HOWTO Use SoundForge CVS for Snitz Forums 2000

by Pierre Gorissen

HOWTO Use SoundForge CVS for Snitz Forums 2000

by Pierre Gorissen

This is a short HOWTO on how to use CVS on SourceForge. The example is made for the developers of Snitz Forums 2000, but should also be applicable for others using Windows and SourceForge.

Table of Contents

Chapter 1. Introduction

About this HOWTO

This is a short HOWTO on how to use CVS on SourceForge. The example is made for the developers of Snitz Forums 2000, but should also be applicable for others using Windows and SourceForge.

This is a first version. Though the info is correct, it is not completed yet.

Who should read this HOWTO

tbd....

Steps to take

If you are a SourceForge/Snitz developer

You're a SourceForge/Snitz developer if you have a developer account on the sf2k project at SourceForge. If you don't have an account there, you still can be a development team member at the Snitz support site at http://forum.snitz.com/, those two accounts are not linked.

First thing you need to do is to read the installation chapter. It explains what to install and how to install that.

After that, you need to create a new repository and checkout the existing code on the site. How you do that is explained in the basic functions chapter.

If you are NOT a SourceForge/Snitz developer

Even if you do not have a developer account, CVS can be usefull to you. You need a bit less software and setup though.

tbd.....

Chapter 2. Installation

Understanding how CVS works is easy once you've got all the software setup. You'll find a lot of HOWTO's concerning this, but most of them (also) have Linux in mind. And that can get very confusing from time to time. To get started you need to have three applications: a CVS client to upload and download the files, a SSH client for the secure connection and preferably something to check out the differences between versions of files.

Before you start

Before you start installing the needed applications, you have to make a few preparations:

Lookup the SourceForge data

If you are a developer using a project on SourceForge, then you need to know your SourceForge username, your password and the exact projectname of the project on SourceForge where you are a developer.

Create a root-directory

WinCVS needs a place to store the files you're going to check-out and edit. This is called the CVS-Root directory. You can use any (empty) directory on your pc for that. I used `d:\cvsroot\` as root-directory

- Create a root-directory, i.e. d:\cvsroot\

Installing WinCVS

Downloading

I downloaded WinCVS version 1.2 from the WinCVS [http://www.wincvs.org/download.html] website. You can choose a ftp mirrorsite that is close to your location for the 3,53 MB download or just use the Sourceforge downloadlink [http://sourceforge.net/project/showfiles.php?group_id=10072].

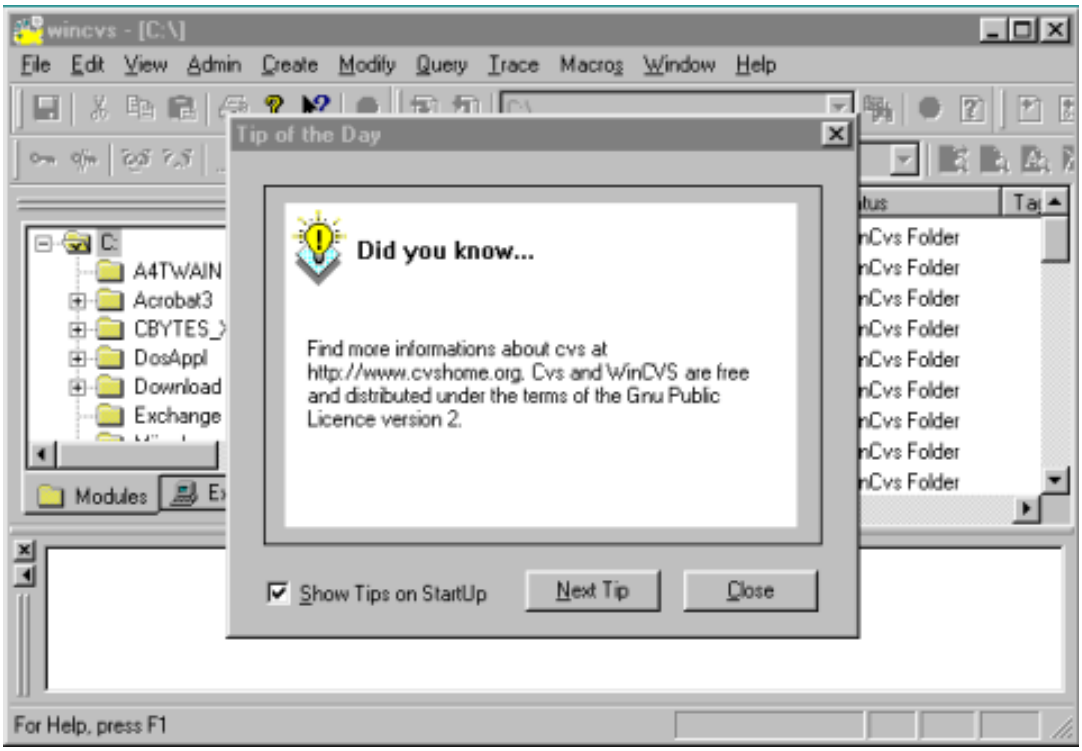
I stayed away from the 1.3 betas, since they are betas afterall, even though the version 1.2 is relatively old, from February 2001). Please let me know if I really am missing out on some great features there.

Setup

First setup of WinCVS is fairly easy. You'll have to unzip the package first before you can run setup.exe. During setup, just keep on clicking on the Next button and use the default setting provided by the setup program. After setup has finished you have to restart your computer.

After you restarted your computer you'll find WinCVS using Start > Programs > GNU > WinCvs 1.2 > WinCvs

- Start WinCVS to see if it installed ok



WinCVS

When you first run WinCVS it show a Startup tip. If you click Close it will open the WinCVS Preferences dialog box where you need to setup things like the CVSRoot. Leave these settings for now and exit WinCVS. You'll first have to setup the SSH client.

Installing SSH

Downloading

To install SSH we're going to use the SFSetup tool that is being provided by SourceForge. You can find the link to it on the SourceForge Setup page [http://sfsetup.sourceforge.net]. At the time of writing this, version 1.2 was the most current, so I used that one.

