CHAPTER 6

Introduction to CVS

The Concurrent Versions System, or CVS, is a source code control application that permits developers to work in parallel on software development projects without the fear of overwriting someone else's work. In addition, it provides a complete project history that includes the ability to review previous versions of the code, and to compare old code with new code.

CVS can be used for any development project, including assembly and machine language to C and C++ and even the code and images necessary to maintain a web site.

The source code is set up in a software repository by the administrator. Each repository is identified by a unique location. A repository may be on a local machine or maintained on the network and accessed remotely. Many Open Source developers utilize anonymous CVS access in order to distribute development snapshots of the source code instead of (or in addition to) tarballs, RPM and other methods of distribution. Each repository many contain multiple projects and many versions of each project. The current version of a project may be checked out, as may older versions. Versions may also be compared to one another to determine specific changes and help to identify bugs that may have crept into a piece of software.

6.1 CVS Policies

In addition to the technical aspects of setting up CVS and granting users access to the system, there are certain operational procedures that need to be set forth prior to opening up a CVS repository for access.

One of the more important aspects deals with users checking in or committing a change into the repository.

Every time a source file is checked into the system, the version number of that file is incremented. This version is separate from the version number of the software project. The differences will be described later.

Each team or project will have to develop guidelines as to when a source file should be checked in, how long one may be checked out prior to committing it, and similar items. Checking a file out for a long period of time, making modifications and then checking it back in may result in conflicts that will have to be addressed. CVS will let you check the file in if your changes conflict with someone else's work, but someone will have to be responsible for ensuring that these conflicts are resolved.

Another policy might state that a file may not be checked back into the repository until it compiles cleanly with the rest of the project, until it has been reviewed, or until other benchmarks are met. These policies will vary from project to project, and while CVS can maintain the source code repository, it has no method of enforcing these policies. It is the role of the project owner and manager(s) to ensure that the policies are implemented and followed by the team members.

6.2 Project Management and Communication

While CVS has many tools to help you manage a software project, it has no inherent ability to manage the project on its own. There are e-mail notifications available that let individuals keep an eye on particular files for changes. There are reports available that can determine who changed which file when, as well as complete logs that describe what those changes did (or at least, were supposed to have done).

As mentioned previously, even using a version control system such as CVS will not prevent conflicts of code from arising. Communication between developers is absolutely necessary in order to ensure that projects proceed smoothly and on time.

6.3 Installing and Managing CVS

In order to install CVS, the first step is to obtain either the CVS source code or, if you are using RedHat, the RPM.

The latest source code is available at http://www.cvshome.org. As of this writing, the current CVS version is 1.11.1p1. RPM may be downloaded via RPMFind at http://rpmfind.net/linux/rpm2html/search.php?query=cvs. Follow the instructions for your preferred installation method.

6.3.1 Configuring CVS

Once CVS is installed on the server, the next step is to configure the machine properly to use CVS. The first step is to create a user and group to be the owners of the repositories. For purposes of example, assume that the user and the group are both named "`cvs`". This example shows how to set up CVS for local and remote "pserver" access. This method of setting up CVS is not the most secure method and does not provide a secure method for accessing the CVS repository remotely; but it is necessary if a project intends to provide remote, anonymous access to the CVS repository. For details on setting up a secure method of accessing the repository, see Section 6.7.1.

To accomplish this, as root, issue the `useradd` command listed below. Under RedHat Linux, this will create the `cvs` user and the `cvs` group. All of the following commands must be issued as the root user.

```
# useradd -r -c "CVS user" cvs
```

As initialized, this account will have no password and will not be able to be logged into. You can either leave it this way and, in the event that maintenance is necessary, log in as root and resolve issues, or assign a password and use that account. Leaving the account with no password is probably the most secure, but in the event that a password is necessary, simply assign one.

```
# passwd cvs
Changing password for user cvs
New UNIX password: <type cvs password>
Retype new UNIX password: <type cvs password>
passwd: all authentication tokens updated successfully
```

The next step is to create a CVS repository. For this example, assume that the CVS root is `/usr/local/cvsroot`. Additional repositories may be set up at this time, simply issue the CVS `init` command as many times as necessary; while changing the **–d** option to reflect the additional repositories.

```
# cvs –d /usr/local/cvsroot init
```

This will create the file structure necessary for CVS to function and will initialize all necessary files.

Once the user and repositories have been created, the ownership of the repositories needs to be changed.

```
# chmod –R cvs:cvs /usr/local/cvsroot [additional repositories here]
```

The next step is to add users to the repositories. Each repository has a password file in the `$CVSROOT/CVSROOT` directory. There are three fields in this file separated by colons. The first field is the user name that may or may not be the same as the user's login id. The second field is the user's encrypted password. This password is encrypted with the `crypt()` function. The final field is the name of the user whose request will be performed. This is the owner of the CVS repository and in our example, the owner is '`cvs`'.

The easiest way to generate encrypted passwords for the password file is by using a simple perl script. An example is presented below. Simply type this script into a file using a text editor and make the file executable. Then run the program with two arguments, the username and the password.

```
#!/usr/bin/perl
#
# Simple script to take a username and password and
# return a line suitable for pasting into the CVS
# password file.
#

($u, $p)=@ARGV;
@d=(A..Z,a..z);
$s=$d[rand(52)].$d[rand(52)];
print $u.":".crypt($p, $s).":cvs\n";
```

If you have saved the file under the name '`cvspassgen`' and set the permission properly, simple run the application as follows:

```
# ./cvspassgen username password
username:mU7OltQzHySmY:cvs
```

As you can see, the information is returned in the format that CVS requires. (Note: since a random salt is used to generate the password, your output will vary from what is presented here.)

You can simply cut and paste the line into the password file. Another option is to redirect the output of the script directly into the password file.

```
# ./cvspassgen username password >> $CVSROOT/CVSROOT/passwd
```

This assumes that the `$CVSROOT` environmental variable is set to the proper directory.

Next, the services file must be updated in order for the CVS server to respond to network requests. The following line will append to the `/etc/services` file. Just be sure that there are two "greater than" symbols or you will erase whatever is already in your services file.

```
# echo "cvspserver 2401/tcp" >> /etc/services
```

Next, the `inetd` server needs to know which service to start when requests come in. RedHat 7.x uses the `xinetd` server and so a file needs to be created in `/etc/xinetd.d` named `cvspserver`.

Using your favorite text editor, edit /`etc/xinetd.d/cvspserver` to include the following information:

```
# Description: cvspserver: version control system
service cvspserver
{
        socket_type = stream
        wait        = no
        user        = cvs
```

```
                passenv     =
                server      = /usr/bin/cvs
                server_args = --allow-root=/usr/local/cvsroot pserver
        }
```

If you have installed multiple repositories, they need to be specified on the `server_args` line. To do this, simply insert additional `–allow-root=` commands. Please note that these additional arguments **must** appear on the same line.

In order for `xinetd` to acknowledge this change, it needs to reread its configuration files. To do this, send the server a `HUP` signal with the `killall` command.

```
# killall -HUP xinetd
```

At this point, the repository should be up and running.

To quickly review the steps for installing CVS:

1. Install the CVS application (RPM or source code)
2. Create CVS user and group
3. Create CVS repositories
4. Change the owner and group of the repositories
5. Add users to `$CVSROOT/CVSROOT/passwd`
6. Change file permissions on the password file(s)
7. Edit `/etc/services` to ensure that `cvspserver` is present
8. Configure `/etc/xinetd.d` to acknowledge CVS requests
9. Restart `xinetd`

6.3.2 Importing a Project into the Repository

Once you have CVS installed and configured, the next step is to import a project into the repository. This creates a directory for the project, copies the files into the repository and sets up the information that will be used to track the project as it gets modified during the development process.

