Date: 3rd March 2005

Source Code Management/Version Control

The Problem:

In a typical software development environment, many developers will be engaged in work on one code base.

If everyone was to be allowed to edit and modify any item whenever they felt, it ends into chaos.

How do you handle the source code?

The Hard way:

You designate a guy to maintain the golden copy. As you finish code you give it to him, and he places it in a directory.

- Forces a guy to do this manually
- What if two people is working on the same file?
- What if you want to go back to an older version of a file?
- What if you want to maintain multiple “forks” of the program?

Instead of suffering under such an environment, most developers prefer to implement the *Source Code Management (SCM)* Systems or *Version Control* Tools.

Source Code Management (SCM):

These are the problems source code management is intended to solve. Effectively it is a database for source code that eases the problems of

- Multiple people in many places working on the same code
- Retrieving old versions of files
- Keeping logs about what changed in a file
- Integrating changed code
- Generating release builds

What it is NOT:

It can do a lot of things for you, but it does not try to be everything for everyone. It is not,

- A replacement for communication
- A replacement for management
- A substitute for other tools, such as bug tracking, mailing lists, and distribution
- It has no understanding of the semantics of your program; as far as it's concerned, It's all just text.

Version Control Tools:

There are many Version Control Tools.

1. Project Revision Control System (PRCS)
2. Source Code Control System (SCCS)
3. Revision Control System (RCS)
4. Concurrent Version System (CVS)

Project Revision Control System (PRCS):

The Project Revision Control System (PRCS) is the front end to a set of tools that (like CVS).
It provides a way to deal with sets of files and directories as an entity, preserving coherent versions of the entire set.
Its purpose is similar to that of SCCS, RCS, and CVS, but (according to its authors, at least) it is much simpler than any of those systems.

http://prcs.sourceforge.net/

Source Code Control System (SCCS):

The Source Code Control System (SCCS) was developed at Bell Telephone Laboratories in 1972.
Though SCCS lacks some of the amenities of RCS (a natural result of its early date of development), it is a generally equivalent system and has a few capabilities that RCS does not.

http://www.oreilly.com/catalog/rcs/chapter/ch04.html

Revision Control System (RCS):

The Revision Control System (RCS) was designed at the Department of Computer Science at Purdue University in 1982.
RCS is a software tool for UNIX systems which lets people working on the system control "multiple revisions of text ... that is revised frequently, such as programs or documentation."
It can be applied to development situations of all sorts, including the creation of documents, drawings, forms, articles, and of course, source code.

http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/rcs/

Concurrent Version System (CVS):

CVS is an open source version control system layered on top of RCS, designed to manage entire software projects.

It is an important component of Source Configuration Management (SCM).

Basics description of the CVS system:

It works by holding a central `repository' of the most recent version of the files.

You may at any time create a personal copy of these files by `checking out' the files from the repository into one of your directories.

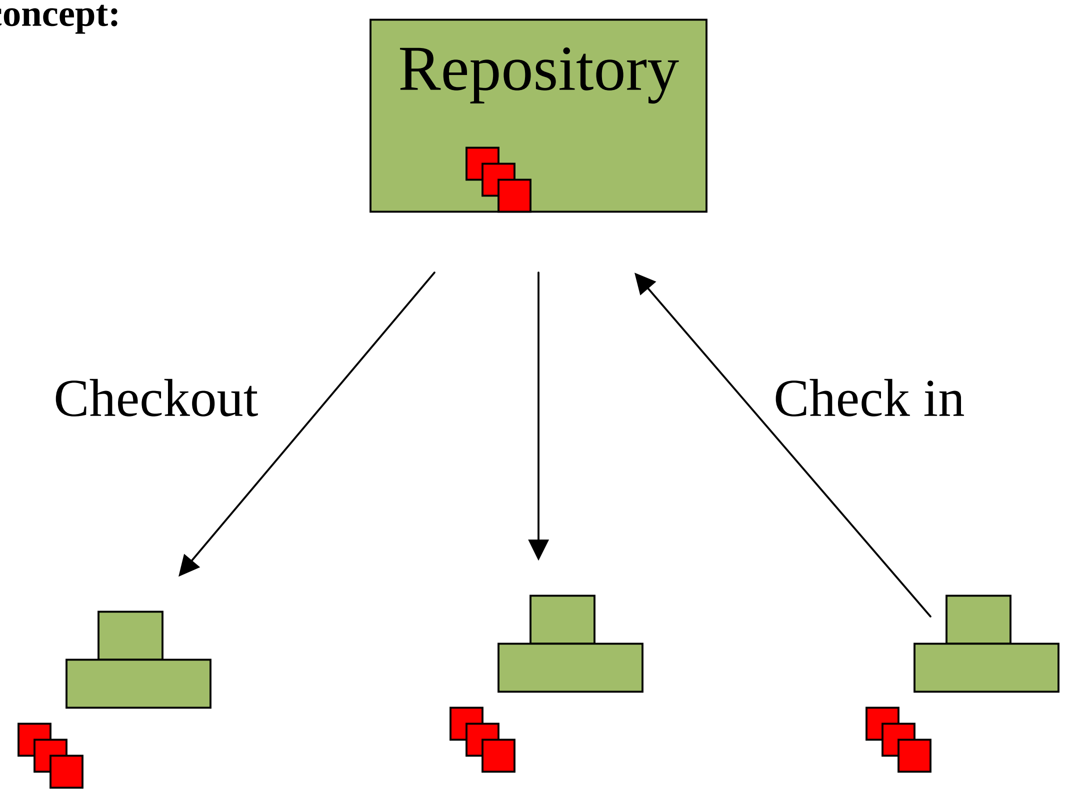
If at a later date newer versions of the files are put in the repository, you can `update' your copy.