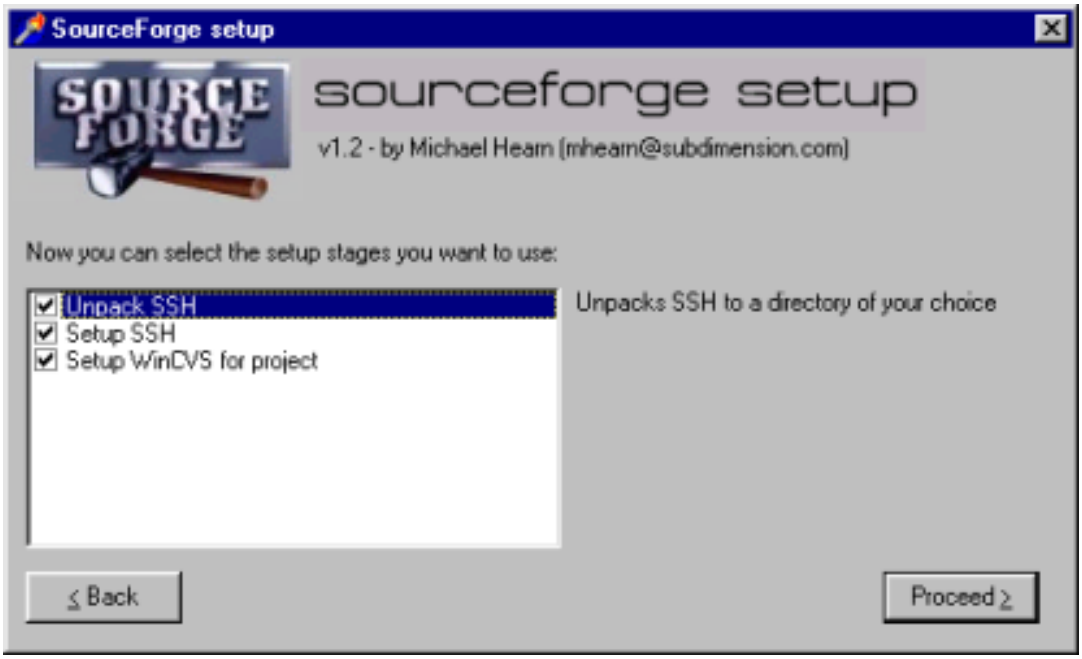
Setup

For the setup tool to work, you'll need to know you SourceForge username and the projectname.

- Download setup.zip
- Unpack it
- Read the README.TXT

- Run the program
- Click on Proceed> at the first screen

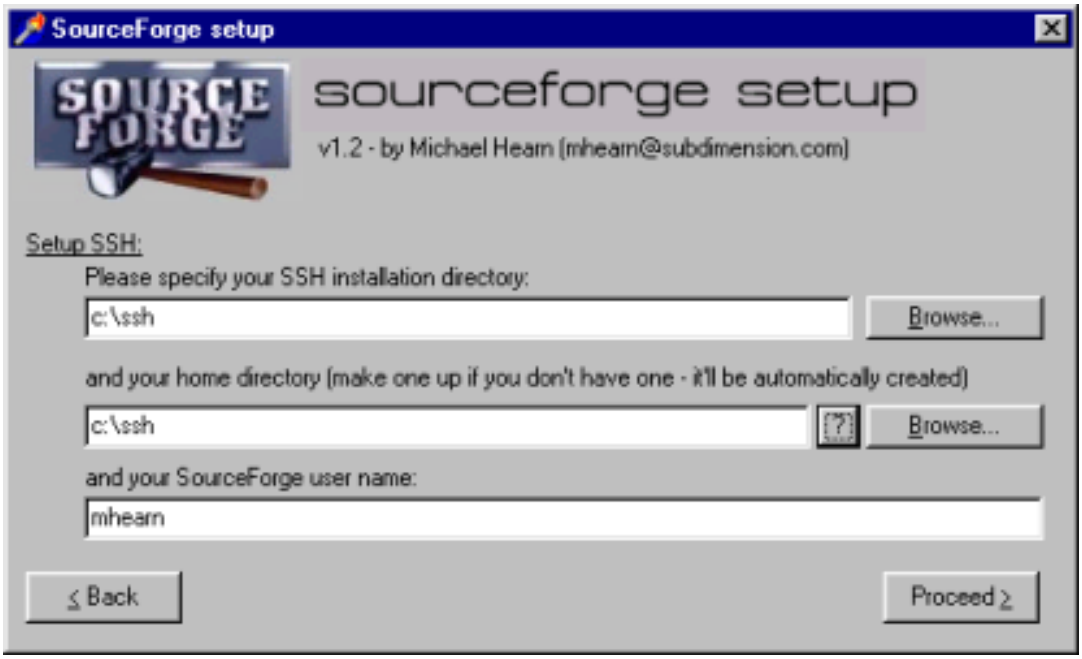
The second screen displays what setup stages you can have the tool perform for you:



If everything is ok, all three stages of the setup process are selected.

- Leave it that way and click on Proceed >
- Select a directory to install SSH in. The suggested directory c:\ssh will do, so just click on Proceed>

The next screen asks (again) for the SSH installation directory, your home directory and you username:



If you took my advise and didn't change the installation directory, you can leave both the SSH installation directory and the home directory unchanged here also. If you *did* change it, you'll have to change both entries here also.

- Change the default SourceForge user name to your own SourceForge username.
- Click on Proceed> again

The setuptool now asks for the name of the project you want the set up, you username (again) and the root-directory you created to store the files in.

- Enter the name of the project you want to set up
- Leave you username like it is (assuming it still is the same)
- Enter the root-directory you created before
- Click on Proceed>

SourceForge Setup is now ready for installation.

- Click on the Start button

After Setup has been completed you need to restart your computer.

- Restart your computer

BTW, if you like the setup tool, Michael Hearn [mailto:mhearn@subdimension.com], who created it

would appreciate your feedback on it!

Generate a SSH public key

Generating a SSH keyset allows you to use WinCVS without having to provide your password for each single action. You have to goto the DOS-prompt to do that.

- In Windows select: **Start** > **Run** > **Command** to get a DOS-prompt
- Goto the directory where you installed SSH
- Create a subfolder named .ssh (starting with a dot !)

> ### Note
>
> You can't create the .ssh folder from within Windows, you need to do it from the command promgt. The keyset gets stored in this folder.