Assuming that the directory `/home/lcp/fuzion` holds the source code and the files of the project, to import the project you need to change to that directory and issue the CVS import command.

The options for the import command include the name of the project, the vendor tag, and the release tag(s). The vendor tag is not currently used by CVS, and can contain any information that you care to put in. The release tag can be used to designate specific releases, but the source code can also be checked out by the numeric release designation. Any information that you care to put in these fields may be used, but they must be included with the import command.

```
# cd /home/lcp/fuzion
# cvs import Fuzion OSS_Software release_0_1
```

At this stage, you will be presented with your preferred editor (as defined by the environment variables $CVSEDITOR or $EDITOR). This will allow you to create a log entry that describes the project, the initial state that it is in, and anything else of importance.

When you save that file, the information, along with the imported files, will be saved in the repository.

To avoid having the editor displayed, you can include a log entry on the command line with the **–m** option. Log entries with spaces or other special characters in them should be enclosed in quotes.

```
# cvs import -m "Initialized the project" Fuzion \
OSS_Software release_0_1
```

To ensure that the files have been imported, issue the following command from the server that holds the repository:

```
# ls -l $CVSROOT/Fuzion
```

You should see a copy of each file in your project with a ',v' appended to the filename.

At this point, you should remove the original directory (after ensuring that you have a backup) and check out the project with CVS so that any modification can be committed to the repository.

```
# cd /home/lcp
# rm -rf fuzion
# cvs co Fuzion
# cd Fuzion
```

At this point you are ready to continue development on this project with CVS.

NOTE Special care must be taken if you are importing binary data into a CVS repository. This can be the case if you are importing files from a Windows platform, or if you use CVS to manage the files necessary for maintaining a web site. Graphic files and dlls will not import correctly unless special care is taken.

One way to do this is with the wrapper function of CVS. To import a project and treat all files with a .jpg extension as binary data, issue the following command:

```
cvs import -I ! -W "*.jpg -k 'b'" Pname V_tag R_tag
```

6.4 Using the CVS Client

There are several methods of accessing a CVS repository. The first is local access. This can be accomplished by a user logging in at the terminal or logging in remotely over the network. Either way, the developer must have a local account on the machine, a home directory, and the repository to be accessed via the local session.

Remote access can take several forms. This can be a remote connection from a user's machine to the `cvspserver` that runs on port 2401. Another method would be to have CVS connect via a remote shell (rsh) when the repository needs to be accessed.

CVS access takes place in clear text over the network with no inherent facility to prevent passwords and data from being intercepted and used by unauthorized persons to access the server. Several options later in the chapter describe a more secure method of access. One of those methods is similar to the `rsh` method, but the `CVS_RSH` environment variable is set to the `ssh` instead of the insecure `rsh` command.

6.4.1 Local Repositories

In order to work easily and efficiently with CVS, it is necessary to set up a series of environmental variables that tell CVS where your CVS repository is, which editor you prefer, etc. These variables can be set in either the system-wide or user specific files that are read when the user logs in, or each user may set or change them manually.

The first variable is `CVSROOT` and it tells CVS which repository that you wish to work with. Assuming that your CVS repository is located on the local machine, simply set the `CVSROOT` variable to the location of the repository.

If your repository resides at "`'/usr/local/cvsroot`'" and you are using `sh`, `bash`, `ksh` or a similar shell, simply insert the following line into your startup file (`.profile, .bash, etc.`)

```
export CVSROOT=/usr/local/cvsroot
```

If you are using `csh` or `tcsh`, insert the following line into your `.cshrc` or `.tcshrc` file:

```
setenv CVSROOT /usr/local/cvsroot
```

Should you wish to temporarily work with a different repository, the path to the repository can be supplied on the command line with the **-d** option, as in this example:

```
$ cvs -d /usr/local/cvsother checkout BigProject
```

Once a project has been checked out of the repository, this information is also kept in the `CVS/Root` file in the project directory, as shown below:

```
$ cat CVS/Root
/usr/local/cvsroot
```

The information in the `CVS/Root` file takes precedence over the `CVSROOT` variable in order to ensure that the files are checked back into the same repository that they were checked out from.

The `CVS/Root` file may be overridden by the `-d` command line option. This allows you to specifically place the file into a repository other than the one from which it was checked out.

To sum up, which checking a file back into the repository, the CVS root is determined the following order:

1. The **-d** command line option
2. The CVS/Root file
3. The CVSROOT environment variable

6.4.2 Remote Repositories

As mentioned earlier, there are several methods of accessing a remote CVS server. The first one we will discuss is the built-in functionality. This method utilizes a network connection from the developer's desktop to port 2401/tcp on the server. It requires that all of the intermediate networking equipment (routers, firewalls, etc.) permit this traffic to pass.

Accessing a repository using the pserver method requires additional information when compared to the local access in order to determine the name of the remote host, the user name to be used, and the remote connection method.

For example, in order to log into a remote CVS repository, you would need to specify the following:

```
# export CVSROOT=:pserver:user@hostname:/usr/local/cvsroot
```

Where user is your CVS username on the machine hostname. This may be used in the CVSROOT variable or on the command line with the **-d** option.

Prior to checking out, committing, importing or performing any other actions against a CVS server, the user is required to authenticate himself first by issuing a login command. With the $CVSROOT variable set as indicated, type the following:

```
# cvs login
Logging in to :pserver:user@hostname:2401/usr/local/cvsroot
CVS password:
```

Type in your CVS password, which will then be stored in a file in your home directory (.cvspass) for future use.

At that point you should be able to check out and commit software from the repository.

6.4.3 Checking out a Project

In order to begin work on a project contained in a CVS repository, you must first check the package out. This brings your working directory up-to-date (creating it if necessary) and creates a CVS directory in your working directory where CVS keeps information about the project locally.

To check out a project that is in the repository, simply issue the cvs command with the checkout option and the name of the project. For example, to check out the Fuzion project that was imported into the repository earlier, type:

```
# cvs checkout Fuzion
cvs server: Updating fuzion
U Fuzion/1.0.fuzion.pl
U Fuzion/em.dat
```

```
U Fuzion/fuzion.pl
U Fuzion/sk.dat
U Fuzion/tl.dat
U Fuzion/test.dat
```

CVS checks any existing copies of the project with the ones in the repository and updates all of the files on the user's machine.

The option `co` is an alias for the `checkout` option and they may be used interchangeably.

6.4.4 Finding the Status of a Project

After working on a project for a time, it may be necessary for you to determine which files in your working directory and which files in the repository have changed.

In order to accomplish this, you may use the `status` option of the CVS command. If this command is issued from the working directory without a project name, that project will be examined. If it is issued from another directory, you should include the project name to prevent CVS from trying to examine all of the subdirectories of the current directory for project information.

```
# cd fuzion
# cvs status
```

A partial example of the output is presented here:

```
===================================================================
File: 1.0.fuzion.pl     Status: Up-to-date

   Working revision:    1.1.1.1
   Repository revision: 1.1.1.1 /usr/local/cvsroot/fuzion/
1.0.fuzion.pl,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
===================================================================
File: fuzion.pl         Status: Locally Modified

   Working revision:    1.1.1.1
   Repository revision: 1.1.1.1 /usr/local/cvsroot/fuzion/
fuzion.pl,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
===================================================================
File: test.dat          Status: Needs Patch

   Working revision:    1.2
   Repository revision: 1.3     /usr/local/cvsroot/fuzion/
test.dat,v
```

```
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

As you can see, there are two files here that are out of sync with the repository. The first one is `fuzion.pl` that has been modified, but not checked back in; and the other is `test.dat`, which someone else has modified and returned to the repository.

At this point, you can use `diff` to determine the extent of the changes and determine the appropriate course of action.

If your local copy of a file has changed, and the version in the repository has changed from the last time that you checked it out, you will receive a message indicating that the files needed to be merged.

