This laboratory explores document version control systems commonly used on UNIX systems.

Purpose

What is a document version control system? Why do we need them? What is the problem that document version control systems solve?

Let's say you have to write a paper for a class. You write a first draft of the paper, then change it into the paper you hand in. Then, the teacher corrects it, asks you to incorporate the corrections into your paper, and also asks for you to add some additional content. You first make the corrections. You then realize that you want to add some material that was in your first draft. However, you no longer have a copy of your first draft because you overwrote it with the contents of the paper you handed in.

Document version control systems allow you to systematically save copies of documents at various stages of development. In addition, they provide methods for:

- retrieving any version of a given document
- determining the differences between any two versions of a given document
- attaching notes summarizing the changes going into each version, automatically including with that the date and time each version was created
- relieving that wide-eyed, terrified, *oh-S**** look that you get when you accidentally delete a file for which you have no backup
- inserting version information within the text of each version of a document
- coordinating multiple documents by using version labeling

As a result, document version control systems are part of a disciplined approach to developing documents.

These systems are important in software development, and you will likely find them valuable in systems administration as well.

In most engineering organizations in which I worked as a software developer, I used these systems. It was often important for maintenance engineers to rebuild a copy of an old version of software from the source code, so that fixes to bugs could be developed and applied to the code. This made it necessary for development engineers to create versions of the source code at every release to support the efforts of maintenance engineers. It's also important to track what changes one makes to software code and why those changes were made. There's always the time that comes that subsequent to major changes being made in program code, someone discovers months later that a feature now doesn't work. Using a document version control system means that you can go back and see exactly how and why files were changed, and be able to roll these changes back so that you can identify what changes broken the functionality.

I use these systems now for just about every sort of document I write, including:

- source code for computer programs
- configuration files for my computers

- files that I put on my web site
- papers like this

I remember what Brian Mills said, when he saw the triax cable in messy piles from Wednesday's strike after the shooting of *Kiss Me, Kate,* something like “if I had left cables around after striking that were this messy, I'd get fired”. Using a version control system, to me, is like using over-and-under to wind cables when striking a set. It's something you don't argue with, even though it requires more effort, because it demands that effort in a less-critical time period of the production, and, as a trade-off, it reduces risk in more-critical time periods of the production.

Surprisingly, I've had to initiate moves to use document version control systems in a few engineering organizations, once only after overcoming significant resistance from the Vice President of Engineering.

Next, you'll perform exercises to illustrate the features and benefits of document version control systems.

Exercises

We'll use *rcs* in this laboratory.

Run the following:

```
% ~bmcdonald/class/lab07.sh
```

Next, change your working directory to the lab07 subdirectory that was just created for you.

```
% cd ~/lab07
```

In your directory you have one file, named *startFile*.

Preparation

Create a subdirectory named RCS within your lab07 directory by:

```
% mkdir RCS
```

The *RCS* directory you just created will contain files used by RCS to manage versions of documents you put under the control of RCS.

Put the File Under RCS Control

View the contents of the file by:

```
% cat startFile
```

View the permissions for this file by:

```
% ls -l startFile
```

Notice that the file is writeable by you.

Put the file under control of RCS by using the RCS command *ci*:

```
% ci -i -u startFile
```

The *ci* command will ask you for a comment that applies to the entire file. Enter the following:

```
>> Sample RCS file<enter>
```

```
>> .<enter>
```

The *ci* command stands for *check in.* For a file not already under RCS control, the ci command adds the file to RCS control.

View the permissions for the file:

```
% ls -l startFile
```

Notice that the file is now marked as read-only for you.

Version Control Files

Since you created an RCS subdirectory within the *lab07* directory, when RCS creates a RCS version control file for the file *startFile*, it puts the version control file within this directory. This version control file contains all the information needed to recreate the file, so that even if you deleted the original file, you could get it back using RCS.

Never edit the version control file directly, or you may lose the ability to use it. Some document version control systems calculate a *checksum* for each version control file and includes it within each respective version control file, as part of an integrity check against inadvertent changes in the files. If you edit a version control file directly, the checksum will no longer match and version control systems will report the error to you and will likely refuse to work with the file. The version control system may provide a utility to repair the version control file, or you may have to restore the file from a backup.

Deleting and Recreating a Controlled File

First, delete the file:

```
% rm startFile
```

Type `y` and press Enter to confirm that you want to delete the file even though it is marked read-only.

Now look for the file:

```
% ls
```

You shouldn't see anything except for the RCS subdirectory.

Look in the RCS subdirectory:

```
% ls RCS
```

You should see the control file, which will be named similarly to the original file.

Now, retrieve the file you just deleted, using the RCS command *co*.

```
% co startFile
```

Look for the file:

```
% ls
```

You should see the file.

Look at its permissions:

```
% ls -l startFile
```

It is marked as read-only.