- Run the ssh-keygen command: **ssh-keygen -C sf2k.sourceforge.net**
- When asked for the file to save the key to, just press RETURN
- When asked for the passphrase, and the confirmation of that passphrase do the same, so just press RETURN twice

The output for this process should look something like this:

```
C:\ssh>md .ssh
C:\ssh>ssh-keygen -C sf2k.sourceforge.net
Initializing random number generator...
Generating p: ..............................++ (distance 698)
Generating q: ............++ (distance 188)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key ($HOME/.ssh/identity):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in c:\SSH/.ssh/identity.
Your public key is: 1024336778312893402447455503881703356596749697910215
685211008784886799902672042642749883504006438543102951298927793981554718
625914874602186659188681922229932938894773383936458856561968019953329719
728682754549777047079646957589421388391610351569259036790206841022938806
sf2 k.sourceforge.net
Your public key has been saved in c:\SSH/.ssh/identity.pub
C:\ssh>
```

Your C:\ssh\.ssh directory should now hold at least the files identity, identity.pub and random_seed

Configure Sourceforge.net

Next thing to do is to tell the server at Sourceforge.net what you public key is.

- Logon to Sourceforge.net
- Goto your Account maintenance page
- Scroll down to the "Shell account information" part of the page
- Select to edit you SSH key

You now get a page with a textbox. Here you can upload you *public* key.

- Open you identity.pub file in a texteditor
- Select the entire public key
- Copy and paste the key into the textbox on SourceForge
- Click on update (two times)

It takes about 6 hours before the key is operational. So until then, it won't work yet. Leaves us some time to configure WinCVS for SSH.

Configure WinCVS/SSH

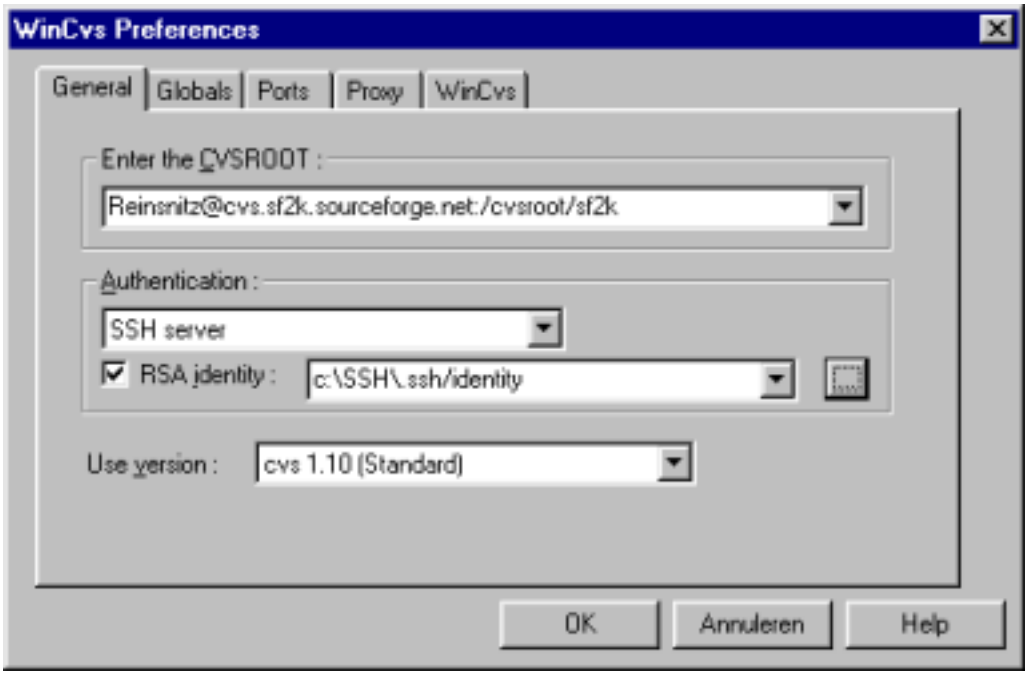
WinCVS needs to know where to find the public key you just generated and what it needs to do to logon. If SFSetup has done its work, most should allready have been added, but we'll check it to be sure.

- Start WinCVS
- Select Admin > Preferences
- The tab General should be selected by default
- The CVSROOT should allready have been added by the SFSetup tool.

  The format for the cvsroot is:
  <Username>@cvs.<Projectname>.Sourceforge.net:/cvsroot/<Projectname>
- Authentication should be set to SSH server

  The checkbox in front of RSA identity should be checked and the keyfile textbox should point to you identity file
- Use version should point to cvs 1.10(Standard)
- Click the OK button to apply the setting

The settings should look like this:



Want to know more about SSH?

I'm not going into the details about SSH in this HOWTO, but if you want to know more, take a look at this documentation page [http://sourceforge.net/docman/display_doc.php?docid=761&group_id=1] at SourceForge.

Installing CsDiff

Downloading

CsDiff is used as an external program to display the difference between different versions of a file. You can use it to find the changes that were made. There are a couple of (free) programs you can use, I used CsDiff version 2.5 [http://www.componentsoftware.com/csdiff/intro.htm]. It is a free tool and it worked for me. You can download it from the ComponentSoftware website [http://www.componentsoftware.com/csdiff/dlcsdiff.htm].

- Download CsDiff from the ComponentSoftware website [http://www.componentsoftware.com/csdiff/dlcsdiff.htm]

Setup

No real setup needed, just unzip the files to a directory.

Configure WinCVS/CsDiff

You need to tell WinCVS were it can find CsDiff.

- Start WinCVS
- Select Admin > Preferences
- Select tab WinCvs
- Check the box in front of "external diff program"
- Click on the browse button and select the CsDiff program
- Click on the OK button to apply the setting

You've now configured CsDiff

Final steps

The software is now ready to be used. Next thing to do is to create a new repository and checkout the code that allready is on the server.

How you can do that is explained in the chapter about Basic Functions.