```
File: test.dat          Status: Needs Merge

   Working revision:    1.2
   Repository revision: 1.3     /usr/local/cvsroot/fuzion/
test.dat,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Again, you will want to determine the extent of the changes to the files and merge the two so that no changes will be lost when the file is checked back into the repository. Should you attempt to commit a change on a file when a newer version of the file exists on the server, CVS will complain and refuse to update the file.

```
cvs server: Up-to-date check failed for `test.dat'
cvs [server aborted]: correct above errors first!
cvs commit: saving log message in /tmp/cvsa15396
```

In the event of conflicts, you can use the `diff` option to examine the differences in the file and the `update` option to merge them. Both of these options are covered below.

6.4.5 Finding Differences

CVS can be used to determine the difference between a file in the repository and a file in the working directory. This is accomplished by using the `diff` option.

```
# cvs diff fuzion.pl
Index: fuzion.pl
============================================================
RCS file: /usr/local/cvsroot/fuzion/fuzion.pl,v
retrieving revision 1.1.1.1
diff -r1.1.1.1 fuzion.pl
134c134
< print "=" x 40 ;
---
> print "=" x 80 ;
```

If you are familiar with the UNIX `diff` command, you will recognize the output of this command. See Section 7.3 for more information on the output of the `diff` command.

As you can see, CVS checked the repository for the latest copy of the file and compared it with the one in the working directory. This output indicates that one line has been changed at line 134 (a formatting change to the print statement).

6.4.6 Resolving Conflicts

The `update` option will merge two conflicting files into one file and insert special markers into the file that help you determine where the differences are and resolve them.

```
# cvs update test.dat
RCS file: /usr/local/cvsroot/fuzion/test.dat,v
retrieving revision 1.4
retrieving revision 1.5
Merging differences between 1.4 and 1.5 into test.dat
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in test.dat
C test.dat
```

The resulting `test.dat` file is shown below, clipped for brevity;

```
# cat test.dat
aerosol
aerosolize
aerosols
aerospace
aesthetic
aesthetically
aesthetics
<<<<<<< test.dat
zebras
zenith
=======
zebras
zenith
zoological
zoologically
zoom
zooms
zoos
>>>>>>> 1.5
```

The <<<<<<< and >>>>>>> markers indicate which version of the file has had which edits made to it. The text between "`<<<<<<< test.dat`" and "`=======`" indicate text that was added from the time that the file was checked out until the time when the merge was done.

The text between the "`=======`" marker and the "`>>>>>>> 1.5`" exists in version 1.5 of the file (in the repository) but not in the local working copy.

The final merged and edited file is shown below and is ready to be checked back into the repository.

```
# cat test.dat
aerosol
aerosolize
aerosols
aerospace
aesthetic
aesthetically
aesthetics
zebras
zenith
zoological
zoologically
zoom
zooms
zoos
```

6.4.7 Checking the Project Back In

Once you have made changes to a file, it is ready to be checked back into the repository. This is done with the `commit` option. The `checkin` and `ci` options are similar to the `commit` option, but instead of simply synchronizing the working file with the one in the repository, `commit` also releases the file and sets certain flags that can be used if the watch functions of CVS are being used.

```
# cvs commit -m "Added information to test.dat" test.dat
```

If there are any further conflicts, CVS will notify you that the files need to be changed.

6.4.8 Adding Files to a Project

Files may be added to a project after it has been imported. This is done by specifying the `add` option name of the file or files to be added to the project.

For example:

```
# cvs add index.html
cvs server: scheduling file `index.html' for addition
cvs server: use 'cvs commit' to add this file permanently
# cvs commit -m "new file" index.html
RCS file: /usr/local/cvsroot/fuzion/index.html,v
done
Checking in index.html;
/usr/local/cvsroot/fuzion/index.html,v  <--  index.html
initial revision: 1.1
done
```

If the file is a binary file you must specify the –**kb** option in order for it to be properly imported into the project.

```
# cvs add -kb banner.jpg
```

6.4.9 Removing Files from a Project

If you no longer wish a file to be a part of a project, you can use CVS's `remove` option to rid yourself of that file. Prior to removing the file, however, it must be deleted from the working directory.

```
# cvs remove index.html
cvs server: file `index.html' still in working directory
cvs server: 1 file exists; remove it first
# rm index.html
# cvs remove index.html
cvs server: scheduling `index.html' for removal
cvs server: use 'cvs commit' to remove this file permanently
# cvs ci -m "Removed index.html" index.html
Removing index.html;
/usr/local/cvsroot/fuzion/index.html,v  <--  index.html
new revision: delete; previous revision: 1.3
done
```

If you wish to delete a file and remove it from the project in one step, you may add the –**f** flag to the remove option.

```
# cvs remove -f index.html
cvs server: scheduling `index.html' for removal
cvs server: use 'cvs commit' to remove this file permanently
# ls index.html
/bin/ls: index.html: No such file or directory
# cvs commit -m "Removed old files"
cvs commit: Examining .
Removing index.html;
/usr/local/cvsroot/fuzion/index.html,v  <--  index.html
new revision: delete; previous revision: 1.5
done
```

Prior to committing a remove to the repository, the file may be resurrected by issuing the add command.

```
# rm index.html
# cvs remove index.html
cvs server: scheduling `index.html' for removal
cvs server: use 'cvs commit' to remove this file permanently
# cvs add index.html
U index.html
cvs server: index.html, version 1.7, resurrected
```

Additionally, if you have not yet issued the `remove` command, you can retrieve a copy of the file from the repository by issuing an `update` command.

```
# rm index.html
# cvs update index.html
U index.html
```

6.4.10 Renaming Files within a Project

There is no direct method of renaming files within the CVS repository. In order to rename a file, you must remove the old name and add in the new one. Here is an example of renaming a file from `oldfile` to `newfile`.

```
# cvs status oldfile
=============================================================
File: oldfile           Status: Up-to-date

   Working revision:    1.1
   Repository revision: 1.1     /usr/local/cvsroot/fuzion/
oldfile,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

# mv oldfile newfile
# cvs remove oldfile
cvs server: scheduling `oldfile' for removal
cvs server: use 'cvs commit' to remove this file permanently
# cvs add newfile
cvs server: scheduling file `newfile' for addition
cvs server: use 'cvs commit' to add this file permanently
# cvs commit -m "Ren oldfile newfile" oldfile newfile
Removing oldfile;
/usr/local/cvsroot/fuzion/oldfile,v  <--  oldfile
new revision: delete; previous revision: 1.1
done
RCS file: /usr/local/cvsroot/fuzion/newfile,v
done
Checking in newfile;
/usr/local/cvsroot/fuzion/newfile,v  <--  newfile
initial revision: 1.1
done
```

6.4.11 Removing your Working Copy

Once you have committed your changes to the repository, or if you wish to abandon your changes, you can release the working copy prior to deleting it. If you have any files checked out that have not been updated with the repository, CVS will inform you and give you the option of releasing the working directory.

```
# cd /home/lcp
# cvs release fuzion
A banner.jpg
M fuzion.pl
You have [2] altered files in this repository.
Are you sure you want to release directory `fuzion': Y
# rm -r fuzion
```

6.4.12 Tags and Releases

One of the strengths of CVS is its ability to create a release of a particular project. A release is a snapshot of a project at a specific point in time and is generally referred to by a numeric designation. When we refer to CVS version 1.1.11p1, we are talking about a specific release of the application that will always remain the same. If changes are made to that version of the software, the version number will change by indicating either a patch level or an increment in the major or minor version number.