You may edit your copy of the files freely.

When you are satisfied with the changes you have made in your copy of the files, you can `commit' them into the central repository.

When you are finally done with your personal copy of the files, you can `release' them and then remove them.

The concept:



You need two pieces of software to do CVS: a server and a client. The server handles the database end, the client the local side.

Server Side: Repository Directory
Client Side: Working/Local Directory

Some basic words and descriptions:

Versions, revisions and releases:

A file can have several versions; likewise, a software product can have several versions.

A software product is often given a version number such as `2.1.1'. Versions in the first sense are called *revisions* here, and versions in the second sense are called *releases*. To avoid confusion, the word *version* is almost avoided.

Repository:
The Repository is the directory, which stores the master copies of the files. The main or master repository is a tree of directories.

Module:
It is a specific directory (or mini-tree of directories) in the main repository. Modules are defined in the CVS modules file.

A Sample Session - Starting a project with CVS:

Server Side – Repository Directory:

CVS can access a repository by a variety of means. It might be on the local computer, or on a computer across the room or across the world.

Steps:

If possible create separate login and create directory for repository (cvsroot). *i.e.* /home/repository/.cvsroot

To tell CVS where to find the repository, set the environment variable $CVSROOT to an absolute path to the root of the repository. *i.e.*
all csh and tcsh users should have this line in their `.cshrc' or `.tcshrc' files:

```
setenv CVSROOT /home/repository/.cvsroot
```

sh and bash users should instead have these lines in their `.profile' or `.bashrc':

```
export CVSROOT=$HOME/.cvsroot
```

Verify it by $CVSROOT from the Terminal.

To initialize the repository tree run the cvsinit script that comes with CVS. For that run below command from /home/repository/.cvsroot.

```
cvs init
```

Creates CVSROOT directory inside /home/repository/.cvsroot.

Now, copy your interest of project, which you want to put it in CVS to /home/repository/. *i.e.* Test (here Test is your project directory).

After it, import this directory into repository. For that you must run this command from Test (/home/repository/Test/) directory only.

```
cvs import -m 'comment' repository ‘vendortag’ ‘releasetag’
```

The comment is for you to document the module.

The repository should be a path under $CVSROOT. *i.e.* Test

vendortag is a release tag that the vendor assigned to the files. If you are the vendor then put whatever you want there: (i.e. TEST_1_01).

Releasetag is your local tag for this copy of the files. (i.e. test_1_01).

For example, the final command looks like this,

```
cvs import –m ‘Testing’ Test ‘Test_1_01’ ‘test_1_01’
```

So now, your project is imported to /home/repository/.cvsroot directory.

The repository is split in two parts:

1. Administrative files (`$CVSROOT/CVSROOT')
2. User-defined Modules

The overall structure of the repository is a directory tree corresponding to the directories in the working directory. The detailed structure has not been discussed here.

With the directories are *history files* for each file under version control. The name of the history file is the name of the corresponding file with `,v’ appended to the end.

You can also name the repository on the command line explicitly, with the -d (for "directory") option.

```
cvs -d /home/repositiry/.cvsroot checkout Test
```

A repository specified with -d will override the $CVSROOT environment variable.