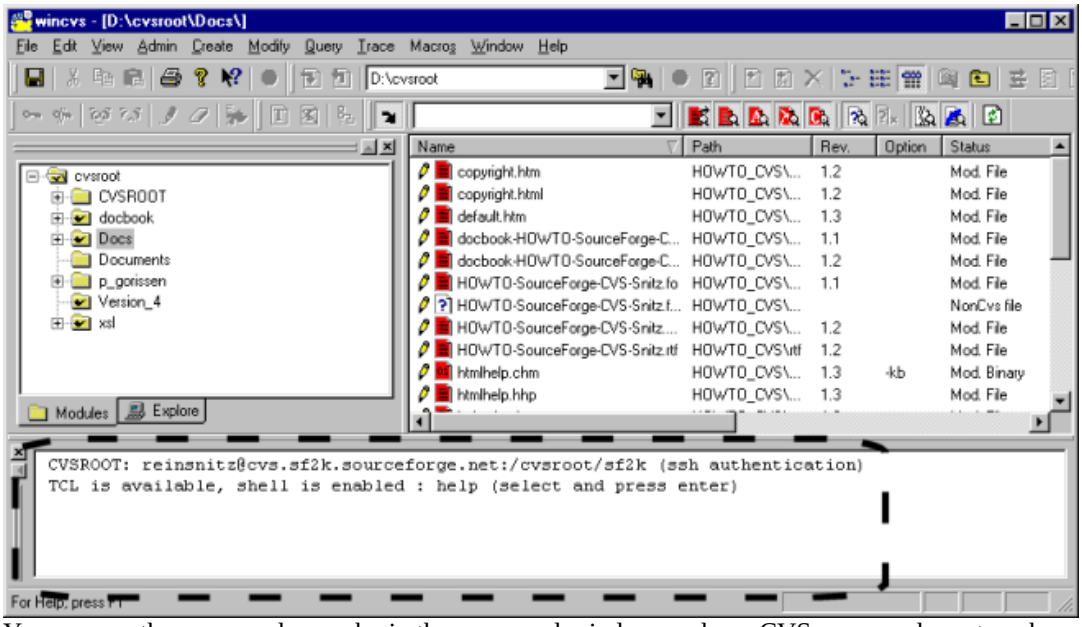
Chapter 3. Basic Functions

More detailed information

For more indepth information about using CVS take a look here a this great manual about CVS [http://www.loria.fr/~molli/cvs/doc/cvs_toc.html]. There is another great resource dedicated to CVS [http://www.devguy.com/fp/cfgmgmt/cvs/]on the Devguy's website, and of course on the WinCvs [http://www.wincvs.org/doc.html] site itself.

Two other sites that might be interesting: Info about the CVS-Concurrent Version System [http://www.opensource.apple.com/tools/cvs/cederquist/cvs_toc.html] and CVS Best practices [http://www.linuxdoc.org/REF/CVS-BestPractices/html/index.html]

Most information you'll find isn't written for WinCVS, but for command line CVS clients. That is no problem since WinCVS also has a command line window.



You can see the command come by in the command-window, and any CVS command you type here gets executed also.

Setting up a new repository

You only have to setup a new repository if you are starting with a fresh project and nobody has uploaded files there before. With the Snitz Forums 2000 project that is not the case, so there is no need to do that here. If you want to find out how to start a project from scratch including setting up a new repository see the short tutorial [http://www.loria.fr/~molli/cvs/doc/cvs_3.html#SEC37] .

Checking out an existing module

If you want to check out an existing module for the first time. You have to know what the name of that module is.

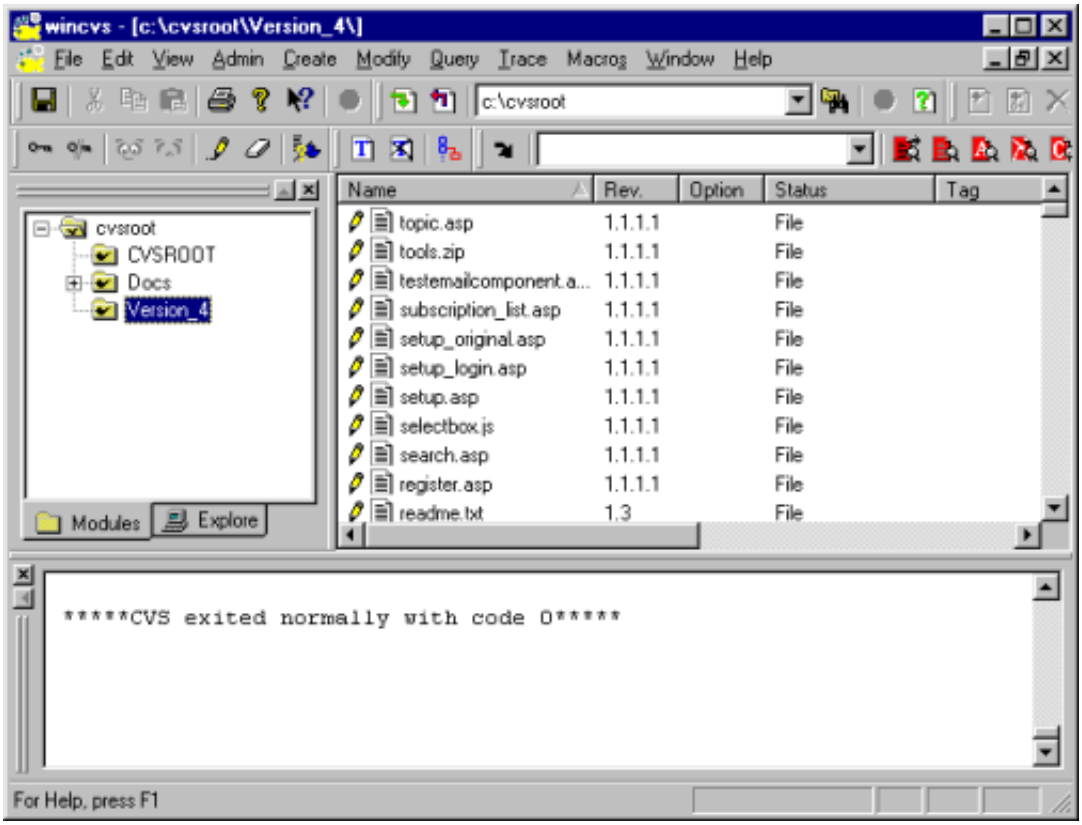
Tip

> I found that by using **./** as name for the module it checked out all existing modules on the server. You don't want to do that is you on the other end of a plain phone-line and the repository on the server is huge, but it is a fast way of getting the complete contents.