This designation can be accomplished with CVS by assigning a `tag` to a project. Tags may be named in any fashion, symbolically with a numerical system as mentioned, or by code names, etc. It is important, however, that once a methodology for naming releases is established that it be abided by in order to prevent confusion in the future.

To apply a tag to a project, issue the following command from within the project directory:

```
# cvs tag Release_1_1 .
cvs server: Tagging .
T 1.0.fuzion.pl
T banner.jpg
T em.dat
T fuzion.pl
T index.html
T newfile
T sk.dat
T tal.dat
T test.dat
```

This does not change any information in the files; it simply places symbolic tags in the CVS repository so that when Release_1_1 of the Fuzion project is requested, these versions of the files will be retrieved from the repository.

To check a specific release back out of the repository, issue the `checkout` command with the **–r** option and the name of the release.

```
# cvs checkout -r Release_1_1 fuzion
```

6.5 Introduction to jCVS

For those more comfortable working with a GUI rather than a command line CVS client, there is a Java client available that uses Java that enables you to access off of the functionality of the CVS repository with a few clicks.

jCVS was written by Timothy Gerard Endres and has been released under the GNU Public License (GPL) and may be found at http://www.jcvs.org.

6.5.1 System Requirements

As previously mentioned, jCVS will run on any platform supported by Java. The current release runs on Java version 1.2 or higher. The installed application is a little over 6MB with source code, so any computer capable of running Java and X should have not problems installing and running jCVS.

6.5.2 Installation Instructions

In order to run jCVS, you must first install the Java Runtime Environment. The latest and most complete runtime libraries available from Sun are http://java.sun.com/j2se/1.4/download.html and can be found for Linux, Solaris and Windows.

Download the installer for your platform and follow the instructions provided. This generally consists of running an installer. In the example provided below, the `jre` was installed in `/usr/local/jre`.

Next, locate the latest jCVS tarball application archive. These are available at http://www.jcvs.org/download.html for download. For the purpose of these examples, assume that the application will be installed in `/usr/local/jcvs`.

```
# cd /tmp
# wget http://www.jcvs.org/download/jcvs/jcvs-522.tgz
--14:14:46--  http://www.jcvs.org/download/jcvs/jcvs-522.tgz
           => `jcvs-522.tgz'
Connecting to www.jcvs.org:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 1,976,228 [application/x-tar]
```
[snip]
```
14:14:55 (226.81 KB/s) - `jcvs-522.tgz' saved [1976228/
1976228]
# cd /usr/local
# tar xzf /tmp/jcvs-522.tgz
# ln -s  jCVS-5.2.2 jcvs
# /usr/local/jre/bin/java -jar \
/usr/local/jcvs/jars/jcvsii.jar
```

6.5.3 Using jCVS

At this point you should be faced with the main jCVS screen. (See Figure 6-1.) The tabs along the top permit you to perform various CVS commands against one or more repositories. If you have problem accessing a repository, the `Test` tab will allow you to check basic connectivity and access rights.

Once you have checked projects out from a repository, you may transfer them to the WorkBench in order to have easy access to them. This is simply an area where all of your current projects can reside.

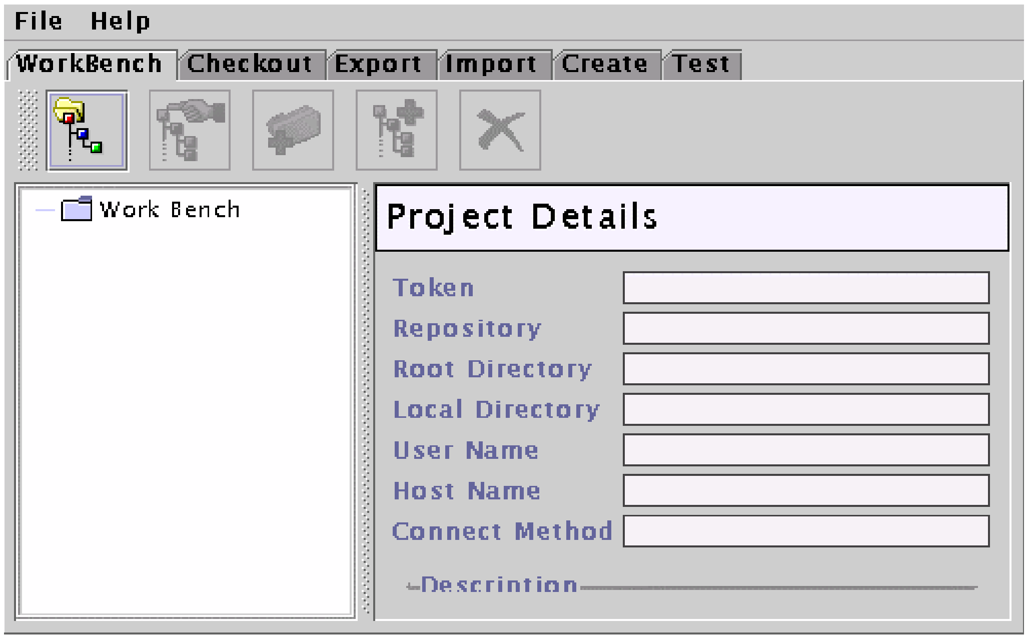


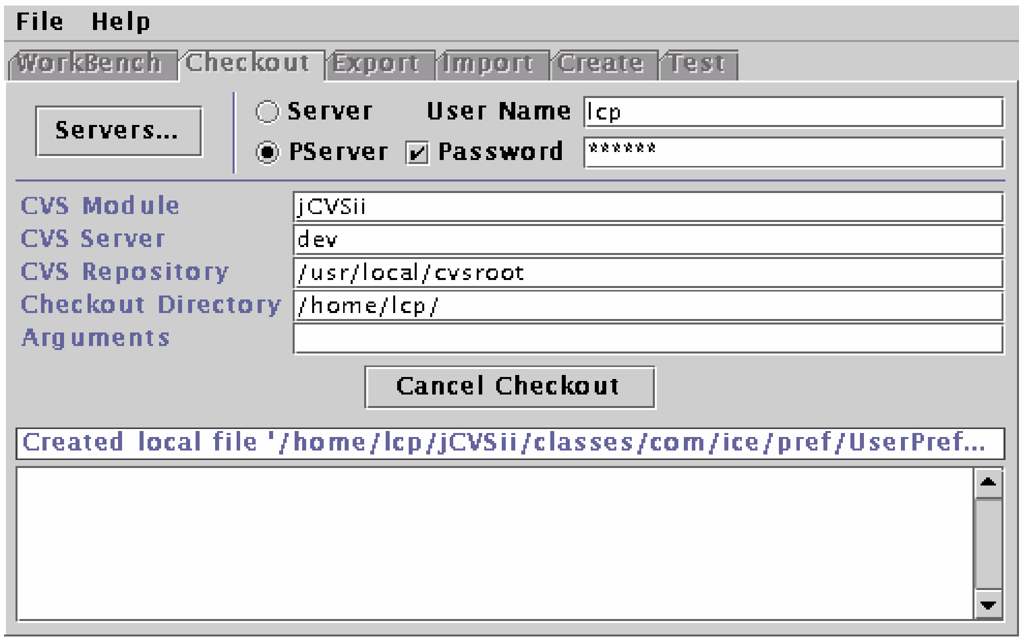
Figure 6-1 The jCVS WorkBench.

