a selected file.

## Replace Comment...

Change the comment of a file version currently selected in the History Window.

## Synchronize File...

Performs a SNIFF+ Synchronize on the selected file. By default, this means running cvs update on the file (regardless if it is writable or read-only), then comparing it with the corresponding shared source file (if this exists) and if the contents is the same, deleting the local file so the shared one can be seen.

If the current file to be synchronized is a Project Description file, this command will also check whether the cvs update command introduced any conflicts, and if yes, discard the update and keep the old file version. In this case, you will need to manually merge the latest changes into your file version.

The default behavior of this command can be overridden in the CVSPrefs.py file to not touch writable files and more resemble the standard SNIFF+ behavior, which is as follows:

- In standard SNIFF+, Synchronize File... only looks at read-only files. Writable files are never changed. Files will always be checked out completely and never be patched, therefore conflicts cannot occur; changes in read-only files will be overwritten.

So if you switch on this behavior, the SNIFF+ Synchronize File and the Custom Menu's CVS Update functions complement each other.

Note that the SNIFF+ Menu Entry **Project > Synchronize Workspace** also does a SNIFF+ Synchronize File for all files of all currently selected projects, plus some additional optimizations.

## Show Differences...

Opens a dialog to select versions of files to compare in the Diff/Merge tool. If a version is selected in the History pane, this version will be selected by default; otherwise symbolic names (configurations) can be used to select file versions. If the local file is writable, merging will be enabled in the Diff/Merge tool.

Note that there is a flag in `CVSPrefs.py` which allows you to tree differences of blanks only as differences or not. (`diff -b` flag).

# Project Menu (Project Editor)

## Synchronize Checkmarked Projects

Runs "**Synchronize File**" (see above) for every file of every currently selected project. This command has some optimizations for running faster if there is direct filesystem-level access to the Repository. However, if a Remote Repository is used this command is slow and we recommend working with CVS Modules instead ("**Update current module**" command).

---

## Configuration Manager

While most functions of the SNIFF+ Configuration Manager work as usual, there are some differences due to the specific nature of CVS; particularly, you should not set any tags from the Configuration Manager but use the **CVS Modules** custom menu instead.

Also note that CVS branch tags will not be shown in the configuration manager if the corresponding branches are empty.

---

## Useful Links

---

**[1]** SNIFF+ FTP Server

<ftp://ftp.takefive.com/pub/SNIFF/integrations/cvs/>

**[2]** CVS Homepage

<http://www.sourceforge.com/CVS>

**[3]** CVS Bubbles

Pascal Molli

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

**[4]** CVS Manual

Per Cederqvist et al. Version Management with CVS for CVS 1.10. Signum Support AB, 1998. Online.

<http://www.fido.de/kama/cvs-de.html>

[http://www.loria.fr/~molli/cvs/doc/cvs\\_toc.html](http://www.loria.fr/~molli/cvs/doc/cvs_toc.html)

<http://www.loria.fr/~molli/cvs/doc/cvs.pdf>

**[5]** CVS tutorial

Jim Blandy

<http://www.cyclic.com/cvs/doc-blandy.html>

**[6]** CVS II: Parallelizing Software Development

Brian Berliner, Prisma, Inc. 1989.

<http://www.loria.fr/~molli/cvs/doc/cvs-paper.pdf>

**[7]** CVS Mailing List

[cvs-info@gnu.org](mailto:cvs-info@gnu.org)

Archived at

<http://www.egroups.com/list/cvs-info/>