We're only going to retrieve the files for the module named Version_4:

- Startup WinCVS
- Select Create > Checkout Module
- As modulename enter Version_4

When WinCVS is finished, you should have a copy of all the files of Version_4 in you cvsroot directory.



Your screen probably looks slightly different from the above screendump since I downloaded all the modules that were in the Snitz repository at the time of writing. Note the revision numbers next to the filenames of the Version_4 modules. Those probably have changed also by the time you do this.

The files with revision number 1.1.1 are new files, meaning they only got checking in once, and haven't been changed/checked in after that. The readme.txt file has been changed and checked a couple of times since its revision number is 1.3

Comparing changes

Between different versions on the server

Lets take the readme.txt file and see what we can find out about it.

Between you local copy and the one on the server

If there is more than one developer that can change the files, or if you are just monitoring the CVS as a non-developer, then over time there will be differences between the version of a file you have stored locally and the ones on the server. You can review those differences also.

- Select Query > Diff selection
- Click on OK

But heay, nothing happened, no messages, nothing, that can't be right ?

Well if you followed this HOWTO step by step that *is* what should have happened, and you did get a messages BTW, it said:

```
cvs -q update -p readme.txt (in directory D:\cvsroot\Version_4\)
*****CVS exited normally with code 0*****
```

These two lines were the confirmation that the command got executed and the the result was empty, if everything goes as expected CVS exits wit code 0 (code 1 means that there has been an error).

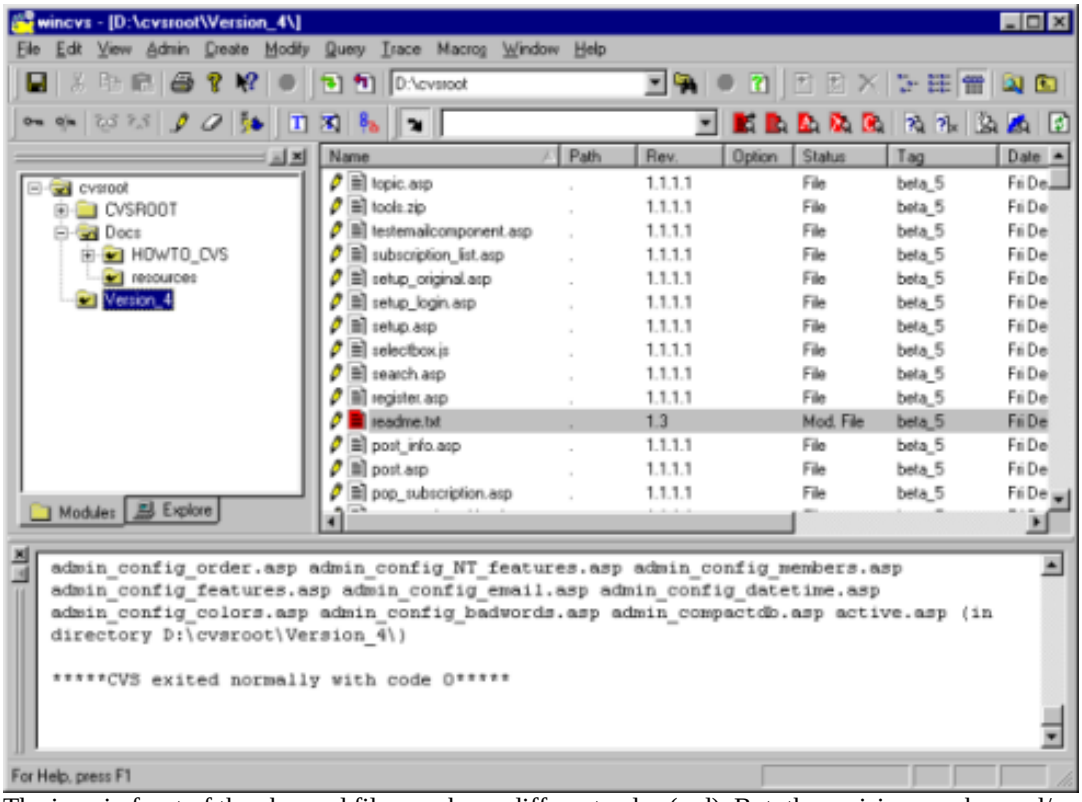
And of course you weren't really expecting differences between your copy of the file and the one on the server with the same revision, since it was only a couple of minutes ago that you downloaded a fresh copy and you hadn't changed it yet.

If you had changed the file in those couple of minutes, then both the copy on the server and your local copy would have been opened in CsDifft so you could look at the changes.

Commit changes

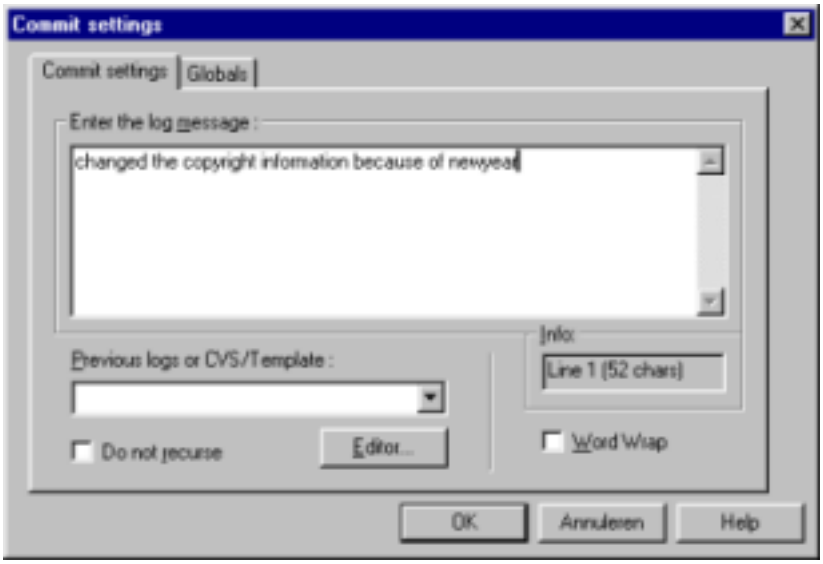
How does a developer make changes to a file on the server?