6.5.3.1 Configuring the jCVS Client

To begin, you will want to set up a few configuration options before beginning work on a project. Select **File | Edit Preferences** … from the main menu. Select **Global | Temp Directory** in order to specify a place on the computer to store temporary files.

6.5.3.2 Checking Out a Project

To begin working with jCVS, click on the `Checkout` tab. The screen shown in Figure 6-2 will appear and you will be able to checkout a project from your repository.



[Server] [✓ PServer]

User Name	lcp
Password [✓]	******
CVS Module	jCVSii
CVS Server	dev
CVS Repository	/usr/local/cvsroot
Checkout Directory	/home/lcp/
Arguments	

```
Created local file '/home/lcp/jCVSii/classes/com/ice/pref/UserPref...
```

Figure 6-2 Checking out a Project from a CVS Repository.

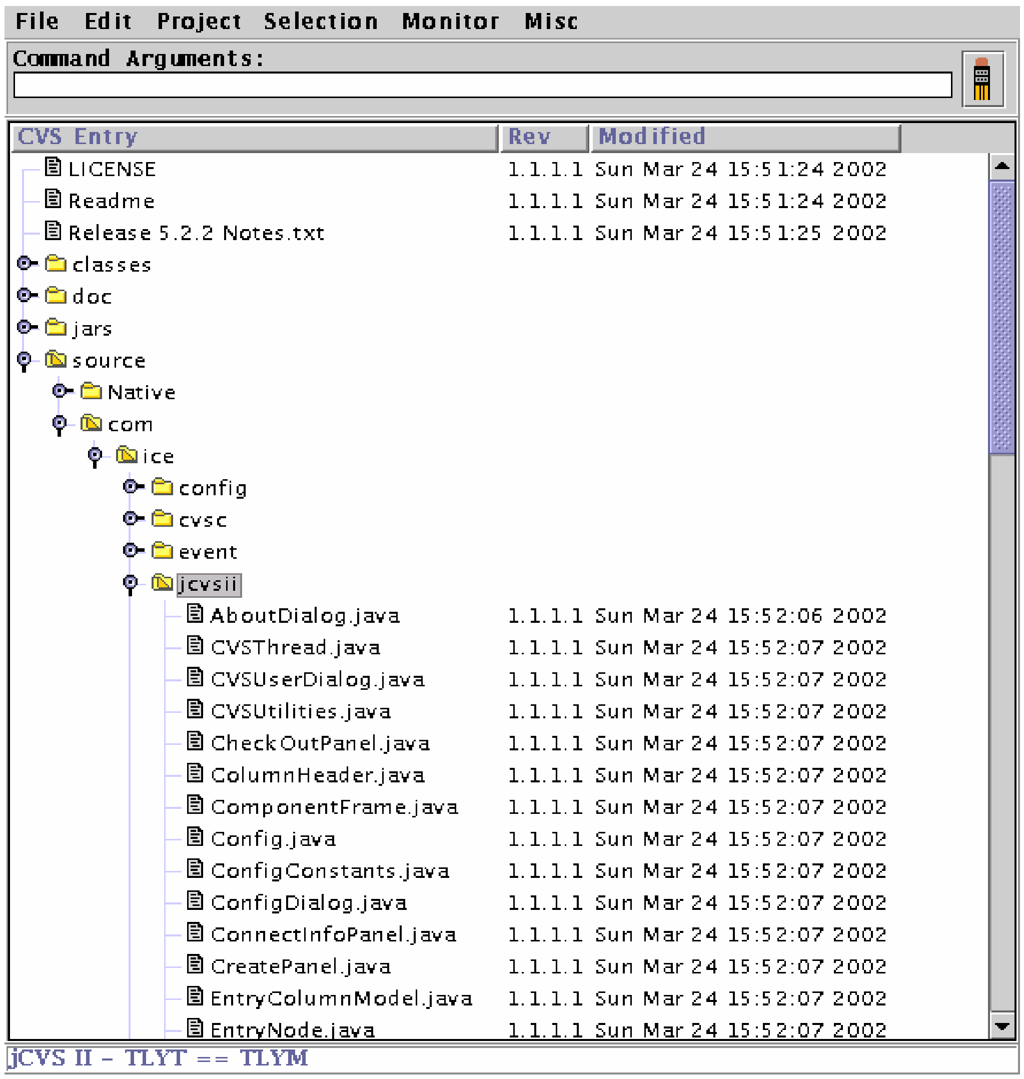
As you can see, the fields presented map to the command line arguments that are required in order to check out a project from a repository. In the example presented, I am checking out a project entitled `jCVSii` from the `dev` server.

The Arguments field can be used to enter any command arguments or options that do not have a specific entry on the form. The entry above would be equivalent to the following commands:

```
# cd /home/lcp
# export CVSRROOT=:pserver:lcp@dev:/usr/local/cvsroot/
# cvs login
# cvs checkout jCVSii
```

When the checkout is complete, a project window will be presented displaying the file structure of the project. See Figure 6-3.

The Project Window allows you full access to the entire range of CVS commands and option. Files may be edited and committed back to the repository, the entire project may be updated, you can view the difference between the project file and the same file in the repository, view the project log, and even resurrect files as we did earlier with the remove and update commands.



```
CVS Entry                          Rev      Modified
  LICENSE                          1.1.1.1  Sun Mar 24 15:51:24 2002
  Readme                           1.1.1.1  Sun Mar 24 15:51:24 2002
  Release 5.2.2 Notes.txt          1.1.1.1  Sun Mar 24 15:51:25 2002
  classes
  doc
  jars
  source
    Native
    com
      ice
        config
        cvsc
        event
        jcvsii
          AboutDialog.java         1.1.1.1  Sun Mar 24 15:52:06 2002
          CVSThread.java           1.1.1.1  Sun Mar 24 15:52:07 2002
          CVSUserDialog.java       1.1.1.1  Sun Mar 24 15:52:07 2002
          CVSUtilities.java        1.1.1.1  Sun Mar 24 15:52:07 2002
          CheckOutPanel.java       1.1.1.1  Sun Mar 24 15:52:07 2002
          ColumnHeader.java        1.1.1.1  Sun Mar 24 15:52:07 2002
          ComponentFrame.java      1.1.1.1  Sun Mar 24 15:52:07 2002
          Config.java              1.1.1.1  Sun Mar 24 15:52:07 2002
          ConfigConstants.java     1.1.1.1  Sun Mar 24 15:52:07 2002
          ConfigDialog.java        1.1.1.1  Sun Mar 24 15:52:07 2002
          ConnectInfoPanel.java    1.1.1.1  Sun Mar 24 15:52:07 2002
          CreatePanel.java         1.1.1.1  Sun Mar 24 15:52:07 2002
          EntryColumnModel.java    1.1.1.1  Sun Mar 24 15:52:07 2002
          EntryNode.java           1.1.1.1  Sun Mar 24 15:52:07 2002
jCVS II – TLYT == TLYM
```

Figure 6-3 The Project Window.

6.5.3.3 Setting up the Server Definitions File

Since, in a development environment, you will be accessing one or more CVS repositories on a regular basis, you can set them up so that some of the information is retained by the system and saves you some effort if you have to switch between repositories.

To do this, create a file in your home directory entitled `.jcvsdef` with the following format:

```
server.myserver=true
param.myserver.method=pserver
param.myserver.name=My CVS Server
param.myserver.module=fuzion
param.myserver.host=dev.mynetwork.net
param.myserver.user=lcp
param.myserver.repos=/usr/local/cvsroot
param.myserver.desc= \
My Server Name goes here.\
As well as a description of the \
available projects.
```

In this example, you would substitute the name of your CVS repository for `myserver`. Use a short nickname, not a fully qualified domain name (FQDN). Don't use periods in this name. The entry for `param.myserver.host` should be the FQDN for the CVS repository, or the IP Address.

