CVS Mini Howto

Table of Contents

CVS Mini Howto

HCVS Mini-Howto 06 Jul 05

- Create the repository (requires full path)

```
$ cvs -d /home/fringer/cvsroot init
```

- Set the environmental variable so that cvs knows where the repository is

```
$ echo "export CVSROOT=~/cvsroot" >> ~/.bashrc
```

Note: To work from the suntans repository, you would use

```
export CVSROOT=:ext:cvsuser@suntans:/home/cvs
export CVS_RSH=ssh
```

- Import files into a new project
    1. Go to the directory that contains your files

```
$ cd hello
$ ls
hello.c
```

    2. Place the CVS macros in your file to write the logs in your file

```
/*
 * $Id$
 * $Log$
 *
 */
```

    3. Import the files in the local directory

```
$ cvs import -m ``CVS Test'' hello hello-import start
N hello/hello.c

No conflicts created by this import
```

- Now you can delete the directory and check it back out so that it is now under CVS control

```
$ rm -rf hello
$ cvs checkout -d hello-debug hello
```

where the `-d` flag creates the directory `hello-debug` and places the repository file into that directory

- To check the difference between a file and the files in the repository

```
$ cvs diff hello.c // Only hello.c
$ cvs diff *.c // Only files with .c extension
$ cvs diff // All files
```

For example, suppose we add the following line to `hello.c`:

```
printf(``Goodbye World!\n'');
```

Then we can check to see what is different about our file with

```
$ cvs diff hello.c
Index: hello.c
===================================================================
RCS file: /home/fringer/.cvsroot/hello/hello.c,v
retrieving revision 1.1.1.1
diff -r1.1.1.1 hello.c
12a13,14
>
>   printf("Goodbye World!\n");
```

- To commit the changes to the repository, use

```
$ cvs commit -m ``Added a new line'' hello.c
Checking in hello.c;
/home/fringer/.cvsroot/hello/hello.c,v  <--  hello.c
new revision: 1.2; previous revision: 1.1
done
```

- We can check the logs on the given file to see its history of changes with `cvs log hello.c`
- We can also revert to a previous revision with

```
cvs update -r 1.1 hello.c
```

This will update the file `hello.c` to its state at revision 1.1. If we change this file and then try to commit those changes we'll get an error. For example, suppose we add the line `printf(``This is a new line\n'');` to the file. If we do a `cvs diff` on this file we get

```
cvs commit: sticky tag `1.1' for file `hello.c' is not a branch
cvs [commit aborted]: correct above errors first!
```

This is because the working copy is version 1.1, and we are trying to commit changes to a file that had already been changed and for which we had already added those changes to the repository to create revision 1.2. The best way to avoid this problem is to make sure you are always working on a ``non-sticky'' revision number. You can bring all files up to date with `cvs update -A`, but make sure you save the changes to your working file first! The problem we'll have, however, is that if you try to update your current file, it will merge the changes in your file with those in the repository. That is, you will get

```
cvs update: Updating .
RCS file: /home/fringer/.cvsroot/hello/hello.c,v
retrieving revision 1.1
retrieving revision 1.2
Merging differences between 1.1 and 1.2 into hello.c
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in hello.c
C hello.c
```

And the file will look like

```
/*
 * $Id: hello.c,v 1.2 2005/07/06 16:37:48 fringer Exp $
 * $Log: hello.c,v $
 * Revision 1.2  2005/07/06 16:37:48  fringer
 * Added a new line
 *
 * Revision 1.1.1.1  2005/07/06 16:34:10  fringer
 * CVS Test
 *
 *
```

```
 */
#include<stdio.h>

int main(void) {
  printf("Hello World!\n");
<<<<<<< hello.c

  printf("This is a new line\n");
=======

  printf("Goodbye World!\n");
>>>>>>> 1.2
}
```

So that if you try to compile this file it will break. To avoid this problem you should move your working copy of the file into a temporary file and then perform an update, i.e.

```
$ mv hello.c hello.c.bak
$ cvs update -A hello.c
```

And then edit this version of `hello.c` by looking at the changes you implemented in `hello.c.bak`.

- You can add files to the repository with

```
$ cvs add goodbye.c
cvs add: scheduling file `goodbye.c' for addition
cvs add: use 'cvs commit' to add this file permanently
```

And commiting the addition with

```
$ cvs commit -m ``Added goodbye.c'' goodbye.c
RCS file: /home/fringer/.cvsroot/hello/goodbye.c,v
done
Checking in goodbye.c;
/home/fringer/.cvsroot/hello/goodbye.c,v  <--  goodbye.c
initial revision: 1.1
done
```

- You can also add directories with

```
$ cvs add dir1
Directory /home/fringer/.cvsroot/hello/dir1 added to the repository
```

But cvs does not recursively add files in the directory so you need to go into the directory and manually add those files.

- You can also assign a tag to the current files in the directory so that you can checkout all the files which were current at the point in which you added the tag. For example, we could add a tag as follows

```
$ cvs tag hello-tag-working-070605 hello-debug
cvs tag: Tagging hello-debug
T hello-debug/goodbye.c
T hello-debug/hello.c
cvs tag: Tagging hello-debug/dir1
cvs tag: Tagging hello-debug/dir2
```

If we edit these files and check in the changes, and then return to find that compilation breaks, we can revert all files to the point at which the tag was assigned with

```
$ cvs co -r hello-tag-working-070605 -d hello-debug hello
cvs checkout: Updating hello-debug
U hello-debug/goodbye.c
U hello-debug/hello.c
cvs checkout: Updating hello-debug/dir1
cvs checkout: Updating hello-debug/dir2
```

Note that even though these have been checked out under the same tag, they do not necessarily have to have the same revision number, that is,

```
$ cvs status
cvs status: Examining .
===================================================================
File: goodbye.c         Status: Up-to-date

   Working revision:    1.1     Wed Jul  6 16:52:32 2005
   Repository revision: 1.1     /home/fringer/.cvsroot/hello/goodbye.c,v
   Sticky Tag:          hello-tag-working-070605 (revision: 1.1)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: hello.c           Status: Up-to-date

   Working revision:    1.2     Wed Jul  6 16:37:48 2005
   Repository revision: 1.2     /home/fringer/.cvsroot/hello/hello.c,v
   Sticky Tag:          hello-tag-working-070605 (revision: 1.2)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

It is possible to update the version number so that all files in the repository contain the same version number, using

```
cvs commit -m "Updating major revision number" -r 3.0 hello-debug
```

This will update the version numbers of all the files so that the next checkout will checkout all files with version number 3.0. To see this history, use `cvs log`:

```
.
.
.
revision 3.0
date: 2005/07/06 17:17:47;  author: fringer;  state: Exp;  lines: +4 -1
Updating major revision number
----------------------------
revision 1.2
date: 2005/07/06 16:37:48;  author: fringer;  state: Exp;  lines: +7 -2
Added a new line
.
.
.
```

- Once you are finished with your directory, you can release it from cvs control and then delete it with

```
cvs release -d hello-debug
You have [0] altered files in this repository.
Are you sure you want to release (and delete) directory `hello-debug': y
```

It is a good idea to delete directories that are not under cvs control (but which are in the repository) so

that you don't find yourself editing a file and then realizing it is not under cvs control after you have made several changes.

Next: About this document ... Oliver Fringer 2005-07-07