Well first the file has to be edited locally of course. This can be done with his or her favorite editor. Just open the file that is in the repository in cvsroot, edit it and save it. WinCVS doesn't have to be running while the file is being edited. When you've saved the file back into the repository and startup WinCVS you can see that the file has been changed:



The icon in front of the changed file now has a different color (red). But, the revision number and/or the file date in WinCVS haven't changed yet. That is because the changes still are only local. To get the changed file on the server we have to commit the changes.

- Select Modify > Commit Selection
- Enter a description of the changes made to the file



- Click on OK

You can read along with WinCVS by looking at the command-window:

```
cvs -q commit -m "changed the copyright information because of newyear" readme.
Checking in readme.txt;
/cvsroot/sf2k/Version_4/readme.txt,v  <--  readme.txt
new revision: 1.3.4.1; previous revision: 1.3
done

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

Working together

Problem?

If more than one person can edit a file, you're in for some problems. Issues like, "what if person A and B edit the same file at the same time and then commit their changes, what version will eventually be stored on the server" need to be resolved.

Editing the same file at the same time

CVS handles the problem of editing the same file at the same time for us.

Lets say person A and person B are editing the same file readme.txt revision 1.4. When person A is ready he commits the changes. The revision gets changed to 1.5 because of that. Person B, not knowing that person A edited the file, tries to commit his changes only a few minutes after person A, and gets an error:

```
cvs server: Up-to-date check failed for 'readme.txt'
cvs [server aborted]: correct above errors first!
```

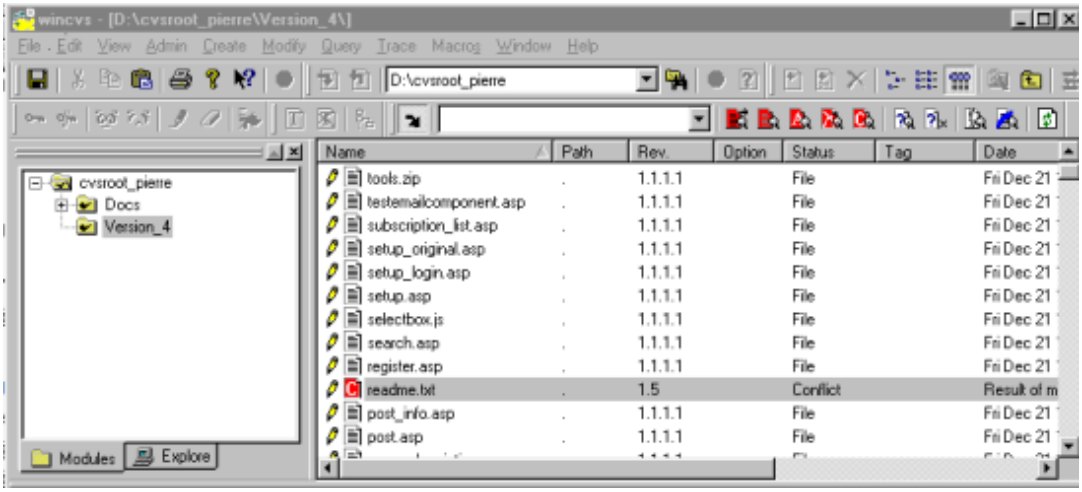
WinCVS has "seen" that the revision 1.4 wasn't the most recent anymore, since the server has a revision 1.5 of file readme.txt.

Person B now selects **Modify** > **Update Selection** in WinCVS, and sees this output:

```
RCS file: /cvsroot/sf2k/Version_4/readme.txt,v
retrieving revision 1.4
retrieving revision 1.5
Merging differences between 1.4 and 1.5 into readme.txt
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in readme.txt
C readme.txt

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

The conflict is no error (CVS exited normally), but something that has to be resolved. The fact that there is a conflict is indicated by both the status of the file and the icon that is used in WinCVS:



If you open the file you can see the conflicts indicated like this:

```
<<<<<<< readme.txt
text 1
=======
text 2
>>>>>>> 1.5
```

where "text 1" is the text like is appears in your local copy, and "text 2" is the text like it appears in the revision (1.5) on the server.

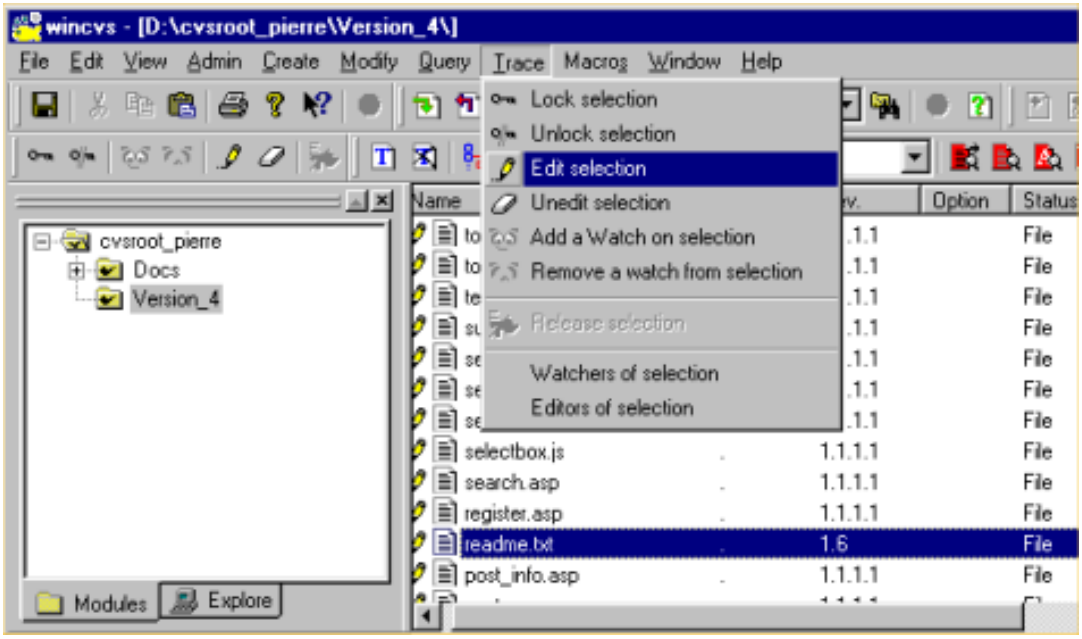
Resolving the conflict is easy: just edit the file so that the changes are merged, and then commit the file back to the server. The revision number then gets updated (again) so that the final revision number in this case is 1.6

If person A is smart enough to run the "update selection" command before editing the file again, he will see the new revision 1.6, otherwise he'll get the same possible conflicts when he tries to commit his version again.