If you are working on multiple projects, you can leave the `param.myserver.module` blank and it will not be filled in automatically. This works for all of the fields.

6.5.3.4 Actions and Verbs

Prior to actually editing the files in the project, you must associate each file type with the application that will be used to `Open` or `Edit` the file.

From the WorkBench screen, re-open the preferences. (**`File | Edit Preferences`** **…**) Then select **`Actions | Verb Commands`** from the menu on the left. This will present the screen shown in Figure 6-4.

Click on the **`New`** **…** button. This displays a dialog button asking for a key. The key is a combination of the file extension and an action that may be used with that type of file. For example, a C Language source code file is designated by the `.c` extension and may be edited or opened. The key to edit any C source code file would be "`.c.edit`".

The key to open a .jpeg graphic file would be "`.jpeg.open`".

To create a key to edit a file with a `.java` extension, enter `.java.edit` into the field in the dialog box. Click the `OK` button.

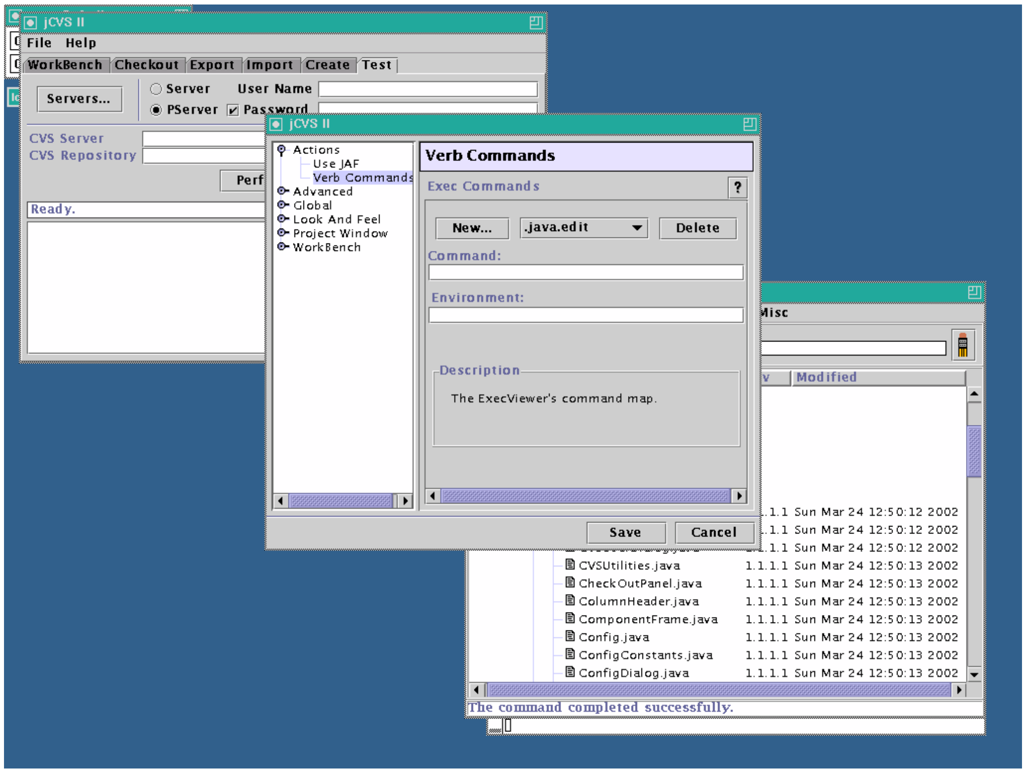
Default actions can be defined by adding a key with the definition of '`._DEF_.verb`'. For example, to create a default editing action for all files not specifically associated with another action, use the key '`._DEF_.edit`'.

In the `Command` field, type the command line that you would use to perform the action associated with the key that you have just entered. The name of the file to be edited is substituted on the command line for the variable `$FILE`.

If, for example, you wished to use `gedit`, a simple Gnome editor, enter

```
/usr/bin/gedit $FILE
```

in the `Command` field.



Ready.

The ExecViewer's command map.

```
.1.1     Sun Mar 24 12:50:12 2002
.1.1     Sun Mar 24 12:50:12 2002
.1.1     Sun Mar 24 12:50:12 2002
CVSUtilities.java        1.1.1.1  Sun Mar 24 12:50:13 2002
CheckOutPanel.java       1.1.1.1  Sun Mar 24 12:50:13 2002
ColumnHeader.java        1.1.1.1  Sun Mar 24 12:50:13 2002
ComponentFrame.java      1.1.1.1  Sun Mar 24 12:50:13 2002
Config.java              1.1.1.1  Sun Mar 24 12:50:13 2002
ConfigConstants.java     1.1.1.1  Sun Mar 24 12:50:13 2002
ConfigDialog.java        1.1.1.1  Sun Mar 24 12:50:13 2002
```

The command completed successfully.

Figure 6-4 Adding verbs to jCVS.

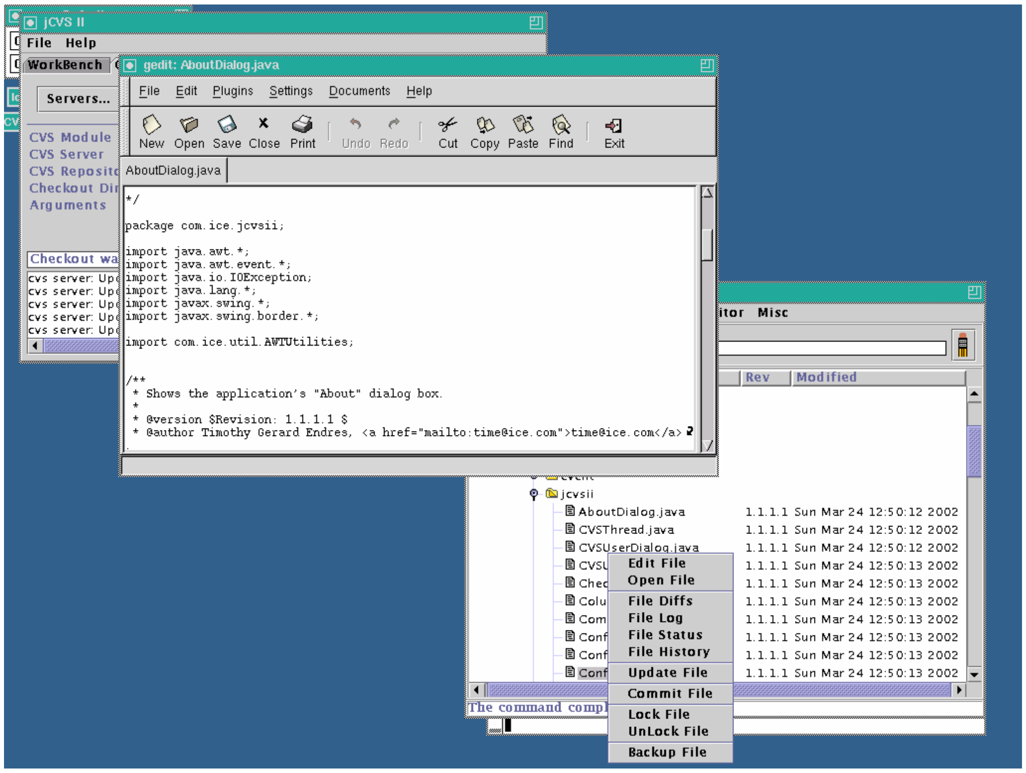
6.5.3.5 Editing Projects

Once you have associated various file extensions with specific actions and helper applications, you are ready to begin working with your project.