Client Side - Working Directory:

Now, on client side to create your own working directory of the project, which is under CVS,
Make sure that you are in the same UNIX group and you have the permissions to read/write the repository files/directories.

Basic commands of CVS:

Most of the below commands should be executing while in the directory you checked out.

If you did a cvs checkout Test then you should be in the Test sub-directory to execute most of these commands.

cvs release is different and must be executed from the directory above.

cvs checkout (or cvs co) module

> To make a local copy of a module's files from the repository execute “cvs checkout module” where module is an entry in your modules file. This will create a sub-directory module and check out the files from the repository into the sub-directory for you to work on.

cvs update

> To update your copy of a module with any changes from the central repository, execute “cvs update”. This will tell you which files have been updated (their names are displayed with a U before them), and which have been modified by you and not yet committed (preceded by an M).

ALWAYS DO AN UPDATE BEFORE STARTING TO WORK ON THE PROJECT.

You will be warned of any files that contain clashes by a preceding C. Inside the files the clashes will be marked in the file surrounded by lines of the form <<<< and >>>>.

You have to resolve the clashes in your copy by hand.

If you feel you have messed up a file and wish to have CVS forget about your changes and go back to the version from the repository, delete the file and do an cvs update.

CVS will announce that the file has been "lost" and will give you a fresh copy.

cvs commit –m ‘message’

> When you think your files are ready to be merged back into the repository for the rest of your developers to see, execute “cvs commit –m ‘message’”. You will be put in an editor to make a message that describes the changes that you have made (for future reference). Your changes will then be added to the central copy.

cvs commit –m ‘message’ filename

> It commits only the specified file (filename) to the repository. Run it from your current directory.

cvs add and cvs remove

> It can be that the changes you want to make involve a completely new file, or removing an existing one. The commands to use here are:
>
> ```
> cvs add `filename'
> cvs remove `filename'
> ```

You still have to do a commit after these commands to make the additions and removes actually take affect.

You may make any number of new files in your copy of the repository, but they will not be committed to the central copy unless you do a cvs add.

CVS remove does not actually remove the files from the repository. It only removes them from the "current list" and puts the files in the CVS Attic.

When another person checks out the module in the future they will not get the files that were removed.

But if you ask for older versions that had the file before it was removed, the file will be checked out of the Attic.

cvs release

When you are done with your local copy of the files for the time being and want to remove your local copy use cvs release module. This must be done in the directory above the module sub-directory you which to release. It safely cancels the effects of cvs checkout. Usually you should do a commit first.

If you wish to have CVS also remove the module sub-directory and your local copy of the files then your cvs release -d module.

NOTE: Take your time here. CVS will inform you of files that may have changed or it does not know about (watch for the ? lines) and then with ask you to confirm this action. Make sure you want to do this.

cvs log

To see the commit messages for files, and who made them, use:

```
cvs log [`filename(s)']
```

cvs diff

To see the differences between your version of the files and the version in the repository do:

```
cvs diff [`filename(s)']
```

cvs tag

One of the exciting features of CVS is its ability to mark all the files in a module at once with a symbolic name. You can say `this copy of my files is version 3'. And then later say `this file I am working on looked better in version 3 so check out the copy that I marked as version 3.'

Use cvs tag to tag the version of the files that you have checked out. You can then at a later date retrieve this version of the files with the tag.

```
cvs tag tag-name [filenames]
```

Later you can do:

```
cvs co -r tag-name module
```

cvs rtag

Like tag, rtag marks the current versions of files but it does not work on your local copies but on the files in the repository.

```
cvs rtag VERSION_2_0 Test
```

cvs history

To find out information about your CVS repositories use the cvs history command. By default history will show you all the entries that correspond to you. Use the -a option to show information about everyone.

```
cvs history -a -o   shows you (a)ll the checked (o)ut modules
cvs history -a -T   reports (a)ll the r(T)ags for the modules
cvs history -a -e   reports (a)ll the information about (e)verything
```

CVS Help:

All CVS commands take a -H option to give help:

cvs -H

Shows you the options and commands in cvs.

cvs history -H

Shows you the options for CVS history.

cvs –n update

To see which files would be updated? The flag –n doesn't do the action, but let you see what would happen.

man cvs

Shows the manual page.

CVS Links:

http://www.cvshome.org
http://www.gnu.org/software/cvs/manual/html_mono/cvs.html#SEC1

Prepared By: Satish Kagathara