This example shows that it is wise to update your local copy often when there are more developers working on the same code.

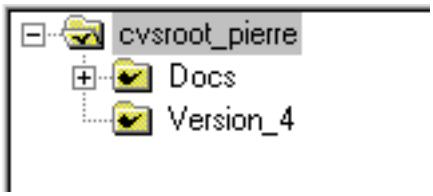
Editing / Watching / Locking

In the previous section I explained what happened when two people are editing the same file. CVS default doesn't use locking when you check out files. All developers are indicated as being editor. If you select the option **Trace > Unedit Selection** you'll notice that nothing happens, you'll still be indicating as editor.



To be able to use the Unedit command, you'll first have to set a "watch" on the files. You can find out if you already have set a watch on a file by choosing:**Trace > Watchers of selection**.

I encountered problems when trying to set a watch on the complete directory at once in Version 1.2. A workaround is to issue the correct command in the commandbox itself. In this case, you can set a watch on all the files in module Version_4, including new files you might add and subdirectories with:**cvs watch add -R Version_4** and remove it again with: **cvs watch remove -R Version_4**. Make sure that while issuing these commands you've got the root folder selected:



and not one of the modules, because it doesn't work with the command then either.

Now that you've set a watch on the files in the module, you can select to edit or unedit the files. First you have to select to edit the files. To do that for all the files at ones, click on the Version_4 folder and select: **Trace > Edit Selection**. After that you can select to Unedit the files by selecting: **Trace > Unedit Selection**. Others can see who is editing a file by selecting **Trace > Edtors of selection** .

This indication doesn't prevent others from editing and updating the file while you are editing it.

To prevent that you'll have to lock the file. You select **Trace > Lock Selection** to lock the file and **Trace > Unlock Selection** to unlock it again.

When a file is locked, other developers can still edit their local copy of the file, but can't commit the changes they made to the server as long as you have locked the file. Locking too many files can also cause problems because there is no visual hint for the other developers that a file is locked, not even after updating the local files with the ones from the server (where the locking information also is kept).

There are a couple of ways to find out if a file has has been locked: 1) try to lock it yourself, you'll receive an errormessage if someone else already has locked it, or 2) Select **Query > Log Selection**. The list you can see coming by also contains the locking information.

Tags / Branches

Tagging a module

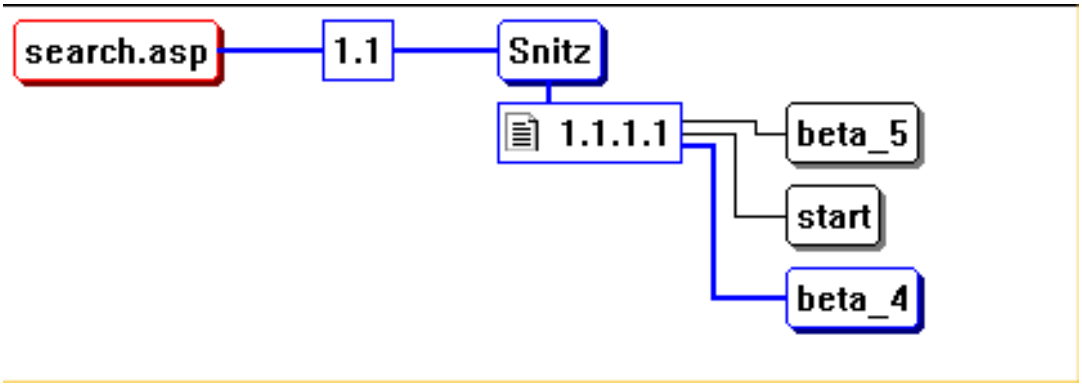
CVS automatically adds revision numbers to files when you check them in. But it is also possible to tag files or sets of files with your own label. That makes it possible to tag a current set of files with many different revisions as "beta_4" of "stableversion". That way you can easily go back to an older set of files by simple checking out the files with the same tag. So even if you already have progressed from beta_4 to beta_10, you'll still be able to compare the changes of a file or set of files between those two versions. You can also make a new branch starting at each tag. You add a tag to a complete module this way:

- Select Create > Create a tag by module
- Enter the tag-name (i.e. beta_5)
- Enter the module to be tagged (i.e. Version_4)
- Click OK

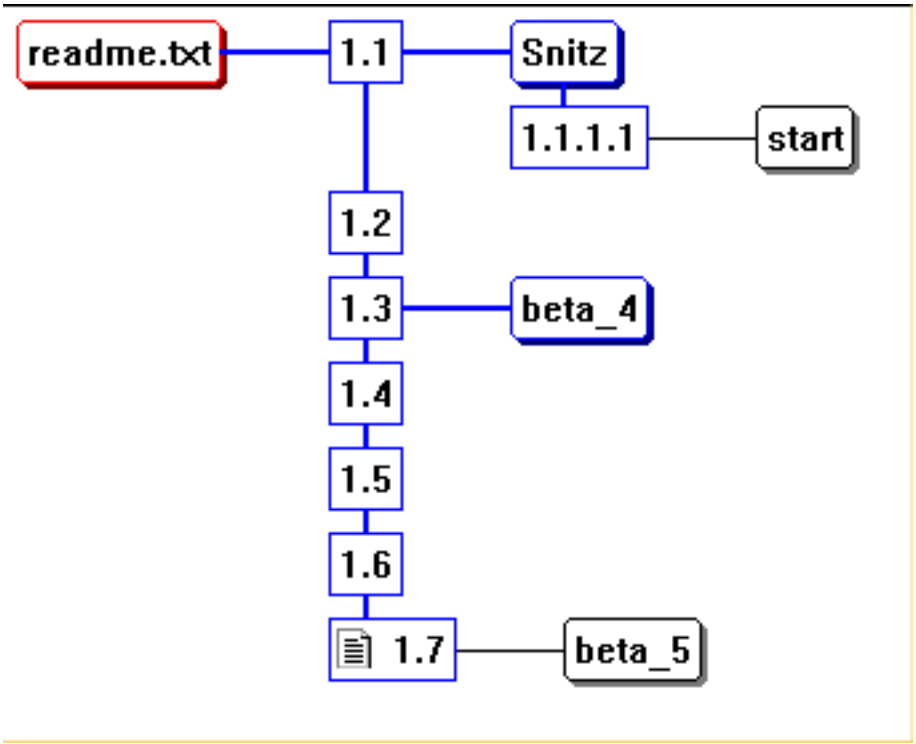
Viewing tags and branches in WinCVS

You can view the revisions and tags of a file graphical in WinCVS by rightclicking the file and selecting the **Graph Selection** command.