By highlighting a file and right-clicking on it, you will display a menu of options for that file. If you select `Edit` from the menu that is displayed, jCVS will launch the application that you have defined for that action as shown in Figure 6-5.

The icons of files that have been modified after being checked out of the repository are shown in red for easy identification.

Running a `diff` of a file, or updating it, is as simple as selecting the appropriate option from the menu.



```
*/

package com.ice.jcvsii;

import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import java.lang.*;
import javax.swing.*;
import javax.swing.border.*;

import com.ice.util.AWTUtilities;


/**
 * Shows the application's "About" dialog box.
 *
 * @version $Revision: 1.1.1.1 $
 * @author Timothy Gerard Endres, <a href="mailto:time@ice.com">time@ice.com</a>
```

```
cvs server: Upd
cvs server: Upd
cvs server: Upd
cvs server: Upd
cvs server: Upd
cvs server: Upd
```

```
jcvsii
  AboutDialog.java      1.1.1.1 Sun Mar 24 12:50:12 2002
  CVSThread.java        1.1.1.1 Sun Mar 24 12:50:12 2002
  CVSUserDialog.java    1.1.1.1 Sun Mar 24 12:50:12 2002
  CVSU                  1.1.1.1 Sun Mar 24 12:50:13 2002
  Chec                  1.1.1.1 Sun Mar 24 12:50:13 2002
  Colu                  1.1.1.1 Sun Mar 24 12:50:13 2002
  Com                   1.1.1.1 Sun Mar 24 12:50:13 2002
  Conf                  1.1.1.1 Sun Mar 24 12:50:13 2002
  Conf                  1.1.1.1 Sun Mar 24 12:50:13 2002
  Conf                  1.1.1.1 Sun Mar 24 12:50:13 2002
```

Figure 6-5 Editing source code.

6.5.3.6 Working with the WorkBench

In order to save a project to the WorkBench, you must first select a folder to save it in. You can organize your projects by adding additional folders to the WorkBench as needed. These can be divided by Vendor, Client or any other method that you choose.

When you have selected a folder for the project, in the Project Window, select **`File | Add to WorkBench...`** Then fill in the details of the project and press `OK`.

Opening a project on the WorkBench is accomplished by double-clicking on the name of the project. This opens the local copy.

To remove a project from the WorkBench, you will want to release the project from the Project screen (**`Project | Release`**) and then delete the project from the WorkBench.

6.6 Using Emacs with CVS

Until recently, (version 1.10) CVS has shipped with `pcl-cvs`, an interface that allowed Emacs users to integrate CVS into the Emacs environment. The latest version of `pcl-cvs` (2.9) is dated 4/15/2000 and is available at ftp://rum.cs.yale.edu/pub/monnier/pcl-cvs/pcl-cvs-2.9.9.tar.gz You will also need to retrieve the `elib` library found at ftp://ftp.lysator.liu.se/pub/emacs/elib-1.0.tar.gz.

6.6.1 Installing pcl-cvs

The `elib` library must be installed first. To do this, unpack the tarball:

```
# tar xvzf elib-1.0.tar.gz
```

Change directories into the newly created file and edit the Makefile to reflect the RedHat environment. Changes need to be made to two lines in the Makefile under RedHat 7.2. Other versions of Linux may or may not require any changes. The original lines and the correction are given below:

Original:

```
prefix = /usr/local
infodir = $(prefix)/info
```

Edited:

```
prefix = /usr
infodir = $(prefix)/share/info
```

Once these changes are made, you may compile the `elib` library by issuing the following commands as root:

```
# make
# make install
```

Once you have installed the `elib` library, you may install the `pcl-cvs` application the exact same way, by making the above changes to the Makefile and then compiling with `make`.

6.6.2 Using pcl-cvs

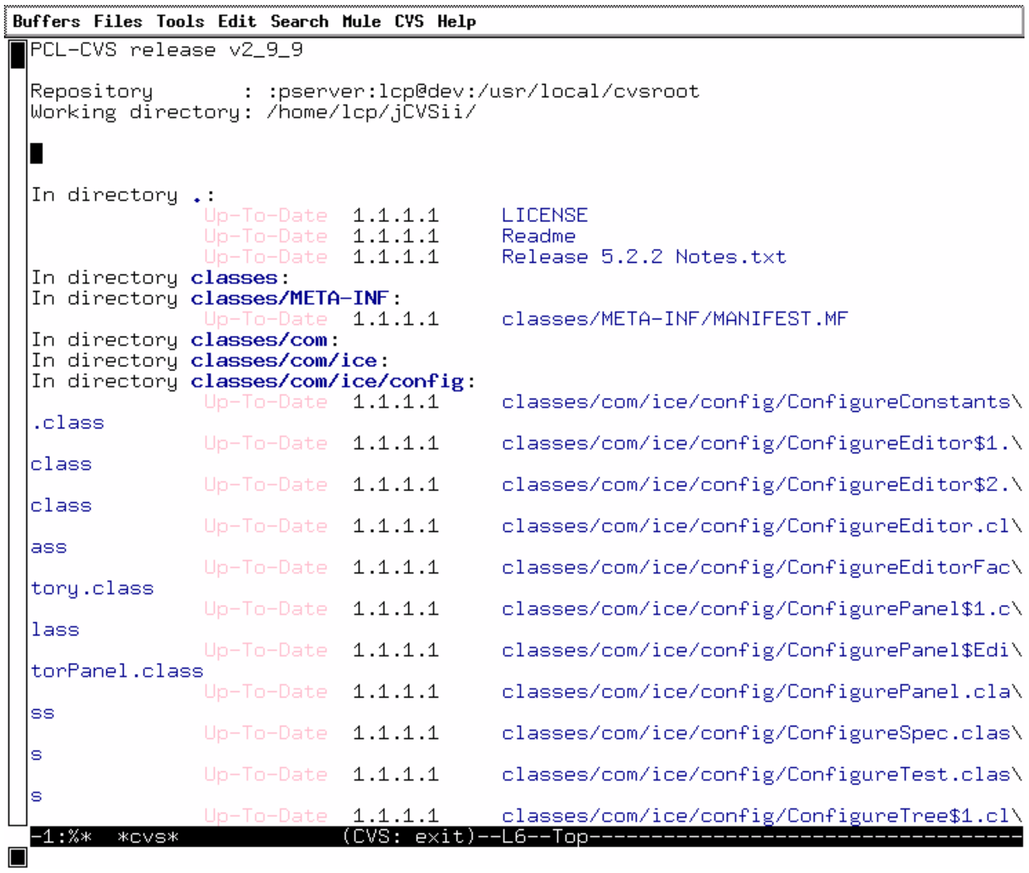
Complete documentation for `pcl-cvs` is available with the installed appliations in `.info` and `.tex` formats. It is also available on the World Wide Web at http://www.delorie.com/gnu/docs/cvs/pcl-cvs_toc.html.

While you will still have to check out the project from the command line, `pcl-cvs` provides you with the ability to compare files, commit changes and perform many other CVS functions from within Emacs.

If you are unfamiliar with Emacs, please refer to Chapter 2 for more detailed information about working with this editor.

As an introduction to working the CVS through Emacs, change into a CVS project directory and start Emacs. Make sure that you have the `CVSROOT` environment variable set up properly.