Viewing Information on the Controlled File

In order to get information on this controlled file, use the RCS command *rlog*:

```
% rlog startFile
```

This shows you information on the status of the file, including version numbers of all versions.

Making Changes to the File

In order to make changes to a file under RC S control, you must check out the file using the RCS command *co* (stands for *check out*).

```
% co -l startFile
```

The `-l` argument (that's an ell, not a one) tells RCS to lock the file for you and you alone.

Look at the file's permissions:

```
% ls -l startFile
```

The file should be marked read-write for you only.

Look at the log:

```
% rlog startFile
```

The log should indicate that you have the file checked out.

Next, vi the file:

```
% vi startFile
```

Append a new line to the file by:

1. pressing the `G` key to move the cursor to the last line,
2. pressing the `o` key to open a new line after the cursor,
3. typing a line of a few words of whatever you want (without pressing Enter)
4. pressing the escape key to complete the addition of the new line
5. typing `:wq` and hit enter to save your work and exit

RCS allows you to review the changes you have made to any file you have checked out, by using the RCS command *rcsdiff*:

```
% rcsdiff startFile
```

You should see a line marked something like:

```
3a4
```

followed by a line that contains the text you just added.

Checking In Changes to the File

Check in your changes using the RCS command ci:

```
% ci -u startFile
```

The *ci* command will prompt you for a comment that will be associated with the new version of this file. Type in something; you can type in multiple lines by hitting the Enter key after each one. To indicate that you are done typing your comment, hit the dot/period (`.`) key and press Enter. The *ci* command will then create a new version of the file with your changes.

Look at the status of the controlled file:

```
% rlog startFile
```

You should see two versions of the file. The first version is the one when the file was placed under control or RCS, the second version is the one you just created.

Look at the file's permissions:

```
% ls -l startFile
```

The file should be marked as read-only.

Examining Differences Between Versions

To see the differences between the versions, run the RCS command *rcsdiff* as follows:

```
% rcsdiff -r1.1 -r1.2 startFile
```

The first two arguments indicate the numbers of the versions that are to be compared. Do not put spaces between the `-r` and the version number.

You should see the same result as you saw before when running the *rcsdiff* command.

Adding RCS Keywords to the File

RCS provides the ability to add keywords to the files it controls. We'll do this now to show you how this works.

Check out the file:

```
% co -l startFile
```

Edit the file:

```
% vi startFile
```

Type `O` (shift-o) to open a new line on the first line of the file. Type in the following and hit the escape key when done:

```
$Revision$ $Date$
```

Save the file and exit *vi* by:

```
:wq
```

Verify the changes you have made to the file:

```
% rcsdiff startFile
```

Check in your changes:

```
% ci -u startFile
```

Enter the following for the comment:

```
Added RCS keywords.
```

Hit the enter key, then hit the dot/period key, then the enter key again.

Look at the contents of your file:

```
% cat startFile
```

Notice that the `$Revision$` string now contains the actual version of the file (probably 1.2), and the `$Date$` string now contains the date on which the version was created.

For a list of keywords available in RCS, run the following:

```
% man ident
```

Labeling the File Version

It's useful to be able to add a label to a version of a file. A label is a text string that you associate with a particular version of a file. This is especially useful when working with multiple files, where you want to associate together different version numbers of different files. For example, you may wish to associate version 1.2 of file A with version 1.3 of file B; this can be done by labeling the respective versions of the respective files with the same label.

Label the current version of your file by:

```
% co -l startFile
```

```
% ci -nNewLabel -u startFile
```

Don't use spaces within labels. Don't put a space between the `-n` and `NewLabel`.

Dump the file's version control log again:

```
% rlog startFile
```

Notice the entry for the label you created.

Retrieving a Previous Version of a File

You can retrieve old versions of a file under version control by:

```
% co -p1.1 startFile >> startFile1.1
```

This retrieves version 1.1 of *startFile* and creates the file *startFile1.1* to contain that version.

Check the difference between the two files by:

```
% diff startFile startFile1.1
```

The *diff* utility is what RCS uses in the *rcsdiff* command.

Additional Information

For additional information on RCS, run the following:

```
% man rcsintro
```

Example Document Version Control Systems

There are numerous examples of document version control systems that run on UNIX systems.

- RCS – an acronym for Revision Control System.
- SCCS – an acronym for Source Code Control System.
- CVS – an acronym for Concurrent Versioning System. CVS provides an easier mechanism to remotely manage document version repositories on different machines. CVS is based on RCS.
- Subversion – from the project's web page: "An open-source revision control system, which aims to be a compelling replacement for CVS". It includes the ability to version multiple documents in a single transaction.
- Perforce – a commercial product that has significant advantages over many commonly-used document version control systems, including the ability to version multiple documents in a single transaction.