Depending on the changes made to the file and the tags attached to it, you'll see something like this:



Hmm, that doesn't look really interesting. The file search.asp has a couple of tags (Snitz, start, beta_5, beta_4) and only one revision 1.1.1.1 associated with it. This means this file has been checked in and hasn't been changed since. If we look at a bit more interesting file, we see this:



Now that is more like it. You can see that readme.txt currently has revision number 1.7

The tag beta_4 is associated with revision number 1.3 and tag beta_5 is attached to revision number 1.7

Between those two tags, the file has been checked in a couple of times (revisions 1.4, 1.5 and 1.6)

If I want to know the changes made to the beta_5 version of the file (revision 1.7) compared to the beta_4 version (revision 1.3) I can compare them:

- press the SHIFT button
- click on both files to select them
- select Graph > Diff Selected

Retrieving an older version/tag

Because I tagged de module with the tag "beta_4" I can select to check out all the files that have that tag.

That means that I don't get files that don't have that tag (because the are newer), and that I get the revision of the file that belongs to that tag. In case of readme.txt that means I get revision 1.3, even though the current revision is 1.7

You probably won't do this with beta versions, but if instead of tags beta_4 and beta_5 you had tags for version 4.1 and version 4.2 and you discovered that there was an error in the readme.txt file. The error had been introduced in revision 1.2 and was still present in revision 1.7.

One way of fixing it would be to open the revision 1.7 file, fix the error there, save the file, commit the changes (CVS increases the revision number to 1.8) and release a version 4.2.1 which uses

readme.txt with revision number 1.8 of the readme.txt file. No problem with that.

But what about the 4.1 version ? That version also has the error in readme.txt, but doesn't get updated. Let's fix that version also.

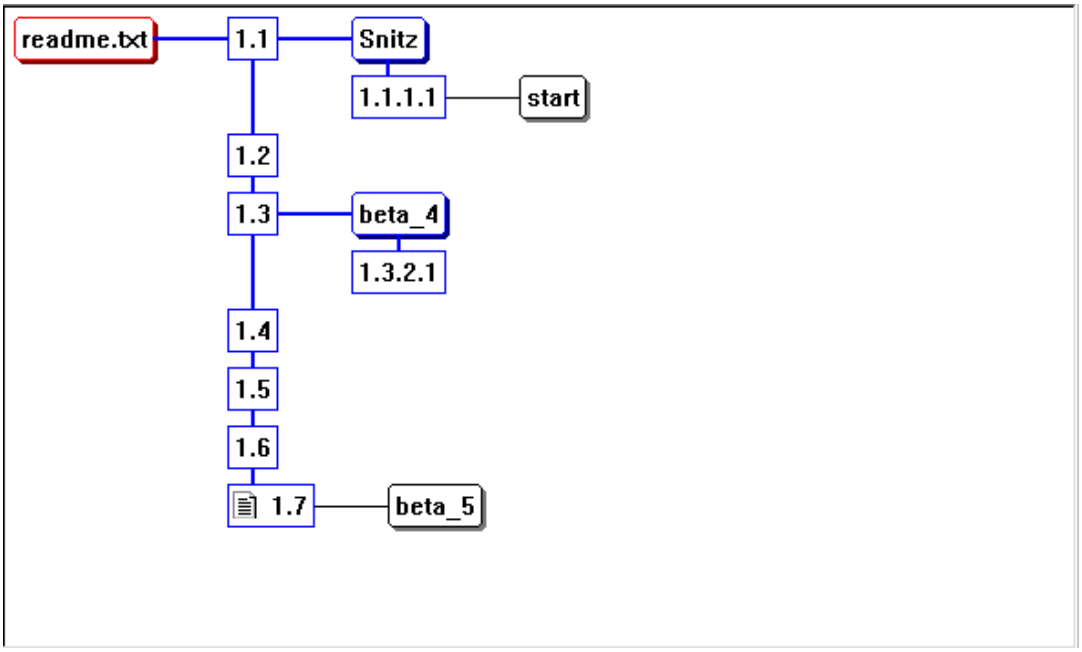
First, since we're going to retrieve the revisions that belonged to the old tag, then we're going to change the file with the error and then we'll upload the changed file back to the server again and have a look at how that affected our current version. I'll be showing this using the beta_4 and beta_5 examples:

- Select Create > Checkout module
- On the Checkout settings tab, check the 'override' option and enter a new name for the module: 'beta_4'
- On the Checkout options tab, check 'By revision/tag/branch' and enter the tagname we want to checkout: 'beta_4'
- Click OK

Now, all the files that have tag beta_4 get checked out into a new module that we also called beta_4. Because we didn't check them out to the default Version_4 module, that module stays unchanged. This way we can still keep on working on the current versions of the files while we are fixing the bug in the beta_4 version.

We can now go into the beta_4 module, open the readme.txt (with revision number 1.3) and fix the error. Then we commit the changes to the server. The revision number gets updated automatically again and is changed to 1.3.2.1

If we go back to our Version_4 module we can see that the revision of readme.txt we had there hasn't been altered, nor has the revision number changed. We can see what did happen if we graph the file again:



The modification we made to the file have been added as a branch to the beta_4 tag. Next time if someone checks out all the beta_4 tagged files, they will get the revision 1.3.2.1 of readme.txt

If someone checks out the most current version of readme.txt or the beta_5 tagged files, they will still get the 1.7 revision.

Chapter 4. Copyright

If you have questions, please contact me, at *HOWTO@PGorissen.com [mailto:HOWTO@PGorissen.com]* via email.

Visit Snitz Forums 2000 on the Web at http://forum.snitz.com/ [http://forum.snitz.com/].