To begin with, let's get the status on our current project. Type in **`M-x cvs-status`** and press enter to accept the directory information that Emacs presents. After checking with CVS, Emacs should display something similar to Figure 6-6.



```
PCL-CVS release v2_9_9

Repository       : :pserver:lcp@dev:/usr/local/cvsroot
Working directory: /home/lcp/jCVSii/


In directory .:
              Up-To-Date  1.1.1.1     LICENSE
              Up-To-Date  1.1.1.1     Readme
              Up-To-Date  1.1.1.1     Release 5.2.2 Notes.txt
In directory classes:
In directory classes/META-INF:
              Up-To-Date  1.1.1.1     classes/META-INF/MANIFEST.MF
In directory classes/com:
In directory classes/com/ice:
In directory classes/com/ice/config:
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigureConstants\
.class
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigureEditor$1.\
class
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigureEditor$2.\
class
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigureEditor.cl\
ass
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigureEditorFac\
tory.class
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigurePanel$1.c\
lass
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigurePanel$Edi\
torPanel.class
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigurePanel.cla\
ss
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigureSpec.clas\
s
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigureTest.clas\
s
              Up-To-Date  1.1.1.1     classes/com/ice/config/ConfigureTree$1.cl\
-1:%*  *cvs*            (CVS: exit)--L6--Top---------------------------------
```

Figure 6-6 CVS Project status in Emacs with `pcl-cvs`.

Files listed in the status screen may be edited by moving the cursor to the appropriate line and pressing the **`o`** key. This will load the file in question into another window for editing. Pressing **`f`** instead of **`o`** will result in the file being opened in the current window and replacing the status information.

When the edited file is saved with changes, the status window will be updated to reflect this and the file will be shown as being modified.

To check a file back into the repository from Emacs, you need to move the cursor to the line with the file name on it, and press the **`c`** key. This will open a new window in which you can type the log information that reflects the changes that you have made to the file. Press C-c twice (**`C-c C-c`**) to save the log information and continue checking the file in. You will be asked to confirm the action prior to the file being updated in the repository.

The **m** key will mark multiple files (indicated by an asterisk) so that they can be committed or have other actions taken on them at the same time. The **u** key can remove the mark from a file.

There are more options available within `pcl-cvs`, but this should give you enough information to begin using the application productively.

6.7 Secure remote access with CVS

The main goal of CVS is to provide a central repository for the storage of development projects and to provide access to and management tools for those projects. CVS by its nature is not a secure protocol.

Passwords, source code files and project data are transmitted across the network in clear text and available to anyone along the data path that cares to eavesdrop and analyze the data collected.

Various developers, aware of the inherent insecurity of the CVS protocol, have started using Secure Shell (ssh) to provide an additional layer of security to the development process. This chapter describes how to set up ssh on the clients and the server in order to ensure that code, data and authentication information are handled in a secure manner.

6.7.1 Secure Shell Access

When describing remote access methods to use for CVS earlier, the method for setting up rsh access to the server was intentionally omitted. The protocol used by rsh is inherently insecure in that it transmits all information across the network in plain text and encourages trust relationships to be set up between computers that can lead to system compromises.

Secure Shell (ssh) is a replacement for rsh that uses client and server authentication as well as strong cryptography. It is a little more difficult to set up ssh access to a CVS repository than to set up rsh access, but the security advantages outweigh any additional time and effort in setting up the server.

The first step is to ensure that the CVS server has ssh installed and properly configured. Ssh may be obtained from several sources. OpenSSH (http://www.openssh.org) is an open source implementation of the protocol that is widely used. This is also what ships with RedHat Linux.

Using ssh with CVS uses RSA Authentication and you need to make sure that this is turned on in the ssh configuration file. This configuration file is `/etc/ssh/sshd_config`.

Locate the RSA Authentication and configuration and make sure that it is turned on. It should read:

```
RSAAuthentication yes
```

This should complete the ssh configuration on the server.

Now log into the client machines that are going to be accessing the CVS repository. Each user that wishes to access the repository must:

1. Have a shell account on the CVS server.
2. Create a public/private key pair on the client.
3. Transfer the public portion of the key to the CVS server.
4. Set up the CVS client to use this new method.

Depending on how the CVS server is configured, it may be possible to set up ssh to bypass the requirement of typing in your password every time that you interact with the CVS repository.

NOTE The key to logging in without having to type in the passphrase each time is to specify a blank password during the key generation process. While this does permit you to log into the server without supplying a passphrase, anyone who has access to your account will be able to do the same. Make sure never to leave your workstation unlocked and unattended.

Assuming that the accounts have already been set up on the CVS server, the next step is to generate the public/private key pair. This is accomplished by using the ssh-keygen program.

To generate the key pair, issue the following command (this may result in a DSA key with openssh3.):

```
# ssh-keygen -N ""
Generating public/private rsa1 key pair.
Enter file in which to save the key (/root/.ssh/identity):
<CR>
Your identification has been saved in /root/.ssh/identity.
Your public key has been saved in /root/.ssh/identity.pub.
The key fingerprint is:
32:66:ea:74:4a:7d:23:78:57:bc:90:12:a3:da:78:20 lcp@client
```

The –**N ""** option tells the program to uses a null passphrase to protect the keys.

As indicated by the program messages, this creates two files in the user's `$HOME/.ssh` directory. The first one, `identity`, is your private key. This should be kept secure and should not be shared. The second, `identity.pub`, as you may have guessed, is the public key. The contents of this file need to be distributed to every machine that you wish to log into and access via ssh.

To accomplish this, you can use ssh itself to copy the file between machines as follows:

```
# cd ~/.ssh
# scp identity.pub user@cvsserver:.ssh/`hostname`.pub
```

Replace `user@cvsserver` with your username on the CVS server. For example if your username were `jon` and the CVS server was named `dev`, you would replace this with `jon@dev`. You could also use the FQDN or IP Address in place of `cvsserver`. Also note that the punctuation marks surrounding the `hostname` are backticks.

Next, you need to log into the CVS server and append the newly created `.pub` files to a file called `authorized_keys`. The `authorized_keys` holds the authentication keys for each user/machine combination that is permitted to use RSA Authentication to log in.

```
# ssh -l username cvsserver
# cd .ssh
# cat client.pub >> authorized_keys
```

While `username` and `cvsserver` are the same as before, `client.pub` should be replaced with the name of the machine that you generated the keys on.

At this point ssh should be set up and you should be able to move ssh into the CVS repository without having to enter a password or passphrase.

The next step is to set up CVS to use this new method of connecting to the repository. This is accomplished by set two environment variables. The first you should already be familiar with; that is `CVSROOT`.

When using ssh instead of accessing the server locally or via `pserver`, the command to set up the `CVSROOT` variable should look something like this:

```
# export CVSROOT=:ext:user@hostname:/usr/local/cvsroot
```

The second environment variable that needs to be set up is `CVS_RSH`. This is used with the `:ext:` method to specify which command should be used to log into the remote server. This should be set to `ssh`. You may need to specify the full path to the ssh application if `ssh` is not in your `$PATH`.

```
# export CVS_RSH=ssh
```

At this point you should be able to use CVS via ssh.

You should be aware that each project, when it is checked out, keeps track of the method that it used to check out that project. This is kept in the CVS directory of the project in a file called `Root`. If you have any projects checked out when switching access methods, you will want to check the projects back in and release them before switching; or manually edit the files for each project to inform CVS of the change in access.

Supplying the **–d** option on the command line to change the access method would also work.

6.8 References and Resources

1. The CVS home page at http://www.cvshome.org
2. Complete CVS documentation at http://www.cvshome.org/docs/manual/
3. The jCVS home page at http://www.jcvs.org
4. The OpenSSH home at http://www.